Appeared in the proceedings of the Tenth Annual IEEE
Workshop on Computer Communications, Sept. 1995

IMPLOSION CONTROL FOR MULTIPOINT APPLICATIONS[1]

Christos Papadopoulos
christos@dworkin.wustl.edu

Guru Parulkar
guru@flora.wustl.edu

Department of Computer Science
Washington University, St. Louis Missouri

ABSTRACT:

One of the new challenges posed by multipoint applications in future networks is scalable multicast error control. Current error control mechanisms do not scale because they do not deal with message implosion, caused by synchronized feedback messages (e.g. positive or negative acknowledgments) from the receivers in a multicast connection.

Implosion control can be performed by the receivers or the sender. The former, where receivers collaborate to control implosion has been the most popular so far. In the latter, the sender is responsible for directly controlling feedback from the receivers. Sender-controlled implosion is appropriate for cases where receivers are isolated from each other (e.g. where privacy or anonymity is desired, where receivers have limited processing capabilities and in one-to-many connections in connection-oriented networks). Implosion can also be controlled by structuring receivers in a virtual tree hierarchy with the sender at the root, where feedback messages are restricted from children to their parents. Such a structure is also suitable for ACK consolidation which is required for implosion control in reliable multicast.

In this paper we present the design and propose to investigate via analysis, simulation and implementation two implosion control mechanisms: a sender-controlled and a dynamic hierarchical implosion control mechanism. Our hierarchical mechanism uses dynamic structures called multicast recovery trees (MRTs) formed by the receivers.

1. INTRODUCTION

Emerging high-speed networks have stirred excitement with their potential to support multimedia, continuous media and multipoint applications like conferencing, collaborative work, video-on-demand, information browsing, multimedia magazines, voting systems, etc. In addition to high bandwidth, future networks will provide quality of service guarantees and efficient multicast to support multipoint applications.

In this paper, we examine some ways error control may evolve in order to be applied efficiently to multipoint applications with a large number of participants. Existing point-to-point error control mechanisms

1. Presented at the Tenth Annual IEEE Workshop on Computer Communications, Rosario Resort, Eastsound WA, Sept 17 - 20, 1995

are unsuitable for these applications because they do not scale due to *implosion*. Implosion occurs when a large number of endpoints in a multipoint connection send messages simultaneously. For example, several receivers may generate acknowledgments, or on loss several receivers may send retransmission requests at the same time. The impact of implosion depends on the underlying communication: in bidirectional one-to-many communication only the sender experiences implosion, whereas in many-to-many communication, all participants are affected. Implosion has detrimental effects on communication, including a large increase in message processing, latency, loss of messages from buffer overflow and waste of network bandwidth. Therefore, it is clear that in order to be scalable multipoint error control must be augmented with implosion control.

We have designed, and plan to analyze, simulate and implement scalable multipoint error control mechanisms, augmented by a set of implosion control mechanisms. Since in the future the communication environment is expected to be very diverse, we plan to take into account factors like the nature of the underlying network (connectionless or connection-oriented), the type of communication required (one-to-many or many-to-many), the application requirements (privacy, anonymity, etc.), the power of the end machines (workstations to hand-held mobile units), as well as the desired performance/complexity application trade-offs.

This paper is organized as follows: Section 2 presents a brief summary of related work. Section 3 defines the properties of an optimal solution to implosion control and divides solutions into three approaches, namely sender-controlled, receiver controlled and hierarchical implosion control. Sections 4 and 5 present our solutions and trade-offs based on sender controlled and hierarchical implosion control. Finally, Section 6 concludes the paper.

2. RELATED WORK

A mechanism for implosion control was first introduced in XTP in the form of *damping* and *slotting* heuristics [10]. Recently, the SRM [6] mechanism was presented, which is an enhancement to the original XTP mechanism, targeted at the Internet environment. In SRM the receivers coordinate to reduce implosion by keeping a request and a repair timer, which introduce a back-off interval to receivers' messages. The timer intervals for each receiver are a function of RTD, which allows receivers closer to the packet loss to send request and repair messages first, thus preventing other receivers from sending more messages. The timer intervals also contain a random element to de-synchronize receivers whose RTD is similar. The mechanism has performed very well in simulation.

Other solutions have been proposed based on a hierarchical approach [7,8,11]. In [7] an error recovery mechanisms was proposed for Distributed Interactive Simulation (DIS). Implosion in DIS is avoided by directing all messages from receivers to geographically fixed servers and sending retransmissions either point-to-point, or via broadcast if data was lost by a large number of receivers. The solution in [8] proposes a fixed hierarchy with network-provided designated receivers (DRs) to provide reliable multicast. In [11] a dynamic hierarchy is proposed where children locate their parents (called Domain Managers) via an expanding ring search using the TTL field in IP. The DMs are created and terminated dynamically, as the multicast tree expands and contracts.

3. CONTROLLING IMPLOSION IN MULTICAST ERROR CONTROL

In this section we begin by defining the properties of what we view as optimal implosion control and argue that implementing such a solution is very hard. Thus we proceed to define a taxonomy of solutions that do not meet our optimality criteria, but easier to implement. We will present two solutions in later sections, one of which comes very close to meeting our optimality criteria.

3.1 Optimal solution to implosion

In our optimal solution to implosion, the sender is under the illusion that it communicates with a single virtual receiver, which aggregates the messages from all real receivers. Therefore, a multicast session appears to the sender as a point-to-point session both in terms of control messages and latency, regardless of the number of receivers. That is, the sender sends exactly the same number of messages and recovery takes at most the same time as in a point-to-point connection. In our definition, an optimal solution possesses certain properties in terms of messages and latency, which we define next:

Messages:

1. *Isolation*: recovery-related messages reach only endpoints that experienced loss and the retransmitting entity
2. *Redundancy*: only one retransmission request is generated from each connection subtree and only one retransmission reaches each endpoint that experienced loss

Latency:

3. *Recovery in one RTD*: a retransmission request is sent immediately by the receiver that first discovers loss and a retransmission is sent immediately by the endpoint closest to the requesting receiver

Additional conditions:

4. *Optional participation*: receiver participation in recovery is optional and recovery should be completely transparent to receivers that do not participate.
5. *Minimal message exchange*: if messages are exchanged between receivers, the number of messages is minimal and does not contribute to implosion

A significant amount of work has been done in optimizing the design and implementation of point-to-point transport protocols [1, 5]. The instruction count for the per-packet processing has been reduced to a few tens of instructions, and optimizations in memory management are resulting in only one or zero data copies of transport protocol packets [2, 4]. It is important that the efficiency gained in optimizing point-to-point protocols finds its way to multipoint protocols and in addition, the efficiency is not compromised by the implosion control mechanism. However, designing and implementing optimal implosion control is hard. State is distributed and receivers are unaware of the state or messages sent by other receivers. Clearly, receivers cannot coordinate by exchanging explicit messages, but must resort to implicit mechanisms whose parameters are hard to estimate. For example, receivers may back-off from sending messages to detect and suppress duplicates. However, estimating the right back-off delay while minimizing latency is hard, especially for receivers scattered over large geographical distances. To summarize then, the implosion problem is challenging because it requires implicit mechanisms that can accurately predict the correct actions for each receiver and minimize latency and the exchange of explicit messages.

Despite its complexity, defining an optimal implosion control solution is useful for evaluating other less complex solutions. Therefore, we use the definition of the optimal solution as a metric for measuring the effectiveness of other solutions.

3.2 Network participation

Before designing implosion control mechanisms, a key question has to be answered: can the designers assume support from the network switching entities (switches, gateways, routers, etc.) for implosion control? Such support seems attractive because it allows identification and discarding of duplicates at the switching entities. However, in order to participate in implosion control the network entities must examine individual packets. This has several drawbacks:

1. Packet headers must be reassembled before they can be examined. This is very difficult at switches that switch cells (ATM).
2. Additional per-connection processing and state is required at the switching entities. These new resources must be shared and managed. In addition, the new resources will also complicate signalling protocols.
3. The network entity examining the packets must interpret the headers of the protocol, meaning that part of the protocol must reside on that particular entity. This makes software updates difficult.
4. Switches must be programmable to perform application-specified implosion control services

For the reasons listed above we believe that network participation is too complex for the current generation of switches. We therefore make the key *initial* assumption that switching entities do not participate in implosion control. As protocols and networks evolve, the possibility of network participation in implosion control must be explored. We believe that a future generation of networks will have programmable switching entities, and applications will be able to program them. For now, however, we assume that implosion control is performed by the transport protocol.

3.3 Implosion control taxonomy

We view a multipoint connection as having two components: (1) a primary multicast channel that the sender uses to multicast data to the receivers, and (2) a secondary channel that receivers use to send control messages to the sender or other receivers[1]. We have defined a taxonomy of implosion control mechanisms based on the type of secondary channel used. The taxonomy divides the mechanisms in three categories. (1) Sender-controlled implosion, (2) Receiver-controlled implosion, and (3) Hierarchical implosion control. We introduce these next.

Sender-controlled implosion:

In sender-controlled implosion all implosion related decisions are taken by the sender. The secondary channel allows requests from receivers to go back to the sender, but not necessarily to other receivers. Typically, the sender controls implosion by: (1) providing each receiver with a back-off delay, and (2) notifying the receivers about which requests the sender has received, so that receivers can cancel their requests. Sender controlled implosion is suitable in the following cases: (1) when the sender must have absolute control of the receivers, (2) when inter-receiver communication is not possible due to the nature of the connection (e.g. one-to-many connections in connection-oriented networks), and (3) when inter-receiver communication is not allowed for security or privacy and anonymity reasons (e.g. in voting systems).

Receiver-controlled implosion

Receiver-controlled implosion is possible when the secondary channel allows messages sent by any receiver to reach all other receivers, so that receivers can collaborate to control implosion. Since messages reach everyone, receivers must be very careful about sending messages to avoid duplicating other receivers' messages. The sender may or may not be involved in the process (for example it may notify the receivers about whether it is experiencing implosion). Receiver-controlled implosion is appropriate for broadcast channels or many-to-many communication (e.g. IP Multicast, ATM many-to-many connections, etc.) and usually requires that receivers be able to buffer and retransmit lost data. The XTP and SRM heuristics (see "2.: RELATED WORK" on page 2) are examples of receiver-controlled implosion.

Note that the cost of receiver-controlled implosion may be different in connection-oriented and con-

1. The primary and secondary channels may sometimes be the same, as for example in many-to-many connections, where the sender and receivers use a group address for data and control.

nectionless networks. For example, the cost of setting up and maintaining many-to-many connections is drastically different in ATM and the Internet: many-to-many connections are readily supported on the Internet with IP Multicast [3], but are currently very expensive in ATM requiring a one-to-many connection from each endpoint.

In both sender and receiver-controlled implosion, we assume that retransmissions are sent over the primary channel and may reach some receivers that do not need them. Also we assume that in sender-controlled implosion the number of messages must be reduced below the maximum the sender can tolerate, whereas in receiver-controlled implosion the number of messages must be reduced below the maximum the sender or any receiver can tolerate.

Hierarchical implosion control

In the hierarchical solution, the receivers are organized in a tree hierarchy with the sender at the root. The secondary channel allows children to send messages only to their parents. Parents either combine messages from children into a single message and forward it up the tree (e.g. acknowledgments), or discard redundant messages (e.g. retransmission requests). Messages converge as they propagate up the tree and the sender finally receives a single message from each of its immediate children.

In the hierarchical approach, retransmissions can be sent by the sender to everybody over the primary channel, or by a parent to its children. The latter allows local recovery and avoids propagation of retransmission requests and retransmissions to the whole group. Parents may use the secondary channel for retransmissions if it is bidirectional, or may insert retransmissions into the primary channel.

In the following two sections we present our solutions for sender-controlled and hierarchical implosion control. In each section we present an outline of the implosion control mechanism and point out its trade-offs. In addition, in the hierarchical solution we examine message consolidation.

4. SENDER-CONTROLLED IMPLOSION

In this section we begin by describing a basic mechanism for sender-controlled implosion. Following the description, we discuss trade-offs and propose a set of enhancements to the basic mechanism. Finally, we present our evaluation plan.

4.1 Basic mechanism.

The key observation in sender-controlled implosion is that a retransmission request sent by a receiver remains invisible to the other receivers until the sender sends a retransmission. Therefore, to avoid implosion, each receiver has no choice but to assume that a request was sent by another receiver and delay sending a new request until it discovers otherwise. An extreme solution would be to arrange receivers in a linear chain, where each receiver waits for a possible retransmission initiated by the previous receiver before acting. Such a linear chain ensures that at most one request reaches the sender at the expense, however, of introducing very high latency (when the receiver set is large). However, typically *some* redundancy can be tolerated at the sender; in addition, not *all* receivers send requests at *all* times. Thus, rather than creating a linear chain, receivers can be bundled into groups where the whole group is allowed to send requests when they experience loss. Such grouping makes the chain shorter and fatter, reducing latency at the expense of generating some redundant requests, which are on the order of the maximum group size. For this reason it is critical that the sender is allowed to control the size of each group to ensure that the size stays below the sender's implosion threshold.

We now present a basic mechanism based on the above observations. In this mechanism the receivers are grouped into a chain of "buckets" depending on their *RTD* from the sender, as shown in Figure 1.

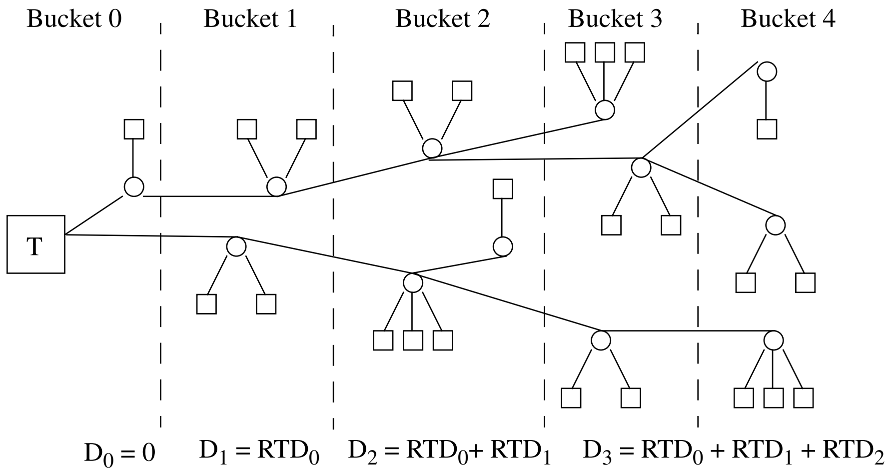

Figure 1: Grouping endpoints into buckets

Grouping receivers based on *RTD* reduces latency by preventing receivers with small *RTD* from waiting for receivers with larger *RTD*. Each bucket is associated with a value RTD_b, which is the maximum *RTD* of any receiver in the bucket and a back-off time D_b. Before sending a request, receivers in bucket b wait for a retransmission assuming that bucket *b-1* has initiated a request, and send their own requests only after a retransmission fails to arrive. D_b calculated using the following expression:

$$D_b = \sum_{k \bullet 1}^{b} RTD_{k-1}$$

We sketch the operation of the mechanism below. For brevity, we have omitted details that are not essential for understanding its operation:

Sender actions:

1. The sender maintains values for the number of buckets, the back-off timer for each bucket and a receiver histogram according to their *RTD*. When a receiver joins the connection, the sender places the receiver in the appropriate bucket based on its *RTD* and relays to the receiver the bucket *ID*, the value of the back-off timer and the value of the maximum back-off timer of any bucket (last bucket).
2. When the sender receives a retransmission request, it sends a retransmission immediately, recording the retransmission count in the request (see below). The sender ignores requests for the same data with the same or lower retransmission count for a period equal to RTD_{max}.
3. The sender keeps its retransmission buffers for an interval equal or greater than the timer value of the last bucket plus the last bucket's *RTD*. Each time a retransmission request arrives, the buffer lifetime is increased.

Receiver actions:

1. When a receiver detects a loss, it sets its back-off timer to the supplied value and waits. If a retransmission is received before the back-off timer expires, the request is cancelled.
2. If the back-off timer expires and no retransmission is received, the receiver sends a request with a retransmission *count = 1* and sets its back-off timer to the maximum back-off time (last bucket).

3. If a retransmission fails to arrive again, the retransmission count is incremented, another request is sent and the timer value is doubled. This continues for *N* iterations, after which sender or network failure is assumed.

Bucket splitting/joining:

The sender must control the size of each bucket to ensure that the bucket does not become too large. The sender splits the bucket using the following procedure:

1. The sender decides on the new values of the request delay timers for the two new buckets by dividing the receivers in two equal groups using the receiver histogram.
2. The sender multicasts the *ID* of the bucket to be split, the split threshold, the new bucket *IDs* and the new back-off values.
3. Receivers belonging to the split bucket update their state and join a new bucket. Receivers in downstream buckets add the new bucket's back-off delay to their back-off delay.

Similarly, the sender can collapse two buckets into one if it discovers that membership has dropped, by multicasting a message containing the *IDs* of the buckets to be collapsed and the new request delay timer. Collapsing buckets is highly desirable since it reduces latency and the buffering at the sender.

4.2 Limitations and trade-offs

The lack of receiver communication allows the mechanism to meet part of the isolation and part of the redundancy optimality criteria (see "3.: CONTROLLING IMPLOSION IN MULTICAST ERROR CONTROL" on page 3). The mechanism suffers from high latency, which increases in a cumulative fashion: the time-out of the last bucket is the sum of the time-outs of all previous buckets. Therefore, the mechanism penalizes receivers with high *RTD*. Additionally, the introduced latency may require large retransmission buffers at the sender. The mechanism in its basic form does not scale well and the maximum number of receivers is bounded by the application latency requirements and the sender's implosion threshold. On the positive side, since the sender has full control of the receivers, the probability of implosion is minimized. Therefore, the mechanism allows the sender to optimize performance by selecting a trade-off between the maximum bucket size and latency. In addition, the mechanism requires minimal implosion-related processing at the receivers thus keeping receiver complexity low.

4.3 Enhancements/optimizations

The mechanism's greatest limitation is latency, especially for receivers with high *RTD*. Next, we examine methods to reduce latency.

Latency is reduced by decreasing the total number of buckets. This can be done by increasing the size of the buckets beyond the sender's threshold, while ensuring that for every bucket the probability that more receivers than the sender's implosion threshold send requests simultaneously, is acceptably low. In order to increase the bucket size we assume that in most cases losses in the network are localized. In other words, given that a packet was lost in a switch, the probability that a subsequent packet will be lost at the same switch is high. This scenario is typical during congestion, which is expected to be the main cause of packet loss in future networks. This assumption is necessary to allow us to predict that in most cases the same set of receivers will detect the loss of a packet. For example, repeated losses at a congested switch trigger requests from the same set of downstream receivers. With this assumption, the sender can reduce the probability of implosion by placing receivers that send requests simultaneously in different buckets[1].

1. If losses in the network are random, this method may not work because there exists a probability that the number of receivers in a bucket that fire simultaneously may exceed the sender's threshold.

The preceding idea can be generalized by allowing all receivers to be distributed in arbitrary buckets to achieve more flexibility. To achieve arbitrary distribution each receiver is assigned *RTD* equal to RTD_{max}. This is necessary to allow any receiver to be placed into any bucket. The bucket back-off values become multiples of RTD_{max}, for example for RTD_{max} = 5 mSec, the bucket values are 0, 5, 10, 15, etc. With arbitrary receiver distribution latency is reduced by allowing larger bucket sizes, thus reducing the number of buckets. We will examine strategies for receiver placement as part of our research.

With arbitrary receiver distribution, the average latency for recovery can be made uniform for all receivers by allowing the sender to rotate the buckets periodically. For example after a bucket rotation, bucket *0* may become bucket *1*, bucket *1* may become bucket *2*, and bucket *n* may become bucket *0*. This can be accomplished by the sender periodically multicasting a bucket rotation message. Bucket rotation, however, does not reduce the maximum latency for recovery.

If receivers are distributed over large geographic distances, it may be more efficient to permanently assign receivers to buckets based on *RTD*. To avoid penalizing receivers with high *RTD*, the sender may supply a probability *p* to select receivers, which allows them to send requests immediately rather than wait for their back-off timer to expire. The sender may give higher probabilities to receivers that experience loss frequently, or may distribute the probabilities to receivers at strategic locations.

5. HIERARCHICAL IMPLOSION CONTROL

In this section we describe a hierarchical implosion control mechanism based on structures we call *Multicast Recovery Trees* (MRTs). MRTs constitute the *secondary channel* (see "3.3: Implosion control taxonomy" on page 4) in multicast connections. For better contrast, in this section we will refer to the *primary channel* as the *Multicast Data Tree* (MDT). MRTs are created by the receivers and are composed of interconnected nodes, each containing a group of receivers. Recovery in MRTs is initiated by the receivers [9]. The MRT node interconnection and communication are managed by a set of MRT components and implosion is controlled by a distributed mechanism. Both the MRT node management and the implosion mechanism are discussed later. MRTs are highly scalable, offer low latency in recovery and can be used on any type of network (connection-oriented or connectionless) to provide near-optimal implosion control.

We begin by describing how receivers are structured to form MRTs. Then we describe the components required to manage MRTs, followed by a description of the implosion control mechanism running on top of MRTs. Finally we discuss the importance of consolidation and how it can be directly supported on MRTs[1]. We conclude by discussing the performance and trade-offs of MRTs.

5.1 MRT structure

MRTs are constructed by the receivers by forming a tree hierarchy with the sender at the root. Each node of the tree is formed by a group of receivers. Each receiver group contains a designated receiver, called a Relay Node (RN). A RN communicates with all the receivers in the group via a separate private channel, and with the node's parent and children (if any), via point-to-point bidirectional connections. Receiver participation is optional and therefore the MRT topology may be different than the MDT topology. The sender may or may not be aware of the existence of MRTs. An example of a MDT and a corresponding MRT is shown in Figure 2. It is important to note that the receivers in a particular MDT are free to create more than one MRT if they need to, with different loss guarantees. For example, some receivers may be connected to a reliable MRT, while others to a low-latency best-effort MRT.

1. Note that sender or receiver-controlled implosion mechanisms cannot directly support consolidation.

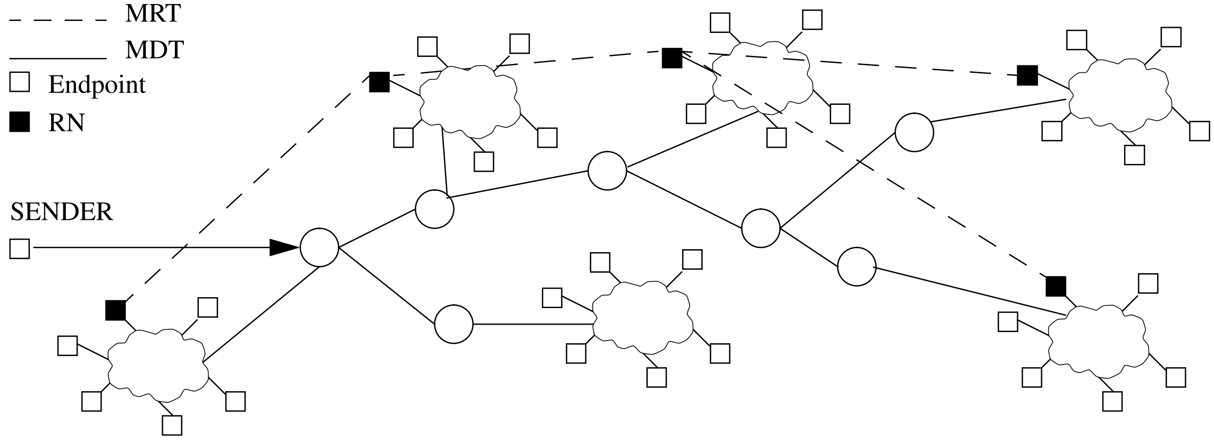

Figure 2: Multicast and recovery trees

5.2 MRT components

MRTs are composed of four components. These are:

1. *MRT management* component: this component deals with the interconnection and membership operations to structure RNs into MRTs. It supplies each RN with information about its parent and children. This component also deals with failures.
2. *MRT communication* component: this component is responsible for communication between RNs. Typically, this is a point-to-point connection.
3. *MRT group management* component: this component deals with the formation, membership and maintenance of groups. It is responsible for notifying new receivers of the RN identity and if required, dealing with failures.
4. *MRT inter-group communication* component: this component is responsible for communication between RN and its group. Typically this is a private (to the group) multicast connection. This component is also responsible for detecting RN failure, if required.

5.3 Implosion control actions

The implosion control actions are described next. We present actions for two cases: (1) when loss occurs outside a group, and (2) when loss occurs within a group. For brevity, we omit details related to fault tolerance.

Loss outside a group (all group members lost the packet)

1. On detecting a loss, all members in a node except RN, start a back-off timer with value greater than the largest propagation delay between any receiver in the group and RN.
2. On detecting a loss, the RN immediately sends a retransmission request to its parent RN and also multicasts the request to everyone in its group.
3. If receivers in a group receive a retransmission request from RN before their back-off timers expire, the receivers cancel their requests.
4. If the parent RN has the requested data, it relays the data to the requesting RN over the point-to-point connection. If the parent RN does not have the data, it has already sent a request to its parent. The parent RN marks that requesting child as “expecting” data.

5. When a retransmission is received from the parent, the RN relays it to the node members and to all “expecting” children.

Loss inside a group (some group members lost the packet)

1. If only some receivers but not the RN have experienced loss, their back-off timer described earlier in step 1 will expire and one or more requests will be sent to RN. RN must have the missing data (since it did not execute step 2) and retransmits the missing data to the receivers.
2. If some receivers including the RN have experienced loss, there are two options: the RN sends a request as before, which results in a transmission from receivers that have the data to the RN, or receivers stay quiet and data is recovered from the parent RN as before. In both cases the RN relays the data to the receivers, some of which will receive redundant data.

5.4 Consolidation

The actions described so far are effective in reducing redundant messages. Some types of messages, however, are not redundant and thus cannot be discarded. These messages are even more likely to cause implosion than redundant messages. Examples of messages that cannot be discarded are positive acknowledgments in a reliable protocol. An efficient method to avoid implosion from these messages is consolidation. In consolidation messages from multiple receivers are combined into a single message at intermediate nodes in the multicast tree, and the resulting message is forwarded up the tree where consolidation takes place again, until a single message is delivered to the sender. The structure of MRTs makes them highly suitable for consolidation by using the RNs as consolidating nodes.

5.5 Performance and trade-offs

The hierarchical solution to implosion adopted by MRTs offers near-optimal implosion control, low latency and excellent scalability. The difficulty in implementing the hierarchical solution lies in carefully specifying, implementing and integrating with the network the four components identified earlier in 5.2.

We characterize the performance of MRTs as near-optimal because performance comes very close to the optimal solution we defined earlier (see “3.1: Optimal solution to implosion” on page 3). MRTs totally eliminate implosion for losses outside the groups. MRTs deviate from the optimal solution in the following cases: (1) a retransmission originating near the root may need to traverse several hops to reach all the groups[1], (2) if loss occurs outside a group, a message from RN to the group is required to notify members that recovery is in progress, and (3) if loss occurs within the group, some receivers will receive duplicate retransmissions. Implosion may become a problem only for losses inside a group, but only if the group size is large. Typically we expect that large groups will be divided into smaller groups before implosion becomes a problem. However, if this is not feasible, large groups can control internal implosion by using one of the sender or receiver-controlled implosion mechanisms described earlier.

An important feature of MRTs is the separation of data recovery from data dissemination. The separation allows optional receiver participation in error recovery, optimized MRT topology and the decoupling of dissemination and recovery protocols. MRTs trade complexity for performance. The complexity of MRTs is higher than either sender or receiver-controlled implosion because they require additional connections and group management protocols. However, MRTs do not interfere with data transfer: for example, operations like receiver additions and deletions are mostly contained within the nodes. The RN tree is expected to be slow changing, e.g. it changes only when a new group is added or deleted, and even then, only the parent node is affected.

1. This can be avoided if RNs can inject retransmissions in the MDT

5.6 Evaluation plan

While the performance of MRTs appears very promising, their success depends on keeping their complexity and overhead low. Our evaluation will therefore focus on characterizing the complexity of MRTs, by designing and implementing on our local ATM testbed the four MRT components described earlier. In designing the MRT and group management components we will examine algorithms from distributed systems. We will also investigate options and trade-offs on how the RNs locate an appropriate parent and how receivers aggregate to form groups. Following the design, we will examine possible locations of these components in the protocol stack. Some options are, integrating the components with the transport protocol, the signalling or reservation protocol, or implementing them as network daemons. Our evaluation will utilize a combination of implementation and simulation similar to the one described earlier in sender-controlled implosion.

6. CONCLUSIONS

In this paper we have presented a set of scalable error recovery methods for multicast applications that incorporate implosion control. We have discussed the mechanisms and trade-offs of these solutions. We plan to investigate the proposed solutions in detail and produce systematic evaluations of their trade-offs and performance. We will also plan to produce practical implementations of these solutions both in the Internet and on our local ATM testbed.

REFERENCES

[1] Clark, D., Jacobson, V., Romkey, J., and Salwen, H., “An analysis of TCP processing overhead”, IEEE Communications Magazine, Vol. 27, No. 6, Jume, 1989, pp. 23-29.

[2] Dalton, C., Watson, G., Banks, D., Calamvokis, C., Edwards, A., Lumley, J., “Afterburner,” IEEE Network, Vol. 7, No. 4, pp. 36 - 43, July 1993.

[3] Deering, S., “Host Extensions for IP Multicasting,” RFC 1112, January 1989.

[4] Dittia, Z., Cox, J., and Parulkar, G., “Design of the APIC: A High Performance ATM Host-Network Interface Chip,” Proc. IEEE INFOCOM 95, Boston, 1995, pp. 179-187, 1995.

[5] Doeringer, W., Dykeman, D., Kaiserswerth, M., Meister, B., Rudin, H., Williamson, R., “A Survey of Light-Weight Transport Protocols for High-Speed Networks.” IEEE Transactions on Communications, Nov. 1990.

[6] Floyd, S., Jacobson, V., McCanne, S., Liu, C., Zhang, L., “A Reliable Multicast Framework for Light-Weight Sessions and Application Framing,” Proc. of ACM Sigcomm ‘95, pp. 342-356, Cambridge MA 1995.

[7] Holbrook, H., Singhal, S., Cheriton, D., “Log-based Receiver-reliable Multicast for Distributed Interactive Simulation,” Proc. of ACM Sigcomm ‘95, pp. 328-341, Sept. 1995.

[8] Paul, S., Sabnani, K.,. Kristol, D., “Multicast Transport Protocols for High-Speed Networks,” Proceedings of International Conference on network Protocols,” pp. 4 - 14, 1994.

[9] Pingali, S., Towsley, D., Kurose J., “A Comparison of Sender-initiated and Receiver-initiated Reliable Multicast Protocols”, SIGMETRICS ‘94.

[10] “XTP Protocol Definition Revision 3.6,” Protocol Engines INC., 1992.

[11] Yavatkar, R., Griffioen, J., Sudan, M., “A Reliable Dissemination Protocol for Interactive Collaborative Applications,” to appear, Proc. of ACM Multimedia ‘95.