Active Reliable Multicast

Li-wei H. Lehman, Stephen J. Garland, and David L. Tennenhouse
MIT Laboratory for Computer Science

Abstract—**This paper presents a novel loss recovery scheme, Active Reliable Multicast (ARM), for large-scale reliable multicast. ARM is "active" in that routers in the multicast tree play an active role in loss recovery. Additionally, ARM utilizes soft-state storage within the network to improve performance and scalability. In the upstream direction, routers suppress duplicate NACKs from multiple receivers to control the implosion problem. By suppressing duplicate NACKs, ARM also lessens the traffic that propagates back through the network. In the downstream direction, routers limit the delivery of repair packets to receivers experiencing loss, thereby reducing network bandwidth consumption. Finally, to reduce wide-area recovery latency and to distribute the retransmission load, routers cache multicast data on a "best-effort" basis. ARM is flexible and robust in that it does not require all nodes to be active, nor does it require any specific router or receiver to perform loss recovery. Analysis and simulation results show that ARM yields significant benefits even when less than half the routers within the multicast tree can perform ARM processing.**

Keywords— **Reliable multicast, active networks, soft state, distributed network algorithms, protocol design and analysis, NACK implosion.**

I. Introduction

Reliable multicast over the Internet is a difficult problem. Both the sender and the network have a limited capacity for responding to reports of data loss. Simultaneous retransmission requests from large numbers of receivers can lead to sender and network overload, causing the well-known NACK implosion problem. Additionally, receivers in a multicast group may experience widely different packet loss rates depending on their locations in the multicast tree. Having the sender retransmit to the entire group when only a small subset of the receivers experience losses wastes network bandwidth and degrades overall performance. In order to scale loss recovery to large multicast groups, efficient mechanisms are needed to control NACK implosions, to distribute the load for retransmissions, and to limit the delivery scope of retransmitted packets.

Another challenge in Internet-based reliable multicast concerns the frequent group membership changes due to new subscriptions, unsubscriptions, and disconnected links. Such changes make it difficult to designate a subset of the receivers as proxies for the sender in retransmitting lost packets. A robust loss recovery scheme must cope with dynamic group membership changes.

Existing schemes provide only partial solutions to the above problems. SRM [1] provides a good solution for NACK implosion, distributes loss recovery to all members in the group, and is robust with respect to changes in group membership or topology. However, its timer-based implosion control mechanism increases recovery latency, and it has problems with duplicate requests and repairs. Additionally, local recovery is still an open issue for SRM. Hierarchical approaches such as LBRM, TMTP and RMTP [2], [3], [4] provide only approximate solutions to scoped recovery, and they do not always shield bottleneck links from unnecessary request and repair traffic. Moreover, they are less fault-tolerant and robust to topology changes, because they rely on designated receivers or loggers to perform retransmissions.

In this paper, we present a novel loss recovery scheme, Active Reliable Multicast (ARM), for large-scale reliable multicast. ARM utilizes intermediate routers to protect the sender and network bandwidth from unnecessary feedback and repair traffic. ARM routers employ three error recovery strategies. First, they suppress duplicate NACKs to control the implosion problem. By reducing the number of NACKs, ARM minimizes the amount of feedback traffic that need to cross the bottleneck links. Second, a router-based local recovery scheme is used to reduce the end-to-end wide-area latency and to distribute the load of retransmissions. More specifically, routers at strategic locations perform "best-effort" caching of multicast data for possible retransmission; as NACKs traverse upstream towards the original source of the data packet, any router can perform retransmission if the request packet is in its cache. Router-based retransmissions allow receiving end-hosts to recover quickly from packet losses on a "local" basis without imposing an unnecessary recovery overhead on the sender or the entire group. Finally, to reduce network bandwidth consumption, routers use partial multicasting to limit the scope of retransmitted data.

Most existing end-to-end solutions require multicast members to know the group topology or the relative locations of other receivers within the multicast tree. SRM relies on topology information to set its timer values, and hierarchical approaches require receivers to locate their designated representatives. ARM is robust to group topology changes, since it does not rely on any particular router or receiver to perform loss recovery. Receivers are not required to know about the group topology, and they are not required to buffer data for retransmission. The ultimate responsibility for retransmission lies with the sender. However, routers allow ARM to scale by suppressing duplicate NACKs and by performing "best-effort" local recovery from their soft state cache.

Our approach is motivated by, but does not depend on, the recently proposed Active Networks technology [5]. In our network model, active routers perform customized computation based on multicast data packet types. Further, intermediate active routers provide "best-effort" soft-state storage. ARM does not require all routers to be active. It degrades gracefully as the network resources available for active processing decrease. Analysis and simulation results show that, with moderate network-based processing and storage overhead, ARM significantly improves end-to-end performance and network efficiency.

In the next section, we present our assumptions about the network model and their impact on ARM. Section III describes the design of ARM. Section IV describes a prototype implementation. Section V discusses simulation results that compare ARM

Research supported in part by DARPA, monitored by the Office of Naval Research, under contract No. N66001-96-C-8522.

performance with SRM. The last sections present our conclusions and discuss future directions.

II. The Network Model

Our network model is similar to traditional packet networks in several aspects. We assume that network resources are multiplexed amongst multiple traffic flows, and that the network provides a "best-effort" service model. In particular, we assume that the underlying network is unreliable, and that packets can be lost, duplicated, or delayed. Therefore, end-points must ultimately be responsible for the reliable transport of data packets.

Our network model differs from the traditional one in that end-points can take advantage of network-based processing and storage to improve end-to-end performance and scalability for certain applications. Some intermediate routers, called active nodes, provide soft-state storage and perform customized computation based on different packet types.

We assume that each active node provides a fixed amount of "best-effort" soft-state storage. Storage is "best-effort" because an active node will cache an item for a specified lifetime as long as the node has enough cache capacity. End points specify the useful lifetime, or TTL (time-to-live), of cached items, because this lifetime is most likely application- or protocol-specific. Active nodes flush their caches periodically to remove items with expired lifetimes.

We make no assumptions about how many routers are active or about the size of the caches at active routers. While ARM utilizes active nodes to enhance error recovery, it does not rely on active nodes to guarantee that error recovery occurs. As for the "soft state" concept introduced by Clark [6], ARM is designed so that state information stored at routers can be flushed at anytime without permanent disruption of the end-to-end communications process. ARM operates correctly even in the face of router failures: end points compensate for flushed caches and router failures by using a combination of timeouts and retransmissions.

We assume that the network provides IP-multicast style multicast routing [7], [8], in which a tree rooted at the sender is formed to deliver multicast packets. Additionally, for simplicity, we assume that the paths for multicast routing correspond to the reverse paths for unicast routing. This is because ARM uses receiver-generated NACKs to set up subscription information for scoped retransmission; for this to be effective, a NACK travelling upstream towards the source must pass through routers on the reverse path of the multicast repair packet. This assumption can be eliminated by routing NACKs back up the multicast tree instead of by unicast routing.

III. Active Reliable Multicast

ARM is a receiver-reliable, NACK-based scheme in which receivers are responsible for detecting and requesting lost packets. Each data packet is labeled by a unique sequence number. Receivers detect losses primarily by sequence gaps in the data packets. An implication of this loss detection scheme is that receivers with a shorter latency to the source are likely to detect losses before receivers farther away. For interactive applications, periodically generated session or heartbeat messages, which contain the highest sequence numbers, can be used to help receivers detect losses quickly. In this case, a receiver may also detect losses if no data have arrived after the maximum sending interval has past[2]. We consider a scenario where there is one sender and multiple receivers in the multicast group.

In ARM, a receiver sends a NACK towards the sender as soon as it detects a loss. Multiple NACKs from different receivers are cached and "fused" at active routers along the multicast tree. If all routers are active and do not flush NACKs from their caches prematurely, the sender will receive at most a single NACK per loss; otherwise, the sender may receive more than one NACK. The sender responds to the first NACK by multicasting a repair to the group. It then ignores subsequent NACKs for this packet for a fixed amount of time (e.g., for the estimated RTT to the farthest receiver in the group).

Since NACKs and repair data may also be lost, a receiver must resend a NACK if it does not receive the repair within a certain time limit, which we assume to be at least 1 RTT between the receiver itself and the original source of the data packet. To identify new NACKs, each NACK contains a NACK count to indicate how many times the receiver has requested a lost data packet. The sender maintains the highest NACK count associated with each requested repair. If it receives a NACK with a higher NACK count, it assumes that the previous retransmission was lost and multicasts the repair to the group again.

We do not make any further assumptions about end-point behavior, such as sending rates and timeout schemes, which are specific to different reliable multicast protocols. The assumptions about end-point behavior listed above are the only ones required to support ARM.

Intermediate routers perform the following actions:

- **Data caching for local retransmission.** Routers at strategic locations perform best-effort caching of multicast data for possible retransmission. When a router receives a NACK, indicating that a receiver has detected a packet loss, it retransmits the requested packet if that packet is in its cache; otherwise, it considers forwarding the NACK towards the sender.
- **NACK fusion/suppression.** Routers control implosion by dropping duplicate NACKs and forwarding only one NACK upstream towards the source per multicast subtree.
- **Partial multicasting for scoped retransmission.** Routers perform partial multicasting of retransmitted packets so that they are delivered only to receivers that previously requested them.

These actions are triggered by the receipt of one of three kinds of ARM packets: multicast data packets, NACK packets, and retransmitted packets. The following subsections describe how each of these packets are handled.

A. *Caching Data at Routers*

ARM caches multicast data packets at routers in the multicast tree for possible retransmission. When a NACK travels upstream towards the sender, any router along the path can retransmit the requested packet if it has that packet in its cache. Since end-to-end latency over a wide-area network can be long, retransmissions from intermediate router caches can significantly reduce the recovery latency for distant receivers. Additionally, by distributing the load for retransmission to routers along the multicast tree, ARM protects the sender and bottleneck links from retransmission requests and repair traffic.

An ARM multicast data packet has the following fields in its header: multicast group address, source address, sequence number, cache TTL t, and NACK count. The cache TTL field specifies the useful lifetime of a data packet in an active router's cache. The NACK count is significant only for repair packets, which have the same header fields as multicast data packet; it indicates the number of times a single receiver has requested the repair. An active router processes a multicast data packet, as follows; an inactive router simply forwards the packet along the multicast tree.

```
Algorithm for data packet DP:
If (cache is available at this node) {
  Store DP in cache;
  Set DP's cache TTL to DP.t;
}
For each (outgoing link) {
  If (downstream receivers are subscribed
      to DP.group) {
     Forward DP down link;
  }
}
```

The amount of time a data packet is cached at a router depends on how soon the router expects to receive a NACK for that packet from the "farthest" receiver downstream. As discussed above, receivers detect losses from sequence gaps in the packets or from the expected packet sending rate. In either case, the amount of time it takes a receiver to detect a loss is a function of the expected inter-packet (data or session message) sending rate and the latency between the receiver and the source. Therefore, the amount of time a fresh data packet should be cached at a router can be approximated as a function of the inter-packet sending rate and the max RTT (round trip latency) between the sender and the "farthest" receiver downstream. If a protocol recycles its data packet sequence numbers, the caching time of a data packet should also be bounded by the amount of time it takes the source to send one "cycle" worth of data.

To further reduce storage and processing costs, we recommend caching packets only at routers at "strategic" locations. Research is currently in progress to determine the optimal locations at which to cache data. For now, we describe several factors that influence the placement of these caches.

Caching is particularly valuable when some receivers are connected to the rest of the group over lossy links, such as wireless links. Caching data packets immediately before lossy links can significantly reduce recovery latency. A similar approach in router-based cached retransmission has been adopted by the snoopTCP protocol [9] to improve the performance of TCP over wireless networks.

Caching presents a tradeoff between network-based storage and bandwidth. Studies on packet loss patterns in the current MBone [10] indicate that most packet losses occur at the "edges" of the network, instead of in the bandwidth-rich backbone. By locating caches where bandwidth is scarce (for example, where the backbone meets slow-speed access links), ARM can shield other parts of the network from frequent retransmission request and repair traffic.

Existing end-to-end approaches to reliable multicast, such as RMTP, TMTP and LBRM, offload the sender site by having designated proxies perform retransmissions. However, these approaches do not protect the proxies' bottleneck links from being overloaded. If a proxy is responsible for retransmissions to receivers that are behind lossy links, frequent repair and request traffic may eventually cause the proxy's access links to become congested also.

B. NACK Suppression and Local Retransmission

ARM processes NACK packets so as to prevent unnecessary request traffic from propagating beyond an active node. NACKs also provide subscription information for repairs, enabling routers to determine which outgoing links contain group members interested in receiving a repair. Finally, NACKs can trigger local retransmission from an active node that has the requested data in its cache.

ARM caches information in routers' soft state to support NACK suppression and scoped, local retransmission. In a traditional IP multicast network, a router maintains the group, source address, and membership information for each ongoing multicast session it supports [8]. An active ARM router also caches, for a short amount of time, the following information for each loss that it is handling: a NACK record, a REPAIR record, and possibly the data packet itself. All three items are uniquely identified by the group address, source address, and sequence number of the lost data packet.

A NACK record for a data packet contains the highest NACK count received for that packet and a *subscription bitmap* indicating the outgoing links on which NACKs for the packet have arrived. Routers use the NACK record to suppress subsequent duplicate NACKs, and the subscription bitmap to determine the outgoing links on which it should forward the subsequent repair packet.

A REPAIR record for data packet p contains a vector indicating the outgoing links on which the repair for p has already been forwarded during the time the REPAIR record was cached. Routers use the REPAIR record mainly to suppress NACKs sent by receivers before they receive a repair that is in transit. For each link, the vector indicates the highest NACK count contained in a repair for p already sent down that link.

An ARM NACK packet has the following fields in its header: address of receiver originating the NACK, address of original source of the lost data, multicast group address, sequence number of requested data packet, NACK count, and suggested cache TTL t for NACK and REPAIR records.

The processing for a NACK packet not only suppresses duplicate NACKs, but also makes the necessary preparation for scoped retransmissions. It first checks to see if the requested repair has just been forwarded down the link on which the NACK arrived. If so, the router drops the NACK. If not, and the requested repair is in the router's cache, the router retransmits the repair. Otherwise the router subscribes the originator of the NACK to a subsequent transmission of a repair, and it forwards only the first NACK for this repair to the sender. A NACK packet with a higher NACK count, sent when a receiver does not receive an expected repair, overrides the suppression set by previous NACKs or repairs, which may have been lost. In more detail, an active router processes a NACK packet as follows; an inactive router simply forwards the packet towards the sender.

```
Algorithm for NACK packet NP arriving on
link k:
Look up unexpired NACK record NR, REPAIR
record RR, and data or repair packet DP
for (NP.group, NP.source, NP.seqNumber);

If (RR found && RR's NACK count for link k
               >= NP.nackCount) {
  // Do nothing
} Else If (DP found) {
  Set DP's packet type to REPAIR;
  Set DP's NACK count to NP.nackCount;
  Deliver DP down link k;
  If (RR not found) {
     Create REPAIR record RR for
      (DP.group, DP.source, DP.seqNumber);
  }
  Set RR's cache TTL to NP.t;
  Set RR's NACK count for link k to
      NP.nackCount;
} Else If (NR found &&
           NR contains subscribed link &&
           NR.nackCount >=
                NP.nackCount) {
  Subscribe link k to repair packet;
} Else {
  If (NR not found) {
     Create NACK record NR for
      (NP.group, NP.source, NP.seqNumber);
  }
  Set NR.nackCount to NP.nackCount;
  Set NR's cache TTL to NP.t;
  Subscribe link k to repair packet;
  Forward NP toward the source;
}
```

In general, the amount of time a NACK record should be cached at a router depends on how soon the router expects to receive the corresponding repair. A repair can come either from the original source or from routers along the path. Thus, the NACK record should be cached for at least one RTT from the router to the original source. The amount of time a router should cache a REPAIR record should be approximately one RTT from the router to the farthest receiver downstream. Since we do not expect a router to always have the estimated RTT value from itself to end-points, a reasonable caching time for NACK and REPAIR records can be approximated as a function of the RTT between the sender and the "farthest" receiver in the group.

C. Scoped Retransmissions

ARM processes repair packets in much the same way as multicast data packets, which have the same header fields. An important difference is that active routers scope retransmission of repair packets to the portions of the multicast group experiencing loss. They do this by looking up the corresponding subscription bitmap, which was created and left in the router's soft-state cache by previous NACKs. If the relevant subscription information is found in the cache, a router forwards a repair only to subscribed links. If no subscription information is available (e.g., because a cache was flushed or a route was changed), the router merely caches the repair. If no cache is available, the router forwards the repair down all links.

```
Algorithm for repair packet RP:
Look up unexpired NACK record NR and
REPAIR record RR for
(RP.group, RP.source, RP.seqNumber);

If (cache available at this node) {
  Store RP in cache;
  Set RP's cache TTL to RP.t;
  If (NR found) {
      Forward RP down each subscribed
        link in NR;
      Remove NR;
  }
} Else {
  For each (outgoing link) {
  If (downstream receivers are subscribed
      to RP.group) {
     Forward RP down link;
  }
}

If (RR not found) {
   Create REPAIR record RR for
     (RP.group, RP.source, RP.seqNumber);
   Set RR's cache TTL to RP.t;
}
For (each link i on which RP
     was forwarded) {
   Set NACK count for i in RR to
       RP.nackCount;
}
```

Active routers cache repair packets in order to accommodate the wide range of possible NACK arrival times from geographically dispersed multicast group members. If network-based storage is limited, it is more important to cache repair packets than the original multicast data packets, because a loss is likely to be experienced by several receivers, whereas many original packets may not be lost at all.

ARM degrades gracefully even when network-based storage is not available for caching repairs, but is still available for caching NACK and REPAIR records. Assuming that the network does not lose the repair itself, and that NACK and REPAIR records remain cached for the max RTT, ARM in the worst case multicasts a repair to the entire group, even if multiple receivers send NACKs for that repair.

IV. Implementation

We have implemented an ARM prototype in Java and have run it on SPARCs under Solaris 2.5. We built the prototype using the Active Node Transport System (ANTS) [11], which provides a set of Java classes on which to base active network applications. Among other classes, the `Capsule` class attaches code fragments to data packets. ANTS has a flexible code distribution scheme in which capsule code fragments can be "demand-loaded" into an active node on a dynamic basis. Thus code for ARM need not reside permanently in all active nodes, but can be downloaded as needed. In our prototype, code for ARM is loaded into an active node during multicast session initialization; each capsule transmitted after the initialization process contains only the header but not the code fragment.

Different ARM processing is invoked when different ARM capsule types (i.e., multicast data capsules, NACK capsules, and repair capsules) arrive at an active node. Each ANTS node also provides a LRU-based soft-state storage mechanism which allows capsule code to cache transient data items. The cache periodically flushes items with expired lifetimes.

We emulated a multicast environment by creating multiple ANTS nodes on different workstations in a 100 Mbit Ethernet LAN. The ANTS nodes communicate with one another through UDP. Our prototype uses a shortest-path based multicast forwarding service.

Since we implemented our prototype in a purely interpretive user-space environment, processing a basic multicast data capsule takes approximately 4 ms on a 167-MHz UltraSparcs with 128 MB of memory. We expect that a more efficient runtime environment will reduce the time required to process ARM packets by at least a factor of ten, by using compiled rather than interpretive code, by eliminating user-space/kernel context switches, and by reducing the number of times packets are copied.

More importantly, our measurements show that NACK suppression and scoped retransmission require at most a 25% increase in processing time over basic multicast routing. Considering that wide-area latency across the U.S. is 80 - 100 ms, and assuming that an efficient implementation of ARM adds an overhead of 200 μs to basic routing, ARM in-network processing imposes negligible increases in end-to-end wide-area latency. Additionally, using our approach, a router only performs active processing, such as NACK suppression and scoped retransmissions, when a group member downstream experienced a loss. Therefore, in most cases, only a small fraction of the overall traffic requires active processing. Caching of fresh multicast data packets may involve a larger fraction of traffic; however, it is not as important to cache fresh data as it is to cache repairs.

V. Simulations

We simulated ARM in order to evaluate its performance (with varying numbers of active routers in the network), to measure the tradeoffs it provides between router storage/processing and network bandwidth/latency, and to compare its with that of SRM. The results show that ARM performs well with respect to the following metrics.

- **Recovery Latency.** The end-to-end recovery latency is lower for ARM than for SRM, even if ARM does not cache multicast data packets. Most of ARM's improvement in recovery latency can be obtained when fewer than 20% of the routers cache fresh multicast data packets.
- **Implosion Control.** The maximum number of NACKs that a single node (either a router or an end-point) must handle during recovery from a single loss is approximately the same for ARM and SRM. ARM is able to control implosions when fewer than 50% of the routers are active.
- **Bandwidth Consumption.** In comparison with SRM, NACK packets consume less bandwidth in ARM, which performs well in this respect when fewer than 50% of the routers are active. Repair packets also consume less bandwidth in ARM, which provides a local recovery mechanism where SRM does not. Here again, significant benefits are obtained from scoped retransmission and cached repairs when only 40% of the routers are active.

To compare ARM directly with SRM, we used the same assumptions about the network, as well as the same simulator, as did LBNL in their analysis of SRM [1], [12]. The LBNL simulations examine recovery behavior on a per-loss basis; they do not consider scenarios in which NACKs and repairs are lost in addition to fresh data packets. In our experiments, we used the LBNL simulator to construct a bounded degree tree of N nodes as the underlying network topology ($N = 1000$ and the degree is 4). The simulator then randomly chooses G of the N nodes as multicast group members (G varies from 10 to 100 in our experiments), randomly picks a source S from among the G members, and randomly picks a link as the packet drop point. The simulator assumes that it takes one time unit to traverse each network link.

For some simulations, we vary the number of nodes in the multicast tree that are capable of active processing. To generate a tree in which $1/A$ of the nodes are active, we first allow the simulator to construct the shortest-path multicast tree rooted at the source, and we then randomly pick $1/A$ of the non-leaf nodes in this tree to be active.

A. Recovery Latency

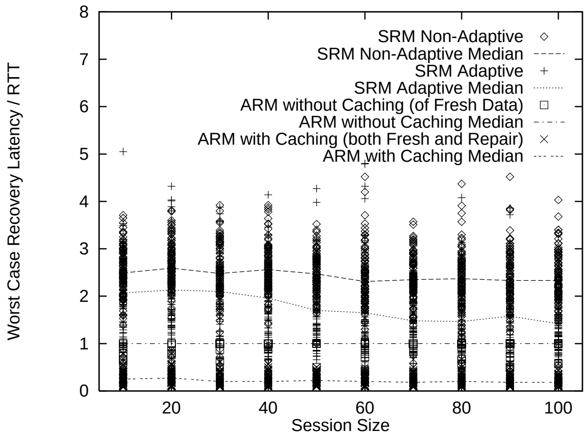

Fig. 1. ARM vs. SRM worst case recovery delay (random loss, 1000 nodes, degree 4). ARM results shown when router caches of fresh multicast data are enabled and disabled. SRM results shown for both non-adaptive and adaptive algorithms.

Figure 1 compares the loss recovery delay of ARM with that of SRM. As in [1], [12], the loss recovery delay is the time from when a receiver first detects a packet loss to when it receives the first repair for that loss. The data points in Figure 1 show the worst case recovery delay for all receivers in the multicast group, measured in multiples of RTT, the round trip time between the sender and the receiver experiencing the loss. For each group size ranging from 10 to 100, the graph shows the results of 100 experiments. Each experiment involves four simulations, two for SRM and two for ARM, based on the same freshly generated network and group topology.

The SRM simulations show the recovery delay for the non-adaptive and adaptive variants of SRM. Both variants impose backoff delays on receivers before they issue NACKs or repair packets. The adaptive algorithm adjusts the delays based on past performance. Our simulations use the same backoff delays as used to produce Figure 6 in [12]: the NACK backoff is chosen randomly between one and two RTT, the repair backoff between one and two one-way delay times $\log_{10} G$. The backoff delays for the adaptive algorithm are from the 40th loss recovery round with a single repeated failure mode for each round.

Because the non-adaptive SRM backoff timers cause receivers to wait before issuing a NACK, and responders to wait before sending a repair, the non-adaptive SRM recovery latency can be long. Figure 1 shows that the median non-adaptive worst-case recovery latency is approximately 2.5 RTT. The adaptive algorithm improves the recovery latency, but it is still more than 1.4 RTT.

In ARM, receivers send NACKs towards the sender immediately upon detecting a loss. Hence the recovery latency is one RTT when no intermediate routers cache multicast data. When all intermediate routers cache multicast data, the latency is reduced to approximately 0.2 RTT. This is because NACKs trigger a "local" retransmission at the first router upstream from the failure link.

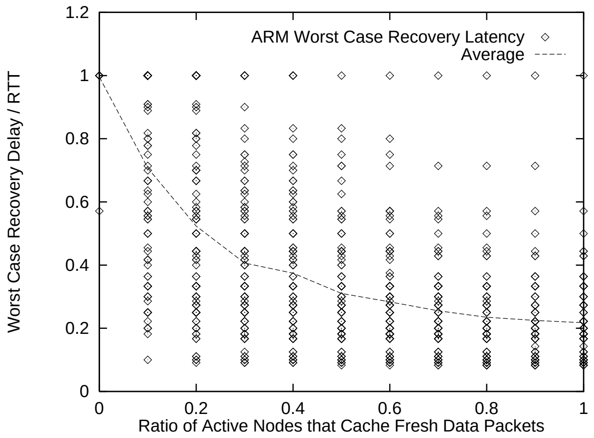

Fig. 2. ARM tradeoff between caching of fresh multicast data and latency (random loss, group size 100, 1000 nodes, degree 4). All non-leaf nodes in multicast tree are active; caching of repair packets is enabled at all nodes.

Figure 2 presents less obvious data about the incremental benefit of caching of multicast data on the end-to-end recovery latency. The data in that graph were generated from simulations in which all non-leaf routers were active and performed both NACK suppression and scoped retransmission. However, the number of (randomly selected) routers caching fresh data packets varied from 0% to 100%, with 100 simulations being performed for each percentage. The data show that the average recovery latency was cut nearly in half when only 20% of the randomly picked routers cached multicast data. This result suggests that ARM can significantly improve recovery latency by caching multicast data at a relatively small number of strategically placed active routers.

B. Implosion Control

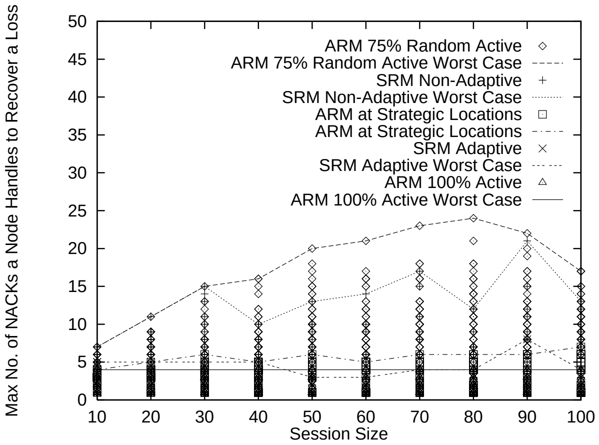

Fig. 3. NACK implosion control (loss near source, 1000 nodes, degree 4). ARM caches repairs, but not fresh data packets.

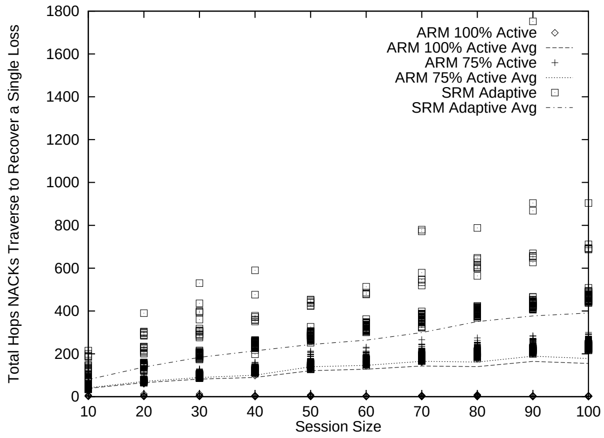

Fig. 4. NACK bandwidth consumption (loss near source, 1000 nodes, degree 4). ARM caches repairs, but not fresh data packets. SRM uses adaptive algorithm.

Figures 3 and 4 measure the effectiveness of ARM and SRM in controlling NACK implosion. Figure 3 shows the maximum number of NACKs that either a router or an end-point handles during the recovery from a single loss. Figure 4 shows the total number of hops NACKs traverse during this recovery. Both figures also show the performance of ARM varies as the number of routers performing NACK suppression decreases. The figures show what happens when a packet is lost near the source, since the implosion effect is most serious when many receivers are affected by the loss.

Since the topology used in the simulations is a bounded degree tree, ARM causes nodes to receive at most four (the degree of the tree) NACKs when every router performs NACK fusion, and to send at most one NACK towards the sender. The SRM adaptive algorithm exhibits approximately the same worst case NACK load, because different receivers can generate duplicate NACKs. However, SRM multicasts the duplicate NACKs to the entire group, which consumes extra bandwidth as shown in Figure 4. Local recovery mechanisms proposed by SRM might reduce total bandwidth consumption by using TTLs to limit the scope of NACKs and repairs. However, as pointed out in [12], when the loss impact area is large, multiple local recovery events may consume more bandwidth than a single global recovery.

The figures show that ARM performs reasonable implosion control even when only 75% of the (randomly picked) routers can perform NACK suppression. More interestingly, when routers at "strategic" locations perform NACK suppression, ARM can achieve most of its benefits with fewer than 50% of its nodes active. A simple criterion defines a router location to be strategic if there are more than two outgoing links to multicast group members. Figure 5 shows the ratio of the number of strategically placed routers to the total number of non-leaf nodes in the multicast tree. Note that for group sizes up to 100, fewer than 50% of the locations in the multicast trees in our simulations are strategic.

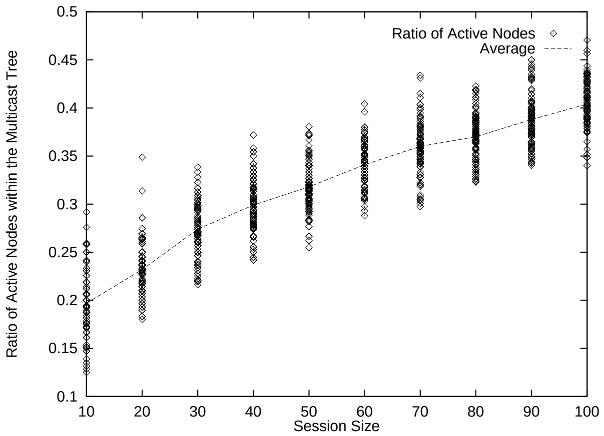

Fig. 5. Ratio of "strategic" nodes to number of non-leaf nodes in multicast tree (1000 nodes, degree 4).

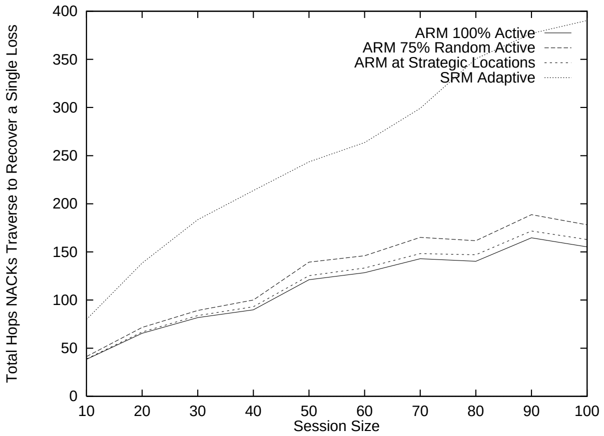

Fig. 6. NACK bandwidth consumption with strategically placed active nodes (loss near source, 1000 nodes, degree 4). ARM caches repairs, but not fresh data packets. SRM uses adaptive algorithm.

Figure 6 compares the performance of ARM when only strategic nodes are active to the performance of SRM and of ARM when the active nodes are chosen randomly. It confirms that strategic placement of active nodes results in performance close to that achieved when all nodes are active, and in any case better than that achieved when a larger number of nodes are chosen randomly to be active.

C. Bandwidth Consumption

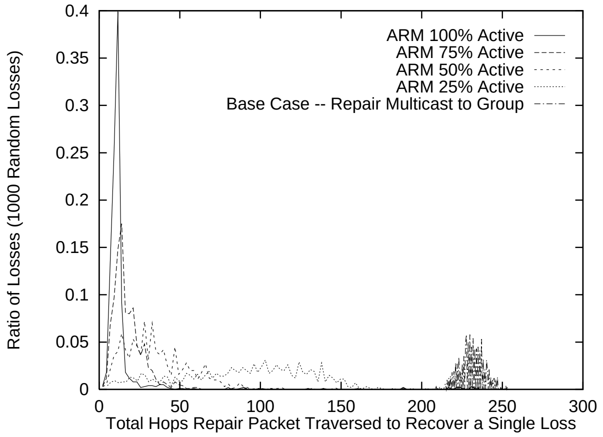

Fig. 7. Effectiveness of ARM scoped retransmissions (random loss, 1000 nodes, group size 100, degree 4). Active nodes scope retransmission and cache repair packets, but do not cache fresh data packets.

Figure 7 shows the bandwidth consumed by repair packets during recovery from a single loss, as measured by the total number of hops traversed by those packets. The baseline case occurs when no routers are active in ARM. In this case, a repair is multicast to the entire group, and Figure 7 shows the normal distribution of the total number of hops in the randomly generated multicast trees. The other cases show how ARM behaves when varying numbers of routers perform scoped retransmission, but none cache fresh multicast data. Although caching fresh data would reduce repair bandwidth, the data in Figure 7 were obtained without it so as to better assess the impact of scoped retransmission.

Figure 7 shows that, when all routers perform scoped retransmission, 40% of all losses require the repair packet to traverse at most 10 hops, and 80% require at most 12 hops. Even when only 25% of the routers are active, 40% of all losses require the repair packet to traverse at most 1/3 the hops in the multicast tree, and 80% require at most 1/2 the hops. The group size in Figure 7 was fixed at 100.

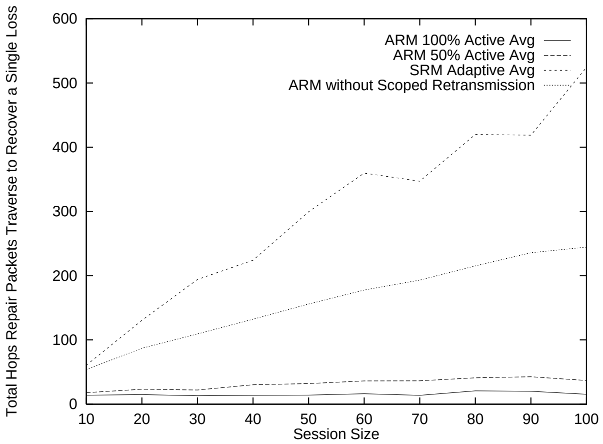

Fig. 8. Hops traversed by repair packets to recover a single loss (random loss, 1000 nodes, degree 4). ARM caches repairs, but not fresh data packets. SRM uses adaptive algorithm, 40th round.

Figure 8 shows the effectiveness of scoped retransmission in

ARM for different group sizes and different numbers of active nodes in the multicast tree. Here again, the base case shows the effect of multicasting the repair to the entire group. The data show that ARM is effective in limiting repair traffic even when only 50% of the (randomly picked) active routers scope retransmission. It also shows that SRM's recovery bandwidth increases dramatically as the group size increases, because SRM multicasts repairs to the entire group.

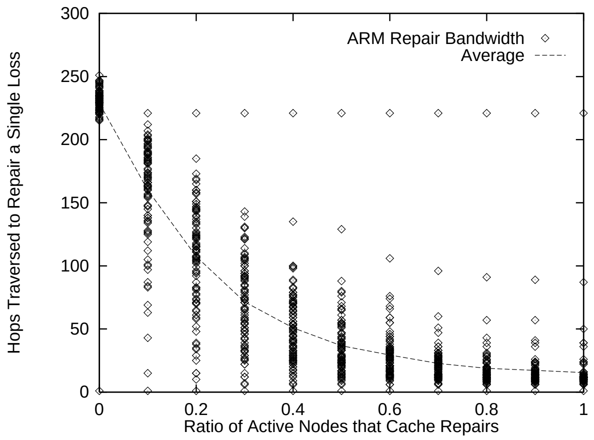

Fig. 9. ARM tradeoff for caching repairs and hops traversed by repair packets (random loss, 1000 nodes, group size 100, degree 4). All nodes active. No nodes cache fresh data packets. Nodes that cache repair packets are picked randomly. Source always caches repairs.

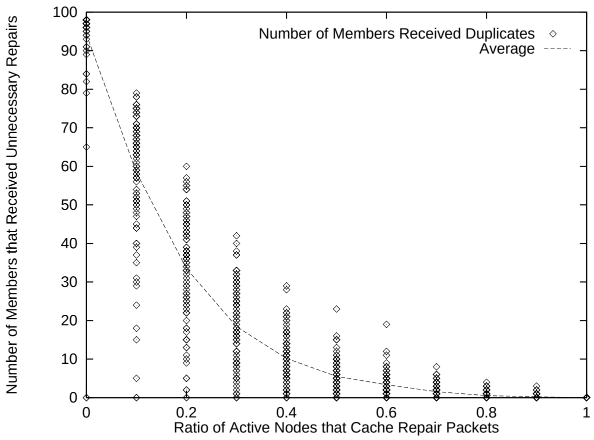

Fig. 10. Effect of caching repairs in ARM and number of receivers that receive unnecessary repairs (random loss, 1000 nodes, group size 100, degree 4). All nodes active. No nodes cache fresh data packets. Nodes that cache repair packets are picked randomly. Source always caches repairs.

Figures 9 and 10 show the effect of caching repair packets on recovery bandwidth and the number of receivers that receive unnecessary repairs. All non-leaf nodes in the multicast tree are assumed to be active, but the number that have cache capacity varies. In all cases, the source is always able to cache the repairs and perform scoping of retransmissions. As Figure 9 shows, most of the benefit of scoped retransmission on bandwidth savings can be obtained when only (a randomly selected) 40% of the routers cache repairs. As Figure 10 shows, when all nodes within the multicast tree have the cache capacity for repairs, ARM achieves "perfect" scoping of repair packets – i.e., none of the 100 session members received unnecessary repairs. When 40% of the (randomly selected) nodes can cache, ARM, on an average, shields approximately 90% of the affected members from unnecessary repairs.

VI. Related Work

Existing work on wide-area reliable multicast loss recovery falls mainly into two categories. The first category, represented by Scalable Reliable Multicast (SRM) [1], [12], uses randomized backoff to avoid the NACK implosion problem. In SRM, NACKs are multicast to the group, possibly with a TTL to limit their delivery scope. Any member that has reliably received the requested data can respond to the retransmission request. The second category organizes receivers into hierarchies. Particular hierarchical approaches include RMTP [4], TMTP [3], and LBRM[2]. In these approaches, a subset of receivers (or some other representatives such as loggers) is designated to provide proxies for the sender in responding to retransmission requests. Receivers are responsible for identifying their designated "representatives" and for sending feedback messages directly to them. A representative will either unicast or multicast the repair to its subgroup if it has the repair.

Neither category of approaches offers a satisfactory solution. SRM is the more fault-tolerant and flexible of the approaches, since any member of the group can perform retransmissions. However, its timer-based implosion control mechanism incurs an additional cost in recovery latency, and it suffers from duplicate requests and repairs due to its probabilistic nature. Furthermore, local recovery in SRM is still an open issue. Hierarchical approaches do a better job in localizing the error recovery process. However, it is difficult to maintain good hierarchies, especially in the face of frequent membership changes.

There are several recent proposals (e.g., [13] and [14]) to solve these problems by extending the multicast routing services to support reliable multicast loss recovery. The general idea is to have routers deliver NACKs upstream toward the original source until they reach a "turning point," where they are forwarded as multicast packets downstream toward other group members that might have the requested data packet. One problem is that it is difficult to decide where optimal "turning points" are located in the network on a per loss basis. Another is that receivers can still receive duplicate NACK packets.

In comparison with SRM, ARM has a much lower recovery latency and provides a specific solution (router-based scoped retransmission) to local recovery. ARM is more flexible and robust than the hierarchical approaches, because any router with cached multicast data can perform retransmissions, and end hosts do not have to maintain nor have any knowledge of group topology. ARM does not require all nodes to be active, and does not rely on any particular router or receiver to perform loss recovery. Our simulation results show that ARM yields good scaling properties even when less than 50% of the nodes are active.

VII. Conclusions

In this paper, we have shown that network based processing and storage can be used to enhance the performance and scalability of reliable multicast. Our algorithm, ARM, utilizes intermediate routers to reduce both NACKs and repair traffic.

Routers also help distribute the retransmission load by caching multicast data. ARM is flexible and robust in that it does not require all routers to be active, nor does it require any specific router or receiver to perform loss recovery. Additionally, it is robust with respect to dynamic changes in group membership.

Analysis and simulation results show that ARM performs well in terms of recovery latency, implosion control, and repair bandwidth. Whereas SRM trades recovery latency off against repair bandwidth, ARM utilizes network based processing and storage to reduce both. ARM's performance degrades gracefully as the network resources used for active processing decrease. Significant benefits are still obtained when only half the routers within a multicast tree perform ARM processing. Our simulation results suggest that the same benefits can be obtained from a much smaller set of participating routers placed at strategic locations. Research is currently in progress to determine the optimal locations for those routers.

Acknowledgements

The authors would like to thank Sally Floyd for providing the SRM simulator code. We also thank David Wetherall for providing the ANTS toolkit. The work benefited from discussions with Liuba Shrira, David Wetherall, Ulana Legedza, and John Guttag.

References

[1] Sally Floyd, Van Jacobson, Steven McCanne, Ching-Gung Liu, and Lixia Zhang, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing," in *ACM SIGCOMM'95*, October 1995, pp. 342 – 356.

[2] Hugh W. Holbrook, Sandeep K. Singhal, and David R. Cheriton, "Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation," *ACM SIGCOMM'95*, pp. 328 – 341, October 1995.

[3] R. Yavatkar, J. Griffioen, and M. Sudan, "A Reliable Dissemination Protocol for Interactive Collaborative Applications," in *ACM Multimedia*, 1995.

[4] John Lin Sanjoy Paul, Krishan Sabnani and Supratik Bhattacharyya, "Reliable Multicast Transport Protocol (RMTP)," in *IEEE Journal on Selected Areas in Communications*, April 1997, vol. 15.

[5] David Tennenhouse and David Wetherall, "Towards an Active Network Architecture," in *SPIE Proceedings of Conference on Multimedia Computing and Networking 1996*, San Jose, CA, January 1996.

[6] David D. Clark, "The Design Philosophy of the DARPA Internet Protocols," in *ACM SIGCOMM'88*, Stanford, CA, August 1988.

[7] Stephen E. Deering, "Multicast Routing in Internetworks and Extended LANs," in *Proceedings of ACM SIGCOMM'88*, Stanford, CA, August 1988.

[8] Stephen Deering and David Cheriton, "Multicast Routing in Datagram Internetworks and Extended LANs," *ACM Transactions on Computer Systems*, vol. 8, no. 2, pp. 85–110, May 1990.

[9] H. Balakrishnan et al., "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links," in *ACM SIGCOMM'96*, Stanford, CA, August 1996.

[10] M. Yajnik, J. Kurose, and D. Towsley, "Packet Loss Correlation in the MBone Multicast Network," in *IEEE Global Internet mini-conference, GLOBECOM'96*, 1996.

[11] D. Wetherall, J. Guttag, and D.L. Tennenhouse, "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols," in *IEEE OPENARCH'98*, San Francisco, CA, April 1998.

[12] Floyd et al., "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing, Extended Report," LBNL Technical Report, September 1995, ftp://ftp.ee.lbl.gov/papers/wb.tech.ps.Z.

[13] Eshwar Belani and Steve McCanne, "Subcast Based Local Recovery Schemes for Reliable Multicast," Slides from MASH Retreat, June, 1997, http://www-mash.cs.berkeley.edu/dist/mash/sum97-retreat/rm.ps.

[14] Christos Papadopoulos, Guru Parulkar, and George Varghese, "An Error Control Scheme for Large-Scale Multicast Applications," in *IEEE INFOCOM'98*, San Francisco, CA, April 1998.