An Efficient and Scalable Loss Recovery Scheme For Video Multicast

Jun Peng and Biplab Sikdar
Electrical, Computer and Systems Engineering Department
Rensselaer Polytechnic Institute (RPI), Troy, NY 12180
{pengj2, sikdab}@rpi.edu

***Abstract*— With the increased popularity of multimedia services on the Internet, efficient video multicast strategies that can scale easily are of critical importance. This paper addresses the issue of video multicast loss recovery and presents an efficient and scalable scheme: Active Injection Recovery (AIR). The proposed scheme has three distinguishing features: active injection of repair packets into loss regions, on-demand construction of loss recovery structures, and unique rate control over repair traffic. All these features can save considerable network resources in a large-scale video multicast session. In addition, the proposed scheme simultaneously meets the three well-known requirements for efficiency and scalability in multicast loss recovery: request suppression, local recovery, and retransmission scoping. Another important feature of the proposed scheme is its low recovery latency, which is essential for video multicast. Our analysis shows that the proposed scheme achieves significantly better overall performance as compared to existing multicast loss recovery schemes. Simulation results show that the proposed scheme can improve the video quality considerably in a streaming video multicast session.**

Keywords: Algorithm/protocol design and analysis

I. Introduction

Multimedia applications will be among the most popular users of Internet multicast services because of their inherent one-to-many transmission requirement. Efficiency and scalability are essential for multimedia multicast because a multimedia multicast session can have a very large scale in terms of both geographical spread as well as the number of nodes involved (e.g., the multicast of a football game on the Internet may spread across several continents and have tens of thousands of receivers). Therefore, efficiency and scalability are also critical for a loss recovery scheme designed for large-scale video multicast. Research on multicast loss recovery focuses on two directions: end-to-end schemes and network-assisted schemes. The schemes in the end-to-end direction such as [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [1], in general, are not very efficient or scalable because of the lack of information about the underlying network. Therefore, efforts have also been made to explore schemes in the network-assisted direction, such as [14] [15] [16] [17] [18] [19] [20] [21] [22]. Although these network-assisted schemes can achieve better overall performance than most end-to-end schemes, their overhead limits their efficiency and scalability (e.g., some of them require data caching at router sites). Therefore, new techniques for enhancing the efficiency and scalability of multicast loss recovery are still needed for supporting large-scale video multicast.

[1]A survey on multicast loss recovery schemes can be found in [13].

It has been shown in [23] that distributed multicast loss recovery schemes usually outperform source-based multicast loss recovery schemes in efficiency and recovery latency, both of which are essential for large-scale video multicast. However, existing distributed schemes, end-to-end or network-assisted, usually need to prepare a loss recovery structure before or at the beginning of a multicast session (e.g., to distribute distance information among receivers or to set up states in routers across the multicast tree). Even if there is not a single loss during a multicast session, these schemes still set up the structure across the multicast tree. This is not efficient, since setting up a loss-recovery structure requires spreading control messages and sometimes activating states in related routers. Another factor degrading the efficiency of existing distributed schemes is that they do not specifically control the rate of the repair traffic generated by them across a multicast tree. The uncontrolled repair traffic can exacerbate the congestion on a bottleneck that it traverses. Consequently, network efficiency is reduced. In addition, existing schemes either still have low performance in request suppression, local recovery, or retransmission scoping, which are the three essential elements for efficiency and scalability in multicast loss recovery, or they need data caching at router sites, which also limits their scalability.

The proposed scheme, Active Injection Recovery (AIR), overcomes all the above disadvantages of existing multicast loss recovery schemes that limit their efficiency and scalability. Unlike existing distributed schemes, the AIR scheme does not prepare a loss recovery structure across a multicast tree in advance. Instead, AIR only proceeds to build a minimum loss recovery structure when a loss event does occur and recovery is indeed necessary. When losses occur on a branch, the AIR agent residing in the router just above the branch dynamically defines a loss region and proceeds to find a pair of receivers (the request source and the repair source) near the loss site to supply loss information and repair packets, respectively. The AIR agent then subcasts the repair packets to the loss region. After the loss event ends, the AIR agent deletes related states, and the request source and the repair source release their roles. Thus, the temporary loss recovery structure is torn down when there are no further losses. In addition, during the loss recovery process, the request source of a loss region controls the submission of repair requests to alleviate the interference of the repair traffic on the bottleneck. Our simulations show that not only is the number of losses on a bottleneck significantly reduced with our repair traffic control mechanism, but also the average recovery latency is not increased. This is a great advantage for large-scale video multicast.

Using this new active and dynamic approach and by involving only local nodes and routers in the recovery process, AIR can save significant network resources during a large-scale video multicast session. Furthermore, AIR simultaneously meets the three well-known requirements for efficiency and scalability in multicast loss recovery. With AIR, retransmission requests are produced only by the request source instead of all receivers in each loss region. Meanwhile, these requests are transmitted to the repair source by unicast instead of multicast or hop-by-hop extra-processing. Therefore, AIR suppresses requests not only in production but also in spreading. Because repair packets are subcast at the branch where losses occur, the leakage of repair packets to unrelated receivers is impossible with AIR. All these measures allow the AIR scheme to achieve better request suppression and retransmission scoping than other schemes that have been proposed (e.g., SRM [7] and LMS [17]). Also, AIR does not require data caching at router sites to achieve ideal local recovery. Our analysis shows that the AIR scheme has a significantly better overall performance in efficiency and scalability as compared to existing schemes. Furthermore, AIR has very low recovery latency. Our simulations show that AIR can significantly increase the quality of the transmitted video in a streaming video multicast session.

The rest of the paper is organized as follows. The next section introduces the related work to AIR. Section III presents the AIR scheme. Section IV analyzes the interference of repair traffic on bottlenecks in multicast and the mechanism used by AIR to reduce the interference. Analysis of the proposed scheme appears in Section V. Section VI presents simulation results. Finally, we give our conclusion in Section VII.

II. Related Work

In existing network-assisted multicast loss recovery schemes such as [14] [15] [16] [17] [18] [19] [20] [21] [22], LMS [17] is probably the most closely related work to the proposed scheme. LMS achieves good performance in request suppression, local recovery, and retransmission scoping by maintaining a replier link in each router of the multicast tree for each multicast session. Although with the same goal of achieving high efficiency and scalability in multicast loss recovery, LMS and the proposed scheme have a fundamental difference: how the recovery structures in a multicast tree are constructed and maintained. With the proposed scheme, the recovery structure for a loss event is constructed and maintained actively and on-demand; with LMS, the recovery structure is set up passively and maintained permanently in a multicast session. In addition, the proposed scheme has a mechanism for rate control over its repair traffic, which LMS does not have. Because of the difference of the approach in constructing and maintaining the loss recovery structures in a multicast tree, the proposed scheme, in general, introduces less overhead for recovering the lost packets in a multicast session than LMS, as analyzed in Section V. However, with its permanently maintained replier links across a multicast tree, LMS does not need to temporarily set up the replier links for a loss event before the recovery process begins. So it have advantages in loss recovery latency in some situations where the request or repair source can not be found locally, which is discussed in the scheme analysis part, Section V. In the end-to-end direction, SRM [7] is possibly the most related scheme to AIR in terms of the roles that receivers play in loss recovery. SRM does not set up special agents/servers in a multicast tree as repair sources. Instead, it solely depends on receivers and the multicast source in providing repair packets in a multicast tree. This characteristic also applies to LMS and AIR. Basically, SRM spreads the loss information of a loss event across the multicast tree and each receiver capable of repairing the losses is a potential repair source. Meanwhile, SRM uses distance-based random timers to suppress duplicate repair requests and repair packets. Although SRM is robust, it usually introduces significant overhead in loss recovery. In addition, SRM has relatively long recovery latency because of the use of suppression timers. Section V analyzes the three related schemes and gives details about their performance. Before that, we first present the AIR scheme in the next section.

III. The AIR Scheme

A. Scheme Overview

To illustrate how the proposed scheme works, this subsection considers the simple scenario of a multicast tree with a single bottleneck. The next subsection will present the details of the complete AIR scheme and how the scheme works in multiple-bottleneck scenarios.

Part of the topology of a multicast tree is shown in Fig. 1. In this topology, if congestion occurs on the link between the routers RT0 and RT3 (we call this link LK0-3), some packets will be lost there, and all receivers downstream from LK0-3 will not receive the lost packets. So a multicast loss recovery scheme is needed to retransmit all or some of the lost packets to those receivers (for video multicast, some lost packets that may not be recovered timely for decoding are usually neglected). Ideally, only one receiver downstream from LK0-3 (e.g., RV4) needs to supply the loss information, only one receiver near LK0-3 (e.g. RV1) needs to supply the repair packets, and the repair packets only need to be subcast at LK0-3. If all of these are realized, then the three well-known requirements for an efficient and scalable multicast loss recovery scheme are met: only one retransmission request is produced by RV4 for lost packets; repair packets are locally supplied by RV1; and only the receivers that experienced losses receive the repair packets.

The AIR scheme aims to achieve the ideal loss recovery scenario described above. The router just above a congested link is called a *reference router* in AIR, and the two receivers that need to cooperate to provide repair packets to a loss region are called a *pair*. In the example above, RT0 is the

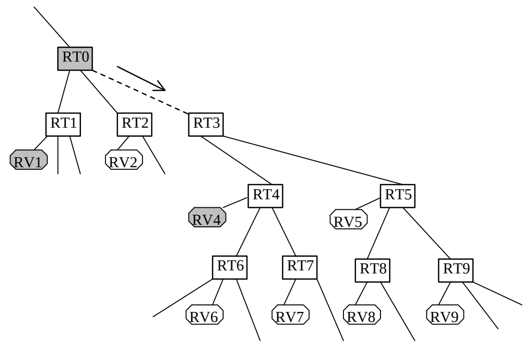

Fig. 1. Topology for Scheme Overview

reference router, while RV1 and RV4 form the *pair*. Since a reference router has the best information about the loss event at the bottleneck, it is responsible for initiating a process to select the *pair*. After the *pair* have been selected (the details of the selection process will be given in the next subsection), the reference router becomes passive and the *pair* play active roles in the rest of the loss recovery process (e.g., ensuring that the recovery process is complete). This helps to relieve the reference router from excessive burden. In addition, unless explicitly requested by a reference router, no receiver produces or sends retransmission requests. Therefore, the proposed scheme suppresses requests in both production and spreading. After the loss event ends, the reference router deletes related states and the *pair* release their roles.

Now we briefly describe how the *pair* is selected in the example above. When the reference router RT0 detects that congestion occurs on LK0-3, it sends a control message downstream to LK0-3. This control message is processed hop by hop and only reaches the nearest receiver along each branch. In this example, only RV4 and RV5 get the message, and then each of them unicasts a response to the reference router RT0. Upon receiving the first response (e.g., from RV4), RT0 sends out another message to seek a repair source. This message is also processed hop by hop and only reaches receivers in a specific region (details in the next subsection). We assume that RV1 and RV2 are the only receivers getting the message, and then each of them unicasts a response to RT0. RT0 only acknowledges the first responder (e.g., RV1). Upon getting the acknowledgement, RV1 knows that it has assumed the role of repair source and starts to contact RV4. RV4 then sends loss information continuously to RV1 (the information is usually not sent for those lost packets that may not be recovered timely for video decoding. For simplicity, we will not mention this in the following text). Meanwhile, RV1 sends repair packets to RT0 to subcast at LK0-3.

In this overview on how the scheme works, many details are omitted, such as the details of processing control messages and how the scheme works when several bottlenecks exist concurrently in a multicast tree. The next subsection answers these questions and presents the details of the scheme.

B. The Scheme

The subsection above introduces the AIR loss recovery process in a simple scenario. This subsection presents the details of the AIR scheme in a general multiple-bottleneck scenario. In a multiple-bottleneck scenario, each bottleneck in a multicast tree initiates a loss recovery process like that introduced in the previous subsection. Since each lost packet is recovered right at its loss site, every receiver eventually gets all the data sent from the multicast source even if most of them do not actively participate in the loss recovery process.

We give some definitions below to assist the description of the scheme:

- *a reference branch*: a multicast branch that is congested and in loss recovery.
- *a reference router*: a router that is just above a reference branch.
- *a pair*: the two receivers selected by a reference router to provide loss information and repair packets.
- *a good branch*: a multicast branch that is not experiencing congestion and did not experience congestion recently (if no AIR loss recovery state exists for a branch, then that branch is a good branch).
- *a stable receiver*: a receiver that has been in a multicast group for a period of time (e.g., 30 seconds).
- *a direct receiver of a router*: a receiver located in the same LAN with the router.
- *NACK_Delay*: a parameter whose value decides the delay between a receiver detecting losses and producing a loss report. Initially, it is set to 0.

1) Finding a Request Source: An optimal request source of a reference branch should be a receiver downstream from and close to the reference branch. Furthermore, there should be no other congested links along the path from the reference branch to the receiver. In this case, the request source will request retransmissions only for the packets lost on the reference branch. However, it is possible that there is not such an optimal request source in existence. In this case, a sub-optimal request source has to be selected. Upon detecting congestion on a branch, the reference router initiates a process to find a request source by sending out a control message. The reference router first sets the ENFORCE flag in the header of the message to 0 to search for an optimal request source. If no optimal request source is found, the reference router sets the ENFORCE flag to 1 to get a sub-optimal request source. The specific procedure for finding a request source is as follows:

(a). The reference router first sends a SEEK_NACK_SRC message downstream to the reference branch with the ENFORCE flag in the packet header set to 0. If the reference router does not receive a NACK_SRC_ACK response from a receiver after a period of time, it sends the SEEK_NACK_SRC message again but with the ENFORCE flag set to 1, and then this message is repeated until a NACK_SRC_ACK response is received.

(b). The SEEK_NACK_SRC message is processed hop by hop. When a router receiving the message has a direct receiver, the router only forwards the message to the direct receiver. Otherwise, the router *only* forwards the message to all downstream *good branches* if the ENFORCE flag is 0; if the ENFORCE flag is 1, the router forwards the message to all downstream branches.

(c). When a stable receiver receives the SEEK_NACK_SRC message, it sends a NACK_SRC_ACK message back to the reference router (this message also contains current loss information).

(d). Upon receiving the first NACK_SRC_ACK response, the reference router sends out a SEEK_REPAIR_SRC message, while subsequent NACK_SRC_ACK responses are discarded.

2) Finding a Repair Source: After the reference router receives the first NACK_SRC_ACK response to its SEEK_NACK_SRC message, it initiates a process to find a repair source by sending out another control message. Two fields, *Start Hop* and *Depth*, in the header of the message assist to find an optimal repair source efficiently. Combined with the Time To Live (TTL), *Start Hop* decides which segment to search along the path from the reference router upstream to the original multicast source. Meanwhile, *Depth* decides how far the search should go downstream along each branch of the segment. Thus TTL, *Start Hop* and *Depth* together define a specific area to seek a repair source.

Usually, the receiver nearest to the reference router but not downstream from the reference branch is a good candidate for the repair source. However, if any link along the path from the multicast source to that receiver is congested and if that congested link is not shared by the path from the multicast source to the reference router, the nearest receiver is not an optimal candidate for the repair source. This is because in this case the nearest receiver possibly has to wait for repair packets itself before it can provide repair packets to its own loss region, and thus, such receivers must be avoided. This is the reason why the message for seeking a repair source, SEEK_REPAIR_SRC, is only sent to downstream *good* branches by each router in the procedure below. Furthermore, receivers too far away from the path connecting the multicast source and the reference router are not good candidates for the repair source either. Therefore, *Depth* is used to restrict the search to a specific depth along each branch on the path. In the following procedure, each round of search covers an area farther away from the reference router than the previous round of search. The specific procedure is as follows:

(a). The reference router sends a SEEK_REPAIR_SRC message to the upstream branch and all downstream *good* branches. The TTL, *Start Hop* and *Depth* fields in the message header are set appropriately (details in the next paragraph). If the reference router does not receive any response to its SEEK_REPAIR_SRC message within a period of time, it initiates another round of search by sending out a new SEEK_REPAIR_SRC message with a modified TTL (details in the next paragraph). This process repeats until at least one REPAIR_SRC_ACK response is received. When the reference router receives the first REPAIR_SRC_ACK message, it responds with a REPAIR_SRC_CONFIRMED message. This message is repeated until repair packets are received.

(b). The SEEK_REPAIR_SRC message is processed hop by hop. When a router receives this message, it may take three possible actions. First, if the router has a direct receiver, the router forwards the message to the direct receiver only. Second, if it does not have a direct receiver and the message comes from upstream, the router forwards the message to all downstream *good* branches. Third, if it does not have a direct receiver and the message comes from downstream, the TTL and the *Start Hop* fields of the message are checked. There are two possible results from the check. (1) If the TTL is greater than the *Start Hop*, the message is forwarded upstream only. (2) If the TTL is less than or equal to the *Start Hop*, besides being forwarded upstream, the message is also copied and forwarded to all downstream *good* branches except the one where the original message came from. Meanwhile, the TTL of each copied message is set to the value of the *Depth* field in the original message.

(c). When a stable receiver receives the SEEK_REPAIR_SRC message, it responds with a REPAIR_ SOURCE_ACK message.

(d). When a receiver receives the REPAIR_SRC_CONFIRMED message from the reference router, it assumes the role of the repair source and sends repair packets to the reference router for subcasting. At the same time, it contacts the request source, whose address is in the SEEK_REPAIR_SRC message received earlier, for further loss information.

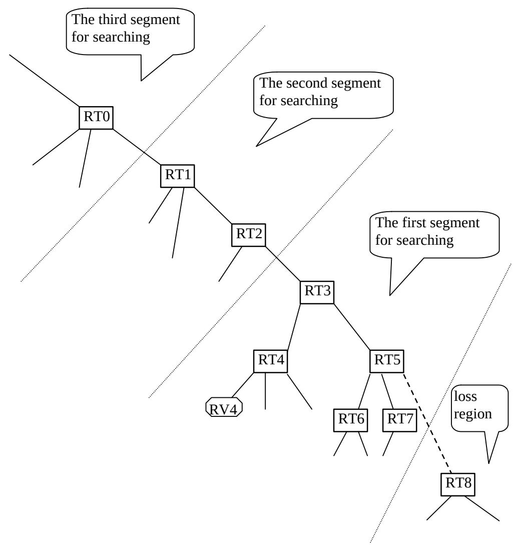
The third segment for searching
The second segment for searching
The first segment for searching
loss region
RT0
RT1
RT2
RT3
RT4
RT5
RV4
RT6
RT7
RT8

Fig. 2. Topology for the Introduction of the Segment-by-Segment Search

A search example is shown in Fig. 2. If it is preferred to search for a repair source two hops per round along the path connecting the reference router and the multicast source, the *Start Hop* should be set to 2. Meanwhile, *Depth* should also be set to a preferred value, such as 3. The TTL then is changed in each round of search to search a different segment. In the first round, the TTL is set to 2. In this case, RT3 and RT5 will search their downstream receivers at most 3 hops away (*Depth* is 3); RT2 discards the message because the TTL is 0 when the message reaches it. If no response is received in the first round of search, the reference router RT5 initiates the second round of search by sending a new SEEK_REPAIR_SRC message with the TTL set to 4. With this new TTL setting, RT3 and RT5 only forward the message upstream, since when the message reaches them the TTL is greater than the *Start Hop*. The TTL becomes less than or equal to the *Start Hop* after the message passes RT3, so RT1 and RT2 copy the message and send a copy to each downstream *good* branch except the one where the original message came. RT0 discards the message because the TTL of the message is 0 when the message reaches it. If there is still no response, the TTL will be set to 6 in the next round of search. This search process continues until a repair source is found.

With the introduction of the *Start Hop* and the *Depth* fields, the search for the repair source is very efficient. Unlike the expanding-ring search, the "segment by segment" search does not involve overlapping search areas.

3) Filtering Interference and Completing the Loss Recovery: After the *pair* is selected, the loss recovery process starts. Now it is necessary to filter out interference among bottlenecks. It is possible that several bottlenecks exist simultaneously in a multicast tree and they may interfere with each other. Although during the pair-search process the chance of potential interference has been considerably reduced by forwarding messages to *good* branches, additional procedures are still needed to deal with the interference that does occur.

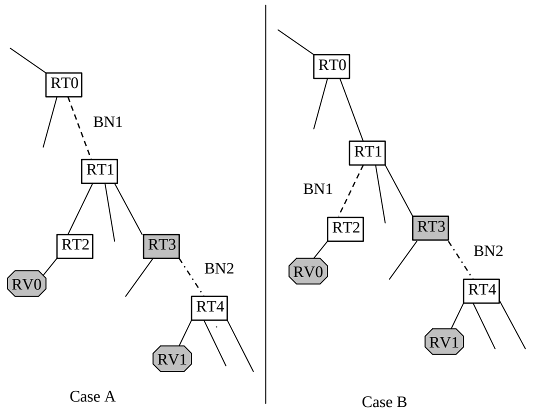

Fig. 3. Topology for the Introduction of Interference

Since a congested branch affects all receivers downstream from it, it is possible that the *pair* of a specific bottleneck is under the influence of several congested branches. In Case A of Fig. 3, the *pair* of the bottleneck BN2, RV0 and RV1, are under the influence of another bottleneck BN1. Because the repair packets from the repair source of BN1 (not shown in the figure) may reach the *pair*, RV0 and RV1, at different times, special situations may arise that cause duplicate repair packets to reach receivers downstream from BN2. For example, if packet A is lost on BN1, a repair packet, A', will be produced by the repair source of BN1 and be subcast to all receivers downstream from BN1. It can be assumed that A' reaches RV0 at time $t1$ and reaches RV1 at time $t2$ ($t1 < t2$). If RV1 produces a loss report between $t1$ and $t2$, it reports the loss of A to RV0. Since RV0 has received A' by that time, it sends A' to RT3 for subcasting. The consequence is duplicate A's for receivers downstream from BN2.

To avoid the above interference problem, the AIR scheme uses the NACK_Delay parameter to compensate for the time difference between $t1$ and $t2$. When a request source receives duplicate repair packets caused by interference, it increases its NACK_Delay parameter by a small step. The request source then waits for a period of time decided by the current value of its NACK_Delay parameter before producing a loss report after detecting losses. In the previous example, if the NACK_Delay parameter of RV1 has been adjusted by other duplicate repair packets before RV1 detects loss A, then RV1 will wait for a period of time before producing a loss report for A. Now, it is possible that RV1 will try to produce the loss report after time $t2$. However, by that time A' has arrived at RV1, so no loss report will be produced. Therefore, the event of duplicate A's is avoided. Fig. 4 shows the timelines for the original case and the modified case where the NACK_Delay parameter compensates for the time difference. The final value of NACK_Delay that can avoid the duplicate repair packets caused by interference depends on the difference between $t1$ and $t2$. The difference between $t1$ and $t2$ is decided by the difference of the delays along the two paths to the *pair*, RV0 and RV1, from the interference bottleneck BN1.

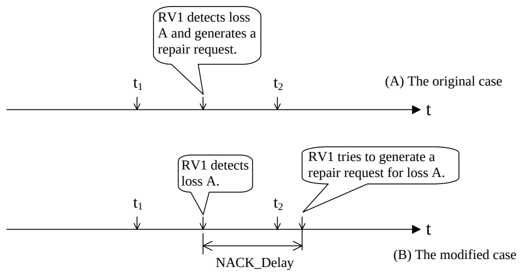
RV1 detects loss A and generates a repair request.
t1
t2
(A) The original case
t
RV1 detects loss A.
RV1 tries to generate a repair request for loss A.
t1
t2
t
NACK_Delay
(B) The modified case

Fig. 4. Timelines for the Introduction of Interference

Another form of interference only causes some delay in recovery but not duplicate repair packets. Case B in Fig. 3 shows an example, where RV0 and RV1 form a *pair*. In this case, the interference bottleneck BN1 only affects RV0. If a packet is lost both on BN1 and BN2, RV0 needs to wait for the repair packet itself before it can produce a repair packet for the receivers downstream from BN2, so an additional small recovery latency is added.

In addition to filtering out interference, how to complete the recovery process is another important issue. With AIR, the request source decides when the loss recovery process completes, since losses on the reference branch are experienced by the request source but not the repair source. When the request source detects the end of the loss event, it notifies the repair source, and then both of them release their roles.

Another concern is that the recovery process must be error tolerant. For this purpose, control messages may be repeated if no acknowledgement is received. Additionally, if either part of the *pair* fails, the other part must be able to detect the failure and start to fix it by informing the reference router.

The specific procedures for accomplishing the above goals are given below. After the repair source has been selected by the REPAIR_SRC_CONFIRMED message:

(a). The repair source sends a NACK_SRC_CONFIRMED message to the request source whose address is in the SEEK_REPAIR_SRC message received earlier. The request source responds with a NACK_INFO message immediately. However, if the value of the NACK_Delay parameter of the request source is not 0, each subsequent NACK_INFO message is delayed for a period of time decided by the value of the parameter.

(b). If the repair source has repair packets for all or some of the losses indicated in the NACK_INFO message, it sends the repair packets to the reference router for subcasting. The repair source ignores the requests that it can not help now (the request source may send the ignored requests again if the lost packets are not recovered by repair packets from other repair sources some time later).

(c). When the request source receives duplicate repair packets caused by interference, it increases its NACK_Delay parameter by a small step.

(d). If the request source does not experience losses anymore or losses are recovered before the request source produces any NACK_INFO message, the request source enters the completion state. If the request source can stay in the completion state for a specified period of time, it sends a RECOVERY_COMPLETE message to the repair source. This message is repeated until a response is obtained. When the repair source receives the RECOVERY_COMPLETE message, it responds with a RECOVERY_COMPLETE_CONFIRMED message and deletes related states to release its role.

(e). If the repair source does not receive NACK_INFO within a period of time before receiving the RECOVERY_COMPLETE message, it contacts the request source. If the request source does not respond, the repair source notifies the reference router to find a new request source. Meanwhile, if the request source does not receive repair packets for its NACK_INFO message within a period of time, it contacts the repair source. If the repair source does not respond, the request source also notifies the reference router to find a new repair source.

IV. Controlling Repair Traffic Rate

The preceding section presents the general AIR scheme that does not apply rate control over its repair traffic. In this section we introduce the repair traffic interference problem in multicast and present the mechanism used by AIR to deal with this problem for enhancing its efficiency and scalability, which are important for supporting a large-scale video multicast.

A. The Repair Traffic Interference Problem

Without appropriate rate control over the repair traffic generated by them across a multicast tree, existing distributed multicast loss recovery schemes can exacerbate congestion on a bottleneck. When congestion occurs on a bottleneck, the queue overflows and packets are lost. With an existing distributed multicast loss recovery scheme, receivers usually instantly submit the repair requests after detecting losses [2]. With distributed loss recovery, repair packets are provided locally, so the repair traffic can reach the bottleneck while the congestion is still severe. Consequently, the congestion on the bottleneck is exacerbated and the network utilization is reduced. The interference problem becomes more serious when the congestion on a bottleneck is more severe. This is because more severe congestion implies more losses, while more losses cause higher rate of repair traffic.

We can analyze the issue over the partial multicast tree shown in Fig. 1. We consider the situation when the link between RT4 and RT6 (LK4-6) becomes congested for a period of time and a distributed multicast loss recovery scheme with local recovery (e.g. SRM [7]) is applied to the multicast session. RV4 is the receiver that is located closest and upstream from LK4-6, so RV4 will be the repair source. In this case, when losses occur on LK4-6, RV4 will generate repair traffic (*RT*) that traverses the already congested link LK4-6 to reach receivers downstream from LK4-6 (e.g., RV6). If the rate of the repair traffic generated by RV4 is not controlled, what will be the consequence? We can assume that the congestion event begins at $t1$ and ends at $t2$. Thus, some packets are lost after $t1$. Packet losses bring the loss recovery scheme into action. So RV6 or another receiver downstream from LK4-6 reports losses to the group and RV4 starts to generate repair traffic a short time later. We can assume that the repair traffic arrives at LK4-6 at $t1 + \delta$. Usually δ is small because of the introduction of distributed loss recovery, which enables local receivers/agents to provide repair packets. So it is possible that $t1 + \delta$ is less than $t2$. In this case, the total traffic rate at the bottleneck becomes:

$$R = R_{ET} + R_{RT}$$

where *ET* denotes the "existing traffic" at the bottleneck except the repair traffic (*RT* denotes the repair traffic). With heavier traffic on LK4-6, the congestion there is worsened. With worsened congestion, more packets are lost. With more losses, the repair traffic rate, R_{RT}, becomes higher. Therefore, R is further increased by R_{RT}. The increased R worsens the congestion further and causes even more losses, which implies even heavier repair traffic (higher R_{RT}) a moment later. This devastating process repeats until the rate of the existing traffic, R_{ET}, is reduced after time $t2$ (due to traffic fluctuation, departing sessions, or congestion control over some sessions). The network utilization is significantly reduced in this process. Consequently, efficiency and scalability of the loss recovery scheme are affected adversely.

In the analysis above we only used a very simple scenario. In reality the situation can be much worse, especially when there are many concurrent multicast sessions and the multicast sessions have a large scale. First, in this case, there may be many concurrent congestion events in a multicast tree, so a congested link may be under the influence of several repair sources. Second, the control traffic of a loss recovery scheme may also spread across a multicast tree, so a bottleneck may

[2]Requests may be intentionally delayed for other reasons such as avoiding redundant requests but not for controlling repair traffi c rate.

also be under the influence of control traffic.

To alleviate the interference of repair traffic on bottlenecks for improving efficiency and scalability, the rate of the repair traffic generated by a distributed multicast loss recovery scheme must be controlled. The next subsection presents the unique mechanism used by AIR to control its repair traffic.

B. Mechanisms for Controlling Repair Traffic

In general, there are two methods for controlling the repair traffic for a loss region. The first is to enhance the repair source, while the second is to enhance the request source. With the first method, the repair traffic can be directly controlled by the repair source according to some criteria (e.g., congestion feedback from the request source). With the second method, the repair traffic cannot be directly controlled, because the request source does not generate the repair traffic itself. Although the first method seems more straightforward, the second method is more effective and efficient. There are two reasons:

- Congestion feedback is saved if the second method is adopted. This is because the request source can infer congestion information from the traffic coming from the bottleneck, while the repair source does not have the traffic to analyze.
- Confusion is avoided with the second method. If the repair source is responsible for controlling the repair traffic rate, it has to intentionally introduce some delay in responding to repair requests for controlling the repair traffic rate. Without additional procedures, this delay in response can cause confusion for the request source, because in this case, the request source has no way to know whether the responses to its requests have been intentionally delayed by the repair source or its requests (or the corresponding repair packets) have been lost in transmission. This confusing situation does not happen if the request source is responsible for controlling the repair traffic rate.

The key mechanism to control the repair traffic by the request source is to control the submission of repair requests. With this method, the request source maintains a request buffer. When the request source needs to decrease the rate of the repair traffic, it simply decreases the rate of submitting repair requests. Some repair requests are then buffered. When congestion is alleviated on the bottleneck, the request source increases its rate of submitting repair requests.

We can take a look at the example in the previous subsection. If the submission of repair requests is properly controlled, RV6 will hold most repair requests when the congestion is still severe (sensed with heavy losses) on LK4-6. Only after the congestion on LK4-6 has been alleviated, RV6 starts to send out buffered repair requests. In this way, the total traffic rate, R, on LK4-6 will not devastatingly increase when a congestion event occurs, so the congestion on the bottleneck will not be exacerbated.

After the above introduction of the general methods for controlling repair traffic rate in multicast, we present the specific mechanism used by AIR in controlling its submission of repair requests at a request source. There are two questions about controlling the submission of repair requests. The first question is when to submit, while the second is how fast to submit (i.e., the submission rate). With AIR, only if the number of in-order packets received by a request source from the original multicast source is greater than a threshold, $Thresh_{submit}$, the request source submits a sequence of repair requests [3]. In each request submission event, the number of losses that can be reported is controlled by another scheme parameter, Max_{loss}, which is the upper limit for the number of losses that can be reported in each event. Generally, greater $Thresh_{submit}$ implies less interference of repair traffic on the corresponding bottleneck, because more in-order packets indicate lighter congestion. Meanwhile, smaller Max_{loss} implies less repair traffic interference, since Max_{loss} indirectly limits the number of repair packets that can be sent from the repair source each time. However, being too conservative in submitting repair requests may cause longer recovery latency. In our experiments in Section 5, $Thresh_{submit}$ and Max_{loss} are set to 3 and 4, respectively.

V. Analysis of The Scheme

This section analyzes the scheme and compares it with two well-known multicast loss recovery schemes, SRM [7] and LMS [17]. SRM, LMS and AIR have one important common characteristic: enabling receivers as potential repair sources instead of having data caching at router sites. The three schemes are first analyzed generally and then numerically over binary trees and an example multicast tree. For simplicity, in the analysis we do not consider the effect of the rate control over repair traffic with AIR. The simulations in Section VI will show that the unique feature of rate control over repair traffic with AIR enhances the efficiency of the scheme without increasing its recovery latency. Therefore, the conclusions drawn in this section are not weakened by the neglect of the rate control over repair traffic with AIR.

For request suppression, SRM uses distance-based random timers to suppress the production of requests, while LMS uses replier links to suppress the spreading of requests but without suppressing their production. With AIR, requests are suppressed both in production and in spreading. So AIR is better than SRM and LMS in request suppression. For local recovery, SRM is the best, since every receiver in a multicast session competes to repair each loss in the session, but the price is the flooding of each retransmission request over the multicast tree. With LMS, only the nearest receiver of a router is considered as a potential repair source, so it is possible that a good repair source may be missed. With AIR, regions near a router are searched for a potential repair source, so the chance of missing a good repair source is quite low. For retransmission scoping, SRM floods the whole multicast tree with repair packets, while LMS may leak repair packets to unrelated receivers in some situations. With AIR, repair packets are subcast at the link where losses occur, so leakage is impossible. For recovery latency, LMS is usually quick

[3]If the session ends before a request source completes its loss recovery, this rule is ignored, since no original packets will fbw in after a session ends.

(the choice of replier links may affect its performance), but the price is to store replier-link related states in each router across the multicast tree of a multicast session, and these states exist during the whole multicast session. SRM uses distance-based random timers for both request suppression and reply suppression, so considerable delay is introduced in loss recovery. AIR needs to find a *pair* at the beginning of a congestion event. The search process takes some time, but since the search is usually local, it is quick. Furthermore, after the *pair* are selected, retransmission requests are unicast from the request source to the repair source without any extra processing along the path. However, with LMS, every request needs to be extraordinarily processed at each hop (e.g., being checked if coming from the replier link). Therefore, AIR is the quickest after the *pair* is ready. In terms of adaptability to changing topologies, both LMS and AIR are excellent because of the assistance from underlying networks while SRM is not so good because it may need timer adjustment and new distance information.

After the general analysis above, we numerically analyze the three schemes over binary trees and an example multicast tree. The numerical analysis has the following assumptions:

1) 50 packets are lost on one bottleneck in one congestion event.
2) On average, 5 lost packets are reported in each retransmission request. Hence there are 50/5 = 10 independent retransmission requests for the 50 lost packets.
3) Message processing delay and queuing delay are the same for all schemes.
4) Path delay is counted as number of hops for simplicity.

Before proceeding to the analysis, we define the following performance indexes [4]:

1) The *request suppression index* is the total number of hops that all requests traverse for a single loss.
2) The *local recovery index* is the number of hops between the repair source and the nearest receiver that needs repair packets from it.
3) The *retransmission scoping index* is the percentage ratio of the number of receivers that do not need the repair packets but receive them over the number of receivers that do need the repair packets.
4) The *recovery latency index* is the total number of hops that the request and the repair packet for a single loss traverse plus other delays in hops introduced by a specific scheme.

Obviously, for each performance index, lower value shows better performance. Table I and Table II presents the final comparison results of the three schemes over binary trees and the example multicast tree shown in Fig. 5, respectively. As shown in these tables, AIR has a significantly better overall performance in efficiency than the other two schemes. Furthermore, Table I shows that the AIR scheme is not affected by the scale of the tree, while the other two schemes are affected adversely. Especially, the recovery latency of AIR does not increase as the scale of the tree increases. This is a great advantage for supporting large-scale video multicast, which requires not only high efficiency and scalability but also low recovery latency.

[4]To our knowledge, no numerical criterion has been defined for evaluating the efficiency of a multicast loss recovery scheme.

We now demonstrate how to calculate the performance indexes of the three schemes over a n-level binary tree (a 3-level binary tree shown in Fig. 6 is used for reference). We assume that the congestion occurs on the last hop of the binary tree (it has been shown that most losses in MBone happen in local networks [24]). Because of the symmetry of the tree, it can be assumed that the losses occur on any last hop of the tree (in the 3-level binary tree, we assume that losses occur on the link between RT3 and RT5, LK3-5). One factor that complicates the calculation of the indexes of LMS is the choice of the replier link of a router. We assume that the two downstream links of a router in a binary tree are equally likely to be chosen as the replier link with LMS.

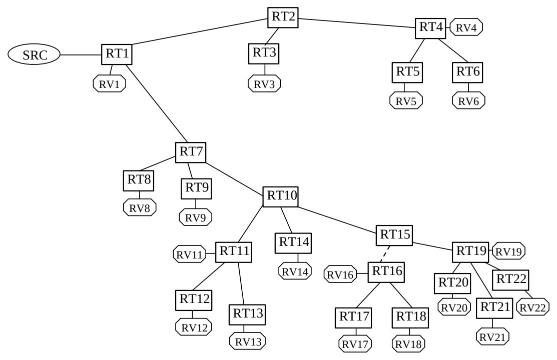

Fig. 5. The Example Multicast Tree

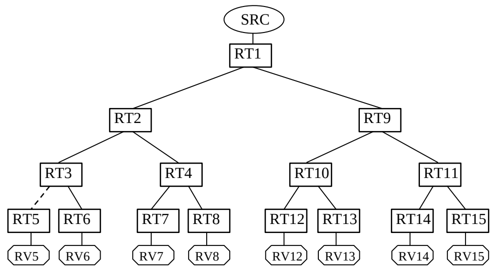

Fig. 6. 3-level Binary Tree for Analysis

First we calculate the request suppression index of each scheme. With SRM, each request is multicast to the whole group and traverses the $2(2^n-1)$ hops of a n-level binary tree. With k losses reported in each request, the index of SRM is $\frac{2(2^n-1)}{k}$ (we assumed $k=5$ in our case). With LMS, we first take the 3-level tree as an example (as noted before, assuming losses on LK3-5 and 5 losses reported in each request). In this case, either LK3-6 or LK3-5 can be chosen as the replier link of RT3 with a probability of 0.5. If LK3-6 is chosen as the replier link, the request from RV5 traverses 2 hops to reach RV6, so the request suppression index is 2/5=0.4. However, if LK3-5 is chosen as the replier link, the request from RV5 will go upstream to RT2. Again, with a probability of 0.5, the request will go downstream to reach RV7 or RV8 (i.e., LK2-4 is chosen as the replier link of RT2) or go upstream to reach

TABLE I

PERFORMANCE INDEXES BASED ON A N-LEVEL BINARY TREE: LAST-HOP CONGESTION

	n = 3			$n \geq 3$		
Scheme	SRM	LMS	AIR	SRM	LMS	AIR
Request Suppression	2.8	0.55	0.4	$0.4(2^n-1)$	$0.8-(0.2n+0.4)(0.5)^{n-1}$	0.4
Local Recovery	2	2.75	2	2	$4-(n+2)(0.5)^{n-1}$	2
Retransmission Scoping	700	100	0	$100(2^n-1)$	$50(n-1)$	0
Recovery Latency	19	5.5	4.5	$4n+7$	$8-(2n+4)(0.5)^{n-1}$	4.5

TABLE II

PERFORMANCE INDEXES BASED ON THE EXAMPLE MULTICAST TREE: CONGESTION ON LK15-16

Scheme	SRM	LMS	AIR
Request Suppression	4.2	0.9	0.4
Local Recovery	2	2.5	2
Retransmission Scoping	500	67	0
Recovery Latency	23	5.0	4.5

the multicast source (i.e., LK2-3 is chosen as the replier link of RT2). In the former case, the index is 4/5=0.8, while in the latter case, the index is 3/5=0.6. Therefore, the average index of LMS over the 3-level binary tree is $\{2\times\frac{1}{2}+4\times(\frac{1}{2})^2+3\times(\frac{1}{2})^2\}\times\frac{1}{5}=0.55$ (as analyzed above, RV6, RV7/RV8, and the multicast source have a probability of $\frac{1}{2},(\frac{1}{2})^2$ and $(\frac{1}{2})^2$, respectively, to become the repair source). We can apply the same logic to a n-level binary tree with k losses reported in each request. In this case, the index becomes a series:

$$\begin{aligned}&\left\{2\times\frac{1}{2}+4\times\left(\frac{1}{2}\right)^2+6\times\left(\frac{1}{2}\right)^3+\ldots+\right.\\&\left.2(n-1)\left(\frac{1}{2}\right)^{n-1}+n\left(\frac{1}{2}\right)^{n-1}\right\}\times\frac{1}{k}\\=&\left\{n\left(\frac{1}{2}\right)^{n-1}+\sum_{m=1}^{n-1}2m\left(\frac{1}{2}\right)^m\right\}\times\frac{1}{k}\\=&\left\{4-(n+2)\left(\frac{1}{2}\right)^{n-1}\right\}\times\frac{1}{k}\qquad(1)\end{aligned}$$

With AIR, each request traverse 2 hops to reach the repair source (in the 3-level tree case, the request from RV5 traverses 2 hops to reach RV6), so the index of AIR is $\frac{2}{k}$ if k losses are reported in each request.

For local recovery index, with SRM, the repair source is 2 hops away from the nearest receiver that needs repair packets (from RV6 to RV5 in the 3-binary tree case), so the index of SRM is 2 hops. With LMS, in the 3-level binary tree case, RV6, RV7/RV8, and the multicast source have a probability of $\frac{1}{2},(\frac{1}{2})^2$ and $(\frac{1}{2})^2$, respectively, to become the repair source. Therefore, the average local recovery index of LMS over the 3-level binary tree is $2\times\frac{1}{2}+4\times(\frac{1}{2})^2+3\times(\frac{1}{2})^2=2.75$ hops. Using the same logic over a n-level binary tree, we get the index of LMS:

$$\begin{aligned}&2\times\frac{1}{2}+4\times\left(\frac{1}{2}\right)^2+6\times\left(\frac{1}{2}\right)^3+\ldots+\\&2(n-1)\left(\frac{1}{2}\right)^{n-1}+n\left(\frac{1}{2}\right)^{n-1}\\=&\ n\left(\frac{1}{2}\right)^{n-1}+\sum_{m=1}^{n-1}2m\left(\frac{1}{2}\right)^m\\=&\ 4-(n+2)\left(\frac{1}{2}\right)^{n-1}\qquad(2)\end{aligned}$$

With AIR, the repair source is 2 hops away from the nearest receiver that needs repair packets (from RV6 to RV5 in the 3-level binary tree case), so the local recovery index of AIR is 2 hops.

For retransmission scoping, the performance index of SRM over a n-level binary tree is: $(2^n-1)/1\times100\%$, since only 1 receiver needs the repair packets and the other (2^n-1) receivers will get them unnecessarily because of the flooding of the repair packets over the tree. With LMS and in the 3-level binary tree case, with a probability of 0.5, RV6 will be the repair source. In this case, the index is 0 (no leakage). With a probability of 0.25, RV7 or RV8 will become the repair source. In this case, the index is 1/1x100%=100% (RV6 will get the repair packets unnecessarily). Also with a probability of 0.25, the multicast source will be the repair source. In this case, the index is 3/1x100%=300% (RV6, RV7, and RV8 will get the repair packets unnecessarily). Therefore, the average index of LMS over the 3-level binary tree is $\{0\times\frac{1}{2}+1\times(\frac{1}{2})^2+3\times(\frac{1}{2})^2\}\times100\%=100\%$. Applying similar calculation over the n-level binary tree, we get the index of LMS:

$$\begin{aligned}&0\times\frac{1}{2}+1\times\left(\frac{1}{2}\right)^2+3\times\left(\frac{1}{2}\right)^3+\ldots+\\&(2^{n-2}-1)\left(\frac{1}{2}\right)^{n-1}+(2^{n-1}-1)\left(\frac{1}{2}\right)^{n-1}\\=&\ (2^{n-1}-1)\left(\frac{1}{2}\right)^{n-1}+\sum_{m=1}^{n-1}(2^{m-1}-1)\left(\frac{1}{2}\right)^m\\=&\ \frac{1}{2}(n-1)\times100\%\qquad(3)\end{aligned}$$

With AIR, there is no leakage (in the 3-level binary tree case, repair packets from RV6 only reaches RV5), so the index of AIR is 0.

Last, we calculate the recovery latency index of each scheme. In SRM, we assume that the timer parameters C1, C2, D1 and D2 are 1, 6, 1 and 1, respectively (these values are used in an example in SRM [7]). In the 3-level binary case, the performance index of SRM is: (3+3x7)/2 + (2+2x2)/2 + 2 + 2 = 19 hops. Here, (3+3x7)/2 is the average delay introduced by the request suppression timer (RV5 is 3 hops away from the multicast source), and (2+2x2)/2 is the average delay introduced by the reply suppression timer (RV6 is 2 hops away from RV5). The last 2+2 is the path delay of the request and the repair packet between RV5 and RV6. Extending this logic to a n-level binary tree, we get the recovery latency index of SRM:

$$\begin{aligned} & (n + n \times 7) \times \frac{1}{2} + (2 + 2 \times 2) \times \frac{1}{2} + 2 + 2 \\ = \; & 4n + 7 \end{aligned} \tag{4}$$

With LMS, we first take a look at the 3-level binary tree case. If RV6 is the repair source, the index is 2+2=4 (2 hops for each request and 2 hops for the corresponding repair packet); if RV7 or RV8 is the repair source, the index is 4+4=8; if the multicast source is the repair source, the index becomes 3+3=6. Therefore, the average recovery latency index of LMS over the 3-level binary tree is $4\times\frac{1}{2}+8\times(\frac{1}{2})^2+6\times(\frac{1}{2})^2 = 5.5$ hops. Similarly, the index of LMS over a n-level binary tree is:

$$\begin{aligned} & 4 \times \frac{1}{2} + 8 \times \left(\frac{1}{2}\right)^2 + 12 \times \left(\frac{1}{2}\right)^3 + \ldots + \\ & 4(n-1)\left(\frac{1}{2}\right)^{n-1} + 2n\left(\frac{1}{2}\right)^{n-1} \\ = \; & 2n\left(\frac{1}{2}\right)^{n-1} + \sum_{m=1}^{n-1} 4m\left(\frac{1}{2}\right)^m \\ = \; & 8 - (2n+4)\left(\frac{1}{2}\right)^{n-1} \end{aligned} \tag{5}$$

With AIR, the recovery latency of some lost packets is affected by the delay in seeking a request source and a repair source. The search delay is 5 hops in a n-level binary tree (2 hops to find the request source and 3 hops to find and confirm the repair source). If we assume that the average link delay (transmission, queuing plus propagation) is 30ms, the search delay is about 5x30=150ms. If we consider that the session data rate is 512kb/s and the packet size is 1024 bytes, the maximum number of packets of the session that can be lost in 150ms is (150/1000)/(1024x8/512000)=9.4. So when the search is finished, there are at most 10 packets of the session lost. Thus 1 request is affected by the search delay (when the second request needs to be submitted, the request source has been found). Therefore, the performance index of the proposed scheme is: (1x5x(5+4)+9x5x4)/50=4.5 (50 losses are recovered, the first 1x5 repair packets have a latency of 5+4 hops, while the last 9x5 repair packets have a latency of 4 hops each). As observed above, the recovery latency index of AIR may change slightly with the *pair* search delay and the session rate. However, the index of AIR does not increase when the scale of the tree increases.

The good performance of AIR comes from its active and on-demand construction and maintenance of minimum recovery structures. Meanwhile, AIR also has several disadvantages that existing schemes suffer from. If the loss rate is very low in a multicast tree, the overhead of finding the *pair* for a loss event (the overhead of setting up and maintaining states in routers or receivers across the multicast tree in other schemes, such as LMS and SRM) is significant compared to the recovered packets. In addition, the search process may increase the recovery latency considerably for AIR if the *pair* cannot be found locally. So at the cost of permanent states, other schemes (e.g., LMS) may outperform AIR in loss recovery latency in some situations. However, local request and repair sources usually exist for a loss event in a typical multicast session, since it has been shown that losses mainly occur in local networks in MBone [24]. Another common disadvantage for most network-assisted schemes is the “soft state” problem: before some “soft states” expire, a failure or un-optimal state may not be recovered. With AIR, the possibly failed part of a *pair* cannot be detected until the state maintained by the other part expires. Similarly, other network-assisted schemes suffer from the same kind of problems. Meanwhile, the locations and times of the loss events in a multicast session can affect the efficiency and effectiveness of a multicast loss recovery scheme. For example, if losses occur heavily in a small part of a multicast tree, some receivers/agents (the *pair* in AIR and the request and repair sources in some other schemes) can be overloaded. Besides, if the topology of the multicast tree changes dramatically and continuously in a multicast session, a multicast loss recovery scheme may work in a sub-optimal state temporarily (e.g., with AIR, the *pair*s may be temporarily sub-optimal; with LMS, the replier links may be temporarily sub-optimal; SRM may temporarily not have the full distance information).

Another thing that needs to be mentioned is that generally there are two kinds of loss sources in reliable packet transmission: congestion losses and bit-error losses. Congestion losses come from queue overflows in networks, while bit errors happen during packet processing and transmission. Thanks to the high quality of the links in existing networks, bit-error losses are negligibly low compared to congestion losses. The active mechanism introduced in Section III only deals with congestion losses but not bit-error losses. If total reliability is required, receivers send requests to the original multicast source to recover the losses that are not recovered by the active mechanism (receivers can use timers to suppress redundant requests as in SRM [7]). This does not significantly affect the total performance of the proposed scheme because bit-error losses are negligibly low compared to congestion losses.

VI. Simulation Results

The previous section analyzes the AIR scheme generally and numerically for its efficiency, scalability and recovery latency; this section uses simulations to further test the performance of AIR. The simulations are over the example multicast tree shown in Fig. 5. In the tree, the capacity of LK7-10 and LK15-16 is 512kb/s, while the capacity of other links is 1024kb/s. The link delay of LK1-2, LK1-7 and LK2-4 is 100ms, and the link delay of LK7-10 and LK10-15 is 50ms, while LK15-19 has a delay of 20ms. Other links have a delay of 10ms. When necessary, a traffic source with exponentially distributed idle and burst times is used to generate congestion on a bottleneck in the simulations.

A. The General AIR Scheme

We first test the general AIR scheme without rate control applied over the repair traffic.

1) One-Bottleneck Scenario: In this scenario, only LK15-16 is congested. The congestion-causing traffic from RV14 to RV18 starts at the first second and stops at the 10th second. The average rate of the congestion-causing traffic is modified to observe the performance of the AIR scheme under different degrees of congestion. Table III shows the total number of losses, the average recovery latency and the total number of duplicate repair packets observed by RV16 with different congestion-causing traffic rates.

From this table, the AIR scheme has low recovery latency, which is a big advantage for video multicast. When the congestion is light, the average loss recovery latency is less than 300ms. Even when the congestion is considerably severe, the average recovery latency is only a little over 600ms. However, the loss recovery latency of 286.8ms in the 70kb/s case is still higher than expected, since the round trip propagation time between RV16 and RV19 (the *pair*) is about 60ms. This can be explained by the following two observations. First, packets will be significantly delayed when the link is congested and the queue is full. Second, repair packets can also be lost at the bottleneck if the congestion is still going on when they reach the bottleneck. The distribution of the recovery latency for the 70kb/s case is shown in Fig. 7. As shown in this figure, most lost packets are recovered between 100ms and 200ms, which is the propagation delay plus processing and queuing delay. Additionally, there are also lost packets that are recovered with a latency of more than 900ms, which is caused by the losses of repair packets.

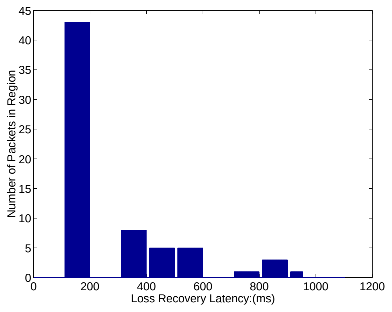

Fig. 7. Recovery Latency Distribution: One-Bottleneck 70kb/s Case

2) Two-Bottleneck Scenario: In this scenario, both LK7-10 and LK15-16 are congested. The difference between the times at which the repair packets from the interference bottleneck LK7-10 reach the *pair* of LK15-16, RV19 and RV16, decides the number of duplicate repair packets that needs to be detected before the NACK_Delay parameter of RV16 is adjusted to a right value (see the example in Section III). The greater the time difference, the higher the interference. To test AIR with various degrees of interference, simulations were conducted with LK15-16 set to different delays. The simulation results for two cases, 10-second and 20-second congestion duration, are shown in Table IV.

In Table IV, the average recovery latency and the number of duplicate repair packets, in general, go up with the increase of the delay on LK15-16. This can be explained by the following observations. The difference between the times at which the repair packets from LK7-10 reach the *pair* of LK15-16, RV16 and RV19, is partly decided by the delay of LK15-16. Larger difference between the times means heavier interference. With heavier interference, RV16 needs to detect more duplicate repair packets before it can adjust its NACK_Delay parameter to a right value. So the number of duplicate repair packets may go up with the increase of the delay on LK15-16. Meanwhile, both the increased delay on LK15-16 and the consequently increased value of the NACK_Delay parameter contribute to the increase of the average recovery latency when the delay of LK15-16 is increased. However, there is one exception in the table. When the delay of LK15-16 is 20ms, neither the recovery latency nor the number of duplicate repair packets is significantly greater than that in the 10ms-delay case. This is because the delay of LK15-19 is also 20ms. In this case, the repair packets from the interference bottleneck LK7-10 reach the *pair* of LK15-16, RV16 and RV19, almost at the same time if the difference of the queuing delays along the two paths to RV16 and RV19 is not considered. Therefore, the interference is the least when the delay of LK15-16 is 20ms.

Another observation from Table IV is that the length of a congestion event does not significantly affect the number of duplicate repair packets. From the table, we can find that there is little or no increase in the number of duplicate repair packets when the duration of the congestion event increases from 10 seconds to 20 seconds. As pointed out and explained in Section III, the number of duplicate repair packets produced in a congestion event is primarily decided by the difference between the delays along the two paths from the interference bottleneck to the request source and the repair source of the interfered bottleneck, while this difference does not depend on congestion duration.

The final observation from Table IV is that the total number of losses changes with the delay of LK15-16. This is because the repair traffic reaches the bottleneck at a different time if the delay of LK15-16 is different (requests from RV16 traverse LK15-16 to reach RV19). Therefore, with a different delay on LK15-16, the repair traffic may interfere on the bottleneck differently. For example, if the repair traffic reaches the bottleneck when the bottleneck is still severely congested, more packets will be lost. The next subsection will show that the rate control AIR applies over its repair traffic can significantly reduce the repair traffic interference on bottlenecks.

B. The Enhanced AIR Scheme

The enhanced AIR scheme applies rate control over its repair traffic to reduce the interference of repair traffic on bottlenecks. We use two criteria to evaluate the enhanced AIR scheme and compare it with the general AIR scheme: the number of losses on a bottleneck and the average recovery latency.

To test the enhanced AIR scheme with various lengths of congestion on the bottleneck LK15-16, we change the average burst/idle time of the congestion-causing traffic from 0.2 to 2.0 seconds with a step of 0.2 second in a series of

TABLE III
SIMULATION RESULTS OF THE ONE-BOTTLENECK SCENARIO

Congestion Source Rate (kb/s)	Number of Losses	Number of Duplicate Repairs	Average Repair Latency (ms)
70	66	0	286.8
80	112	0	427.3
90	180	0	592.8
100	209	0	630.3

TABLE IV
SIMULATION RESULTS OF THE TWO-BOTTLENECK SCENARIO

	Congestion Duration: 10 Seconds					Congestion Duration: 20 Seconds				
Delay of LK15-16 (ms)	10	20	30	40	50	10	20	30	40	50
Number of Losses	83	110	145	126	142	119	139	177	155	171
Average Repair Latency (ms)	323.1	320.3	341.8	413.9	438.3	324.3	326.1	343.5	402.8	424.8
Number of Duplicate Repairs	11	8	14	14	23	12	8	14	14	24

simulations. With each average burst/idle time setting, the simulation duration is 10 burst-idle cycles. For example, if the average burst/idle time is 0.2 second, then the congestion-causing traffic continues for (0.2+0.2)x10=4 seconds. With each average burst/idle time setting, seven simulations were conducted. In the seven simulations, the average rate of the congestion-causing traffic changes from 70 to 100 kb/s with a step of 5 kb/s for testing the AIR scheme with different degrees of congestion. After that, the number of losses and the average recovery latency are averaged over the seven cases. The simulation results are shown in Fig. 8 and Fig. 9. Fig. 8 shows the number of losses over the average per-burst time, while Fig. 9 shows the average recovery latency over the average per-burst time.

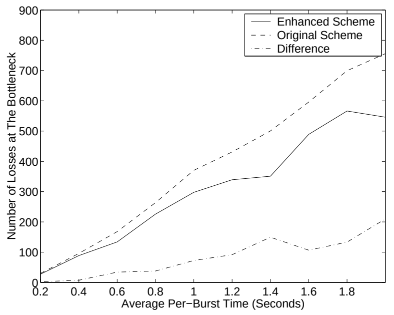

Fig. 8. Number of Losses Over Average Congestion Duration

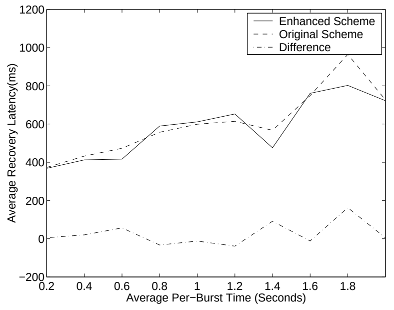

Fig. 9. Recovery Latency Over Average Congestion Duration

As shown in Fig. 8, the enhanced AIR scheme significantly reduces the number of losses on the bottleneck. In some cases, the number of losses is reduced by nearly 30%. This is because with the enhanced AIR scheme, the controlled repair traffic has less interference on the bottleneck. Moreover, the performance of recovery latency is not sacrificed for the reduced number of losses. As shown in Fig. 9, the average recovery latency of the enhanced AIR scheme is even shorter than that of the general AIR scheme in most cases. Two observations can explain this. First, the reduced number of losses on the bottleneck implies less severe congestion there. With less severe congestion, the queuing delay at the bottleneck is reduced. Second, with less severe congestion on the bottleneck, the number of repair packets that are also lost at the bottleneck is reduced. In summary, the efficiency and the scalability of the scheme are considerably enhanced with its unique rate control over its repair traffic, while its recovery latency is still kept low. This strengthens the ability of the proposed scheme to support large-scale video multicast.

C. *Streaming Video Multicast*

Finally, we use the simulations of streaming video multicast to test the enhanced AIR scheme. We simulated the the transmission of 60-second streaming video over the example multicast tree with different degrees of congestion on the bottleneck LK15-16. Fig. 10 shows the PSNR of the received video at receiver 16 over the average rate of the congestion-causing traffic. The results of four cases are shown: without loss recovery, with a recovery latency threshold of 500ms (only lost packets recovered within 500ms are used in decoding), with a recovery latency threshold of 1000ms, and with a recovery latency threshold of 2000ms [5].

We can find from Fig. 10 that in general, the PSNR of the received video decreases as the degree the congestion increases. This is expected because more packets are lost with higher degree of congestion and more losses imply poorer video quality. The figure also shows that there are significant gains when AIR loss recovery is applied to the streaming video multicast session. When the loss recovery latency threshold is 2000ms, the gain from the loss recovery is about from 6 to 7 dB in the test region of congestion; when the loss recovery latency threshold is 1000ms, the gain varies from 3 to 6 dB; when the loss recovery latency threshold is 500ms, the gain

[5]In reality, the video buffer size usually decides the loss recovery latency threshold: larger buffer size means higher threshold.

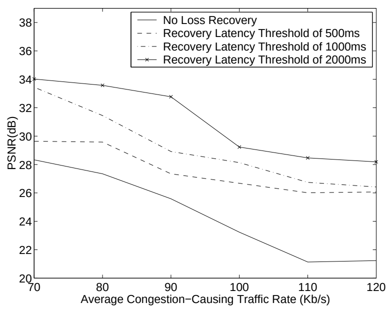

Fig. 10. PSNR of The Received Video

is from 1 to 5 dB. The simulations show that with a moderate buffer size (indicated by the loss recovery latency threshold), the AIR scheme can significantly increase the PSNR of the transmitted video. In addition, a larger buffer usually means a higher gain from the loss recovery (but with increased playback delay).

VII. Conclusion

High efficiency and scalability and low recovery latency are crucial for large-scale video multicast. To achieve these goals, the AIR scheme proposed in this paper approaches the multicast loss recovery problem from a new perspective: active injection of repair packets and on-demand construction and maintenance of minimum loss recovery structures. Instead of waiting passively for loss reports from receivers as in existing multicast loss recovery schemes, AIR actively recovers losses right at each loss site upon detecting loss events. With this new approach, AIR achieves good performance in all the three aspects: request suppression, local recovery and retransmission scoping, which are the three well-known characteristics required for a scalable and efficient multicast loss recovery scheme. In addition, AIR states are set up at the beginning of a congestion event and are deleted shortly after the disappearance of the congestion. Also, the temporary states only exist in the reference router of a loss event. This on-demand construction of minimum loss recovery structure can save considerable resources in a large-scale video multicast session. Another unique feature of the proposed scheme is its rate control over repair traffic. Simulations show that the number of losses on a bottleneck can be significantly reduced with this unique feature. All the above features of the proposed scheme contribute to the high efficiency and scalability of the proposed scheme. In addition, the AIR scheme has very low recovery latency. General and numerical analysis shows that the AIR scheme achieves a significantly better overall performance as compared to existing schemes. Our simulations show that the proposed scheme can considerably enhance the PSNR of the transmitted video in a streaming video multicast session.

References

[1] J. Nonnenmacher, E. Biersack, and D. Towsley, "Parity-based loss recovery for reliable multicast transmission," *IEEE/ACM Trans. on Networking*, vol. 6, pp. 349–361, Aug 1998.

[2] S. Pejhan, M. Schwartz, and D. Anastassiou, "Error control using retransmission schemes in multicast transport protocols for real-time media," *IEEE/ACM Trans. on Networking*, vol. 4, pp. 413–427, June 1996.

[3] P.A.Chou, A.E.Mohr, A. Wang, and S. Mehrotra, "Error control for receiver-driven layered multicast of audio and video," *IEEE Trans. on multimedia*, vol. 3, pp. 108–122, March 2001.

[4] W. Tan and A. Zakhor, "Video multicast using layered fec and scalable compression," *IEEE Trans. on Circuits and Systems for Video Technology*, pp. no. 3, Vol. 11, February 2001.

[5] I. Rhee, S. Joshi, M. Lee, S. Muthukrishnan, and V. Ozdemir, "Layered multicast recovery," in *Technical Report TR-9909*, NCSU, Computer Science Dept, February 1999.

[6] J. Byers, M. Luby, M. Mitzenmacher, , and A. Rege, "A digital fountain approach to reliable distribution of bulk data," in *Proc of ACM SIGCOMM.*, Vancouver, Canada, Sept 1998, pp. 56–67.

[7] S. Floyd, V. Jacobson, S. McCanne, C. G. Liu, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing," in *Proc of ACM SIGCOMM.*, October 1995.

[8] H. W. Holbrook, S. K. Singhal, and D. R. Cheriton, "Log-based receiver-reliable multicast for distributed interactive simulation," in *Proc of ACM SIGCOMM.*, August 1995, pp. 328–341.

[9] S. Paul, K. Sabnani, J. Lin, and S. Bhattacharyya, "Reliable multicast transport protocol (rmtp)," *IEEE J. on Select. Areas Commun.*, vol. 15, pp. 407–421, Apr 1997.

[10] R. Kermode, "Scoped hybrid automatic repeat request with forward error correction (sharqfec)," in *Proc of ACM SIGCOMM.*, Vancouver, Canada, Sept 1998, pp. 278–289.

[11] R. Yavatkar, J. Griffi oen, and M. Sudan., "A reliable dissemination protocol for interactive collaborative applications," in *Proc of ACM Multimedia*, 1996.

[12] X. Xu, A. Myers, H. Zhang, and R. Yavatkar, "Resilient multicast support for continuous-media application," in *Proc of IEEE NOSSDAV*, New York, May 1997, pp. 183–194.

[13] V. O. K. Li and Z. Zhang, "Internet multicast routing and transport control protocols," in *Proceedings of the IEEE*, March 2002, p. Vol 90 No. 3.

[14] T. Speakman, D. Farinacci, S. Lin, and A. Tweedly, "Pgm reliable transport protocol," *Internet Draft*, Aug 1998.

[15] S. Kasera, S. Bhattacharyya, M. Keaton, D. Kiwior, J. Kurose, D. Towsley, and S. Zabele, "Scalable fair reliable mulitcast using active services," *IEEE Network Magazine (Special Issue on multicast)*, vol. 14, pp. 48–57, January/February 2000.

[16] L. Lehman, S. Garland, and D. Tennenhouse, "Active reliable multicast," in *Proc of IEEE INFOCOM.*, San Francisco, California, March 1998.

[17] C. Papadopoulos, G. Parulkar, and G. Varghese, "An error control scheme for large-scale multicast application," in *Proc of IEEE INFOCOM.*, 1998, pp. 1118–1196.

[18] B. Levine and Garcia-Luna-Aceves, "Improving internet multicast with routing labels," in *Proc of ICNP*, Oct 1997, pp. 241–250.

[19] B. Cain, T. Speakman, and D. Towsley, "Generic router assist (gra) building block: motivation and architecture," *Technical report, IETF*, 2000.

[20] A. M. Costello and S. McCanne, "Search party: Using randomcast for reliable multicast with local recovery," in *Proc of IEEE INFOCOM.*, Mar. 1999, pp. 1256–1264.

[21] Y. Gao, Y. Ge, and J. C. Hou, "Rmcm: Reliable multicast for corebased multicast trees," in *IEEE Int. Conf. Network Protocols*, Nov. 2000, pp. 83–94.

[22] M. Calderon and et al., "Active network support for multicast applications," in *IEEE Network*, May 1998, pp. 46–52, Vol. 12.

[23] J. Nonnenmacher and et al, "How bad is reliable multicast without local recovery?" in *Proc of IEEE INFOCOM.*, March 1998.

[24] M. Yajnik, J. Kurose, and D. Towsley, "Packet loss correlation in the mbone multicast network," in *Proc of IEEE Global Internet Conf*, London, Nov 1996.