Receiver Grouping with Local Recovery for Scalable Reliable Multicast

Elias G. Khalaf
Department of Mathematics and Computer Science
Loyola University New Orleans
6363 St. Charles Ave., Campus Box 35
New Orleans, LA 70118

S. Sitharama Iyengar
Department of Computer Science
Louisiana State University
Baton Rouge, LA 70803

Abstract

Scalable Reliable Multicast Protocols that reduce processing requirements on receivers and senders have been the subject of much research in recent years. In [12] and [6] we propose a new protocol that groups receivers for error recovery into fixed-size groups, thus reducing their processing requirement to O(1). We expand on this work by applying the concept of Local Recovery to receiver groups, which further improves the processing requirements on the sender while maintaining a constant O(1) requirement on the receivers. Furthermore, we extrapolate the concept of receiver grouping to repair servers resulting in further reduction in the sender's processing requirements. The processing requirements on both the sender and the receiver for both cases are analytically studied.

Keywords: Reliable multicast, local recovery, receiver grouping, repair servers.

1. Introduction

Internet Multicasting refers to the delivery of data packets from a source (*sender*) to a set of destinations (*receivers*), rather than just a single destination [1]. Data packets can be sent from the sender to each receiver separately, but that would be wasteful of network bandwidth and of the sender's processing power. Instead, using multicasting, network routers perform replication of data when necessary to eventually reach all destinations. The sender in this case only sends out one copy of the data. The receivers, collectively, make up a *multicast group*.

The area of reliable multicast transport protocols has been a very active research area of the past few years. Many applications such as shared whiteboard, file replication and update, stock quote dissemination, web cache update, among others, require reliable multicast. Many researchers have been attempting to create one generic protocol that will be suitable for all applications. It has been realized, however, that there is no one protocol that serves this purpose. Applications must have greater control over the segmentation and framing of data units.

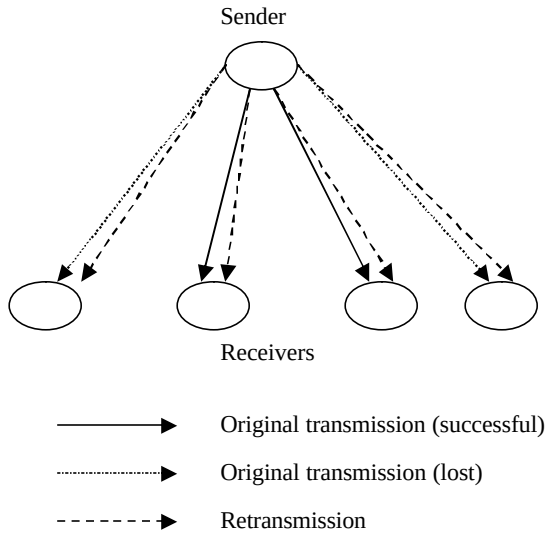

Fig. 1. Retransmission Scoping Problem

On the other hand, there has been a flood of new multicast protocols, each trying to serve a different multicasting need or trying to improve on an existing protocol. However, one major problem to overcome with all protocols is the *retransmission scoping problem* [5] as depicted in Fig. 1. This has to do with the unnecessary processing overhead that is imposed on a receiver that has already received a packet correctly. The problem in reliable multicast is how to scope retransmissions so as to shield receivers and the links leading to them from loss recovery due to other receivers [5].

In this paper we focus on processing requirements at the receivers and at the sender, with the conjecture that network bandwidth will keep outgrowing processing speeds for the next few years. Therefore, we build on our own work in [12] and combine it with the well-known approach of *Local Recovery* to achieve even greater reduction in processing requirements.

Local Recovery is a divide-and-conquer approach that divides receivers into *local regions*. Each local region will have a designated local *repair server* that handles repair and retransmission requests from the receivers in its own region. It requests repairs directly from the server only when it does not have a missing packet. Receiver grouping can now be applied independently in each region, with the view that the local repair server is the original server.

2. Related Work

Several reliable multicast protocols and architectures have been proposed. One of the leading protocols is (scalable reliable multicast) SRM [2], which is NACK-based. SRM suffers from the unwanted redundant packets sent to receivers due to the fact that retransmissions are multicast to the whole group, which is wasteful of bandwidth and receiver processing power. The concept of local recovery helps SRM reduce global retransmissions by isolating domains of loss, but the local groups can themselves suffer from the same global problem should the size of the multicast session grow too large. Other protocols like LBRM [4] and RMTP [8] (and others like [3]) use a hierarchical approach with *designated receivers* (DRs) to supply repairs to lower-level receivers. But the placement of such designated receivers and their processing requirements are not fully studied. The concept of a hierarchical acknowledgment (HACK) was introduced by [7] to refer to a consolidated ACK (positive acknowledgment) of a group of receivers (referred to as a *domain*) which will be periodically sent by the DRs to the next-level DRs in the hierarchy. HACKs help in moving the memory allocation window at the DRs. RMTP, LGMP [13] and TMTP [14] belong to this category.

The first analytical attempt in [9] and [10] to study the performance of reliable multicast protocols paved the way for researchers in this area. In [9], the performance of two fundamental classes of reliable multicast protocols were compared, namely, the *sender-initiated* and the *receiver-initiated* classes. Many other protocols were designed that belonged to neither category. In effect, two more categories were added by [7], namely the *ring-based* protocols and the *tree-based* protocols. The analysis and results can be found in [11] and will be skipped here for space limitations.

3. Protocols and System Model

3.1 System Model

This model consists of one sender *S* multicasting a continuous stream of packets to *R* receivers. The model could possibly be extended to have *R* receivers where any receiver has an equal chance of being a sender itself. For our model we assume:

- All loss events at all receivers for all transmissions are *mutually independent*;
- The probability of packet loss, *p*, is independent of the receiver;
- ACKs (positive acknowledgments) and NACKs (negative acknowledgments) are never lost and these packets (referred to as *control packets*) are typically small;
- Processing requirements at the hosts are more important than network bandwidth in determining the throughput of reliable multicast protocols.

3.2 Network Model

We assume the presence of a *repair tree* similar to the one in [10]. This repair tree is the physical

multicast tree constructed by the routing protocols with the sender as the root, receivers as leaves, and repair servers co-located with routers. In principle, repair servers could themselves be DRs.

3.3 The *K* Protocol

Our previous work on the *K* protocol [12] introduced the concept of *Receiver Grouping*, which is a process that randomly groups *n* receivers for the purpose of packet loss recovery. In the global mode of the protocol, *S* acts as a *Match Maker* and, upon request, groups *n* receivers with each other, *(n-1)* of which had been waiting to be grouped in a queue at *S*. While in the queue, these receivers will recover lost packets from *S*. An example of the operation of K is in Fig. 2.

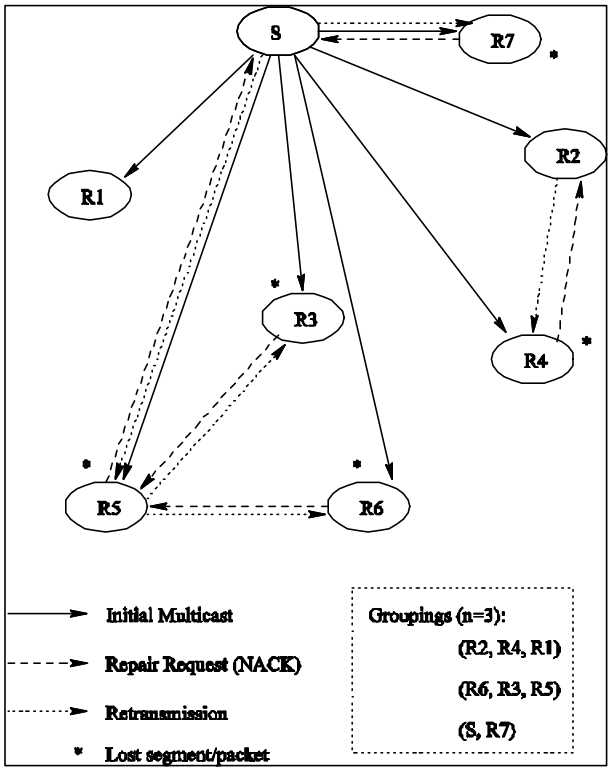

Fig. 2. Example of the operation of *K*

In this example, there is one sender *S* and seven receivers, *R1* through *R7*. The requests for grouping randomly arrive at *S* from *R2*, *R4*, *R1*, *R6*, *R3*, *R5* and *R7* respectively in that order. The resulting groupings are shown in the figure. Receivers *R3* through *R7* lose the packet/segment and request repairs from their buddies in the group. In group (*R2*, *R4*, *R1*), *R4* did not have the packet so it recovered it from *R2*. All receivers in the group *(R6, R3, R5)* lost the segment, so one of them (*R5* in this case) recovered the segment from *S* and supplied it to all others. Note that since *7 mod 3 = 1*, only one receiver (the last to request grouping), namely *R7*, did not find a group, so it paired with *S* for error recovery, awaiting a group.

The processing requirements for receivers under the K protocol has been reduced to a constant *O(1)*. The processing requirements for the sender under *K* is $O(1 + (p^n/n)R)$.

3.4 The new *KL* Protocol

The new protocol presented in this paper and initially described in [6], is a tree-based one that combines the concept of *local recovery* [3], [4], [8], with that of *grouping* to achieve even lower processing requirements on the sender, while maintaining a constant *O(1)* requirement on the receivers. The major difference in this protocol, which we will call *KL*, is that *repair servers* (special designated nodes in the network) now play the role of the sender in terms of grouping and error recovery. Only if a repair server does not have a lost packet does it consult the sender or another repair server in the hierarchy. The protocol exhibits the following behavior:

- Upon joining a multicast group (session), a receiver R_i initially pairs up with its local repair server RS_l until a *group G* of *n* receivers is formed;
- Receiver R_i informs RS_l that it is looking for a group;
- RS_l saves the address of R_i in a queue and waits until enough receivers request grouping;
- Upon receiving a request for grouping, RS_l checks if it has *(n-1)* receivers waiting for grouping. If so, then *S* informs R_i and all other *(n-1)* receivers in the queue that all of them are going to be buddies in the same group for the purpose of packet loss recovery. At that moment, R_i drops its pairing with RS_l and groups with its new buddies in *G*. If there are no receivers waiting for grouping, RS_l behaves as in the previous step;
- All receivers in *G* now use *point-to-point* (peer-to-peer) communication for error correction in a NACK-based fashion (as

opposed to multicasting their NACKs). When group members detect lost packets, they use one of several possible *group recovery schemes* described in [6]. The schemes work in such a way that all *n* receivers will end up having the missing packet. If at least one member has it, then RS_l should not be bothered and all members should recover their loss from that one receiver. Only if none of the receivers in *G* has a missing packet does RS_l get consulted;

- Upon leaving a multicast session, a receiver informs its group members that it intends to leave. The group members then break the group and act as if they have just joined the session, i.e. they pair with RS_l until enough members are found to form a new group;
- If some receiver in a group is not responding, then its buddies in the group can detect this either by polling it periodically, or by expiring timeouts, in which case they behave as in the previous step;
- If a receiver R_i has been dropped (dumped) by its group members, then upon reestablishing a contact with any of the former members, R_i is informed that it needs to find another group. R_i then behaves as if it's joining anew.

3.5 Observations on the *KL* Protocol

We observe the following properties for the *KL* protocol:

- RS_l acts as a *match maker* in forming *G*;
- Protocol *KL* is *stateless* with respect to grouping, i.e. it keeps no record of which receivers are grouped with each other at any point in time. Statelessness is very important to have; otherwise, it would be very costly in terms of memory requirements for a server to remember all groups. In addition, that information will become stale very quickly due to the dynamic nature of receivers joining and leaving a multicast session;
- Unless there are delays in processing groupings, for example due to insufficient processing power, no more than *(n-1)* receivers should pair with RS_l at any point in time, since they will end up being grouped with each other;
- If *R*, the total number of receivers, is a multiple of *n*, then *R/n* groups will be formed with no leftover receivers. Otherwise we will have at most *(R mod n)* receivers paired with RS_l at any point in time.

It is worthy to note that all receivers are required to group under *K* and it is assumed that none of them are incapable of doing so.

3.6 Example

The operation of the *KL* protocol is depicted in the example in Fig. 3. In this example, there is one sender *S*, two repair servers *RS1* and *RS2*, and seven receivers *R1* through *R7*. The fixed receiver group size is *n*=2. Requests for grouping arrive at *RS2* from *R2*, *R4* and *R7*, respectively in that order. The resulting four groupings are shown in the figure.

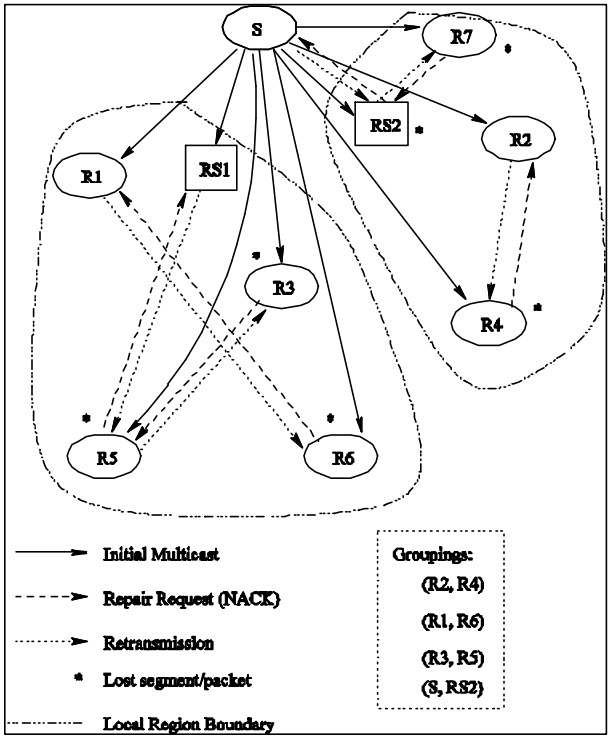

Fig. 3. Example of the operation of *KL*

Note that no requests for grouping arrived at *S*. Also, note that for the local region of *RS1*, the number of receivers is even and they were all grouped. In the case of *RS2*, *R7* was left out, so it remained paired with *RS2* for error correction.

When *R3* requested repair from its buddy *R5*, *R5* did not have the packet either, so it requested it from *RS1*. In the case of *R7*, whose buddy was *RS2* itself, *R2* did not have the packet itself, so it requested it from *S*.

3.7 Effect of Local Grouping on Servers

The analysis of the effect of local grouping on repair servers is similar to that of the effect on the sender *S* presented in [12]. However, one major improvement here is that the number of local receivers in a local group is much smaller than *R*. Therefore, the effect on repair servers will be even more insignificant than that previously experienced by *S*. There is no effect on *S* from local grouping, since *S* is not involved anymore in the grouping operations; repair servers, rather, act as the match-makers. This is a minor improvement on the processing requirements of *S* under *KL*.

4. Processing Cost Analysis

In order to understand the improvements achieved with the *KL* protocol, we now analyze the processing requirements on receivers as well as on the sender. We follow the same analysis and use the same notation presented in [12], adapted from [10] and [9]. The notation used in the processing cost analysis is described in Table 1, which is an extension of a similar table presented in [12] that includes notation specific to local recovery.

4.1 The Receiver

The expected processing requirement at the receiver will not change under *KL* since the group size and the recover schemes remain the same as in *K*. In addition, as far as the receiver is concerned, the local repair server under *KL* is indistinguishable from *S*, which acted as a repair server under *K*. Therefore, $E[Y^{KL}] \in O(1)$. This means that the processing requirement at the receiver under *KL* is *constant* and is *independent of R*, which is a highly desirable property that has been preserved for both *K* and *KL*.

4.2 The Sender

Processing requirements on the sender will definitely drop down due to the fact that the repair servers in the network will now handle repair requests from receivers in their local regions. Repair servers only consult with *S* when it's absolutely necessary.

We are interested in the mean processing requirement per packet in order for the sender to multicast packets reliably to all of the receivers. Processing requirements at the sender can be expressed as the total time needed to perform the following:

- prepare and transmit the first packet (including feeding the packet from the application to transport layer)
- process all NACKs sent by repair servers on behalf of their local regions
- process all retransmissions to repair servers in response to their NACKs.

Table 1. Notation

X_f	Sender time to feed packet from application to transport layer
X_p	Sender time to process transmission of a packet
X_n	Sender time to process a NACK
p	loss probability at a receiver
M	Number of repair servers requesting repair from sender
R	Total number of receivers in session
n	Number of receivers is a group
X^w, Y^w, P^w	Sender, receiver & repair server's packet processing time in protocol w.
B	The branching factor of a multicast tree or the number of repair servers

Therefore, the expected processing requirements at the sender under the *KL* protocol can be expressed as $X^{KL} = X_f + M(X_n + X_p)$, where M is the number of repair servers requesting repair from *S*.

Taking the expectation of X^{KL} we get

$$E[X^{KL}] = E[X_f] + E[M](E[X_p] + E[X_n]).$$

It is worthy here to note that error correction from the sender is done on a one-to-one unicast communication channel. Therefore, we are interested in finding the number of repair servers that will request correction from *S*. For any given lost packet, a repair server will make *only one* repair request to *S*, even if more than one local group did not receive the packet. Therefore, the probability of a repair server requesting a lost packet from S is the same as that of any given group (p^n), with the added advantage that the repair server, being a receiver itself, may have the lost packet. As a result, the probability is reduced to p^{n+1}.

Assuming the existence of B ($<< R/n$) repair servers, the expected value of M can now be given as $E[M] = (p^{n+1})B$. Substituting in $E[X^{KL}]$ we get

$$E[X^{KL}] = E[X_f] + (p^{n+1})B(E[X_p] + E[X_n]).$$

Please note that the n in X_n is for NACKs and is not the same as the size of a group. Therefore,

$$E[X^{KL}] \in O(1 + (p^{n+1})B).$$

As $p \to 0$, $E[X^{KL}] = O(1)$, which is a desirable property. This result is an improvement of the processing requirements of *S* under *K*, which was $O(1 + (p^n/n)R)$.

5. Grouping of Repair Servers

We can further improve the processing requirements on *S* (not affecting that of the receivers) if we apply the concept of receiver grouping to the repair servers themselves, treating them as if they were receivers. The repair servers will now form groups for the purpose of error correction. If one or more groups local to a repair server request a missing packet, and if the repair server itself does not have it, it can ask its buddy repair servers, and only if they don't have it will they consult *S*. Assume that repair servers will form groups of size n as do the local groups, with each repair server's probability of requesting repair from *S* being p^{n+1}. The probability of a repair servers group not having a particular packet is now reduced to $(p^{n+1})^n$. With B/n repair servers groups, the expected value of M can now be computed as $E[M] = (p^{n+1})^n B/n$. Substituting in $E[X^{KL}]$ and taking the expectations we get

$$E[X^{KL}] \in O(1 + (p^{n+1})^n B/n).$$

This further improves the processing requirements on the sender *S*.

6. Conclusions and Future Directions

In this paper we introduced the *KL* protocol, which applied the concept of local recovery to our earlier results of the *K* protocol. The result is that the processing requirements on the sender were reduced, while maintaining the constant processing requirements on the receivers. Further, we extrapolated the concept of receiver grouping to repair servers resulting in further reduction in the sender's processing requirements. The processing requirements on both the sender and the receiver for both cases were analytically studied.

Many unresolved issues remain. First, the processing requirements on the repair servers must be studied. In trying to reduce the processing requirements on the sender *S*, more workload is now shifted to the core of the network, namely to repair servers. The effect of such a shift on network bandwidth must be analyzed. Second, the use of repair servers, and subsequently their grouping may exacerbate the end-to-end latency when it comes to lost packet recovery. Finally, the designation and location of repair servers, the size of a local group and the binding of receivers to repair servers must be explored.

7. References

[1] S. Deering, "Multicast Routing in a Datagram Internetwork," Ph.D. Dissertation, Stanford University, Palo Alto, CA, December 1991.

[2] S. Floyd, V. Jacobson, S. McCanne, C. Liu, and L. Zhang, "A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing," *IEEE/ACM Trans. Networking*, Vol. 5, pp. 784-803, December 1997.

[3] M. Hofmann, "Enabling Group Communication in Global Networks," *Proceedings of Global Networking '97*, Calgary, Alberta, Canada, November 1996.

[4] H. W. Holbrook, S. K. Singhal and D. R. Cheriton, "Log-Based Receiver Reliable Multicast for Distributed Interactive Simulation," *Proceedings of ACM SIGCOMM*, pp. 328-341, August 1995.

[5] S. Kasera, G. Hjalmtysson, D. Towsley and J. Kurose, "Scalable Reliable Multicast Using Multiple Multicast Channels," *IEEE/ACM Trans. Networking*, Vol. 8, No. 3, pp. 294-310, June 2000.

[6] Elias G. Khalaf, "Congestion Control Mechanisms for Internet Multicast Transport Protocols", Ph.D. Dissertation, Louisiana State University, Baton Rouge, Louisiana, May 2000.

[7] B. N. Levine and J. Garcia-Luna-Aceves, "A Comparison of Known Classes of Reliable Multicast Protocols," *Proceedings of ACM Multimedia*, November 1996.

[8] J. C. Lin and S. Paul, "RMTP: A Reliable Multicast Transport Protocol," *Proceedings of IEEE Infocom*, pp. 1414-1424, 1996.

[9] S. Pingali, J. Kurose and D. Towsley, "A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols", *Proceedings of ACM Sigmetrics Conference*, May 1994.

[10] D. Towsley, J. Kurose and S. Pingali, "A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols", *IEEE Journal on Selected Areas in Communications, Vol. 15, No. 3, pp. 398-406*, April 1997.

[11] S. Paul, *Multicasting on the Internet and its Applications*, Kluwer Academic Publishers, 1998.

[12] E. Khalaf and S. S. Iyengar, "Scalable Reliable Multicast Using Receiver Grouping", *Proceedings of the International Conference on Internet Computing*, Las Vegas, 2004.

[13] M. Hofmann, "A generic concept for large-scale multicast," *Proceedings of the International Zurich Seminar on Digital Communications*, (IZS 96), February 1996.

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