Error Control for Continuous Media and Multipoint Applications

Christos Papadopoulos
Washington University, St. Louis Missouri

1. INTRODUCTION

Emerging high-speed networks like ATM have stirred excitement with their potential to support multimedia, continuous media (CM) and multipoint applications. Example applications include conferencing, collaborative work, video-on-demand, information browsing, multimedia magazines, voting systems, etc. In addition to high bandwidth, future networks will provide both quality of service guarantees to support CM applications and efficient multicast to support multipoint applications. However, before these new network services can be successfully realized, changes are required in other protocol layers, the network interface [12] and the operating system [16].

A protocol layer that will undergo significant changes is the transport layer. The traditional transport layer functions, including connection management, error, flow and congestion control, have to evolve to support CM and multipoint applications. To support CM applications, the transport layer must provide bounded latency and loss; to support multipoint applications, the transport layer must provide efficient and scalable multipoint error, flow and congestion control.

In this work, we focus on transport layer error control. Specifically, we are interested in how error control must evolve to be applied to (a) point-to-point CM applications and (b) multipoint applications with a large number of participants. Existing error control mechanisms are unsuitable for these applications because they fail to meet two important requirements: bounded latency required by CM applications and efficient scaling required by multipoint applications.

Error control as implemented in existing transport protocols (e.g. TCP [22]) offers 100% reliability at the expense of latency. For CM applications the limited lifetime of the data makes this trade-off inappropriate. CM applications require error control mechanisms that will not violate the application's latency requirements. For this reason error control for CM applications (especially retransmission-based error control) has created some controversy: some researchers question the success of error control due to the limited time available for recovery, and even its necessity, due to the periodic nature of CM applications. However, we argue that retransmission-based error control can be adapted and successfully applied to CM applications with minimal cost and will also help reduce the bandwidth allocation required to maintain acceptable loss characteristics in VBR applications with high peak-to-average ratio (e.g. compressed video).

In multipoint error control, as the number of participants becomes very large, a new problem emerges, namely *implosion*. Implosion occurs when a large number of endpoints in a multipoint connection send messages simultaneously. For example when a loss occurs, several receivers may send retransmission requests, or several receivers may generate acknowledgments 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 processing and latency from the numerous messages, waste of network bandwidth and loss of messages from buffer overflow. Therefore, in order to be scalable, error control in multipoint communication must be augmented with implosion control.

In this proposal we will investigate solutions to the above problems. Our goal is to provide insights into the issues and trade-offs of these problems as they arise in future networks and produce practical and efficient solutions on our local ATM testbed. The thesis will contain a detailed examination of the issues and trade-offs, performance evaluations of the different options and a detailed description, justification and performance analysis of the selected solutions. The contributions of our work will be as follows:

- design and implementation of a simple, inexpensive and efficient retransmission-based error control mechanism for CM applications and the demonstration of its effectiveness
- design, analysis and implementation of scalable multipoint error control mechanisms, augmented by a set of implosion control mechanisms that take into account the nature of the underlying network (connectionless or connection-oriented), the type of communication (one-to-many or many-to-many) and the desired performance/complexity trade-off

This proposal is organized as follows: Section 2 presents the perspective of our work and describes how the work fits in the design of future transport protocols. Section 3 presents the design of an error control mechanism for CM applications and our evaluation plan. Section 4 describes the problem of implosion as it arises in error recovery and divides implosion solutions into three approaches, namely sender-controlled, receiver controlled and hierarchical implosion control. Sections 5, 6, and 7, present solutions and evaluation plans based on each of the three approaches. Finally, section 8 concludes this proposal.

2. PERSPECTIVE

Multipoint multimedia applications not only have a large and diverse set of distinct requirements, but within each requirement the range of support can vary enormously. The table in Figure 1 presents some of

Application Requirement	Options				
connection configuration	one-to-one	one-to-many	many-to-many		
connection size	small (2 - 10).....	medium (10 - 100)......	large (100 - 1000)		
quality of service	none	loss	delay	jitter	media synchronization
error control	none	application-specified	100%		
flow control	none	variable/policy-based	fully variable		
latency	100 μSec..........	33 mSec........	200mSec.........	1Sec........	

Figure 1: Multipoint multimedia application requirements

the range and diversity of the requirements. Each row in the table lists a distinct requirement of multipoint-multimedia applications and each column lists the range of support for the corresponding requirement. For comparison, the shaded entries show the requirements of the current generation of applications. Note the large increase in both the number of requirements and the range of support within each requirement. Some requirements have a small range: the connection configuration for example, can be one of three types: point-to-point, one-to-many, or many-to-many; respective application examples are file transfer, data distribution and collaborative work. Other requirements, however, have a support range that can vary by several orders of magnitude: for example, the number of endpoints a multipoint connection can range from two to possibly thousands (for entertainment audio and video), spanning three orders of magnitude;

latency requirements can vary from microseconds (RPC, PVM [24]) to milliseconds (audio, video, interactive applications), or seconds (file transfer), spanning almost six orders of magnitude.

Creating a protocol that can support the range of even one such requirement is in itself a challenging task. However, multimedia applications will require the simultaneous support of several such requirements. For example, interactive applications require latency guarantees *and* efficient support for multiple endpoints; PVM applications require latency guarantees, multipoint support, flow and error control; browsing applications require latency guarantees, flow control, but no error control. Some applications may even switch requirements during the lifetime of a connection: for example, when in a multimedia database search the user stops browsing and begins retrieval. Supporting multiple requirements is not easy: in many cases requirements may call for contradictory solutions: for example, to provide multipoint support a protocol may reduce state. However, if error and flow control are also required, the state must be increased to store sequence numbers, buffers, retransmission bitmaps, etc.

We believe that it is unlikely that a single protocol can be designed to support all the above requirements efficiently. In this thesis we focus on one such requirement, namely error control and explore problems and solutions for error control when combined with other requirements, namely connection configuration, connection size and application latency.

3. RETRANSMISSION-BASED ERROR CONTROL FOR CONTINUOUS MEDIA APPLICATIONS

In this section we present the motivation and design of an error control mechanism for continuous media applications based on retransmission. First we argue that retransmission-based error control is less costly than other methods of error control and can be adapted for a wide range of continuous media applications. Then, we present the key components of our mechanism, followed by a discussion about its complexity and trade-offs. Finally, we present our plan to evaluate the mechanism on our ATM testbed and conclude with related work.

3.1 Motivation

With the emergence of high bandwidth-delay networks, retransmission-based error control has drawn criticism about its appropriateness for continuous media applications. The criticism is twofold: (a) retransmission is claimed to be inappropriate because it requires at least one additional round-trip delay for recovery, and (b) retransmission requires CPU participation, which is claimed to be too complex for continuous media applications. The proposed alternatives are to minimize loss with *peak-bandwidth allocation*, *concealment* and *Forward Error Correction (FEC)* [2].

Transmitting *variable bit-rate* (VBR) traffic requiring deterministic loss and delay guarantees means that peak-bandwidth reservation is needed for the duration of the connection. For applications with high peak-to-average ratio, peak-bandwidth allocation is too expensive and inefficient. However, transmitting VBR with statistical guarantees on loss and delay while reserving bandwidth close to the average will result in a high loss rate for applications with high peak-to-average ratio. For example, in [14] a two-hour sci-fi action movie was compressed and the resulting data analyzed. The analysis showed that the average bandwidth of the resulting VBR compressed video was 5.34 Mbps and the peak bandwidth almost 3 times higher (15 Mbps). These peaks were infrequent as witnessed by the coefficient of standard deviation (0.23) and therefore reserving peak bandwidth for the duration of the connection is wasteful. On the other hand, losing data in the network would lower the quality and compromise the entertainment value of the movie. This is especially important for future HDTV.

An alternative to peak-bandwidth allocation is to reserve less than the peak bandwidth and attempt to conceal lost data by taking advantage of the inherent redundancy in CM applications. Examples of concealment include substitution of lost data with previously received data, interpolation and approximation. Supporters of this approach claim that the network loss rates will be low and thus concealment will be adequate. It is clear, however, that as the loss rate increases, the effectiveness of concealment decreases. Moreover, the complexity of concealment increases as the loss rate increases. Therefore, lowering the error rate not only results in better concealment, but requires less complex concealment techniques.

The two choices for a recovery mechanism are retransmission and FEC. FEC requires less time to recover, but requires greater bandwidth than retransmission because it always transmits redundant information, in contrast to retransmission which retransmits only lost data. Despite the additional latency in retransmission, we believe that with enhancements retransmission-based error control can be applied to continuous media applications, while at the same time being significantly less costly than FEC[1]. We expect that with the low loss probability in future networks, as little as one retransmission attempt will suffice in most cases.

With this research, we will attempt to answer the following questions:

1. Is retransmission-based error recovery possible with continuous media applications?
2. How much does recovery cost in terms of bandwidth, protocol complexity and latency?
3. Does it improve the effectiveness of concealment and by how much?

In order to answer the above questions we will design, implement and evaluate an actual protocol for transmission of continuous media. The implementation will be done on our local ATM testbed and the experimentation will include various continuous media, including audio and MPEG video.

3.2 Making retransmission work

Retransmission requires the following actions to recover lost data: (1) loss detection (typically timer-based in existing protocols), (2) sending of a retransmission request by the receiver, (3) sending a retransmission by the sender, and (4) receiving the retransmission. These actions introduce an additional round-trip delay, which may cause continuous media to miss the presentation deadline or *expire,* before a retransmission is received. In order to maximize the probability that a retransmission will arrive before data expires, the protocol uses the following techniques: (1) playout buffering at the receiver and the sender, (2) implicit expiration of the sender retransmission buffer, (3) gap-based loss detection rather than timer-based loss detection, and (4) conditional retransmission. In addition, the mechanism delivers data integrity information to the application to help in concealment. We describe these techniques next.

3.2.1 Playout buffering

The time available for retransmission in CM applications may be increased by introducing limited buffering at the receiver, with no perceptible deterioration in quality by the user. This is called *playout buffering*[2] and the buffering delay is called *playout delay.* In interactive applications the playout delay is limited by the perceptual tolerance of the user, which is around 200 mSec [3]. Therefore, for NTSC video a playout delay of 100 mSec allows three consecutive frames to be buffered at the receiver before delivered to the application (see Figure 2). Buffering increases the time available for retransmission by 100 mSec, which is sufficient for several retransmission attempts in LANs and MANs where the round-trip-

1. FEC is typically used in lossy satellite communication. We find it contradictory to opt for the costlier error control mechanism, namely FEC, when the network error rate is expected to decrease by orders of magnitude

2. Note that the playout buffer will also act as a jitter absorption buffer

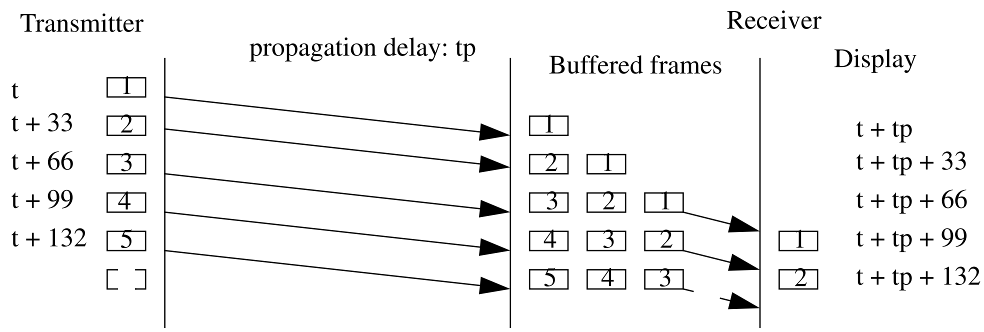

Figure 2: Buffering video frames at the receiver

delay (RTD) is about 10 - 15 mSec. It may even be sufficient for one retransmission on cross-country connections, where RTD is about 60 mSec. It is important to note, however, that the 200 mSec threshold exists in interactive applications only. In case of distribution (e.g. video-on-demand) the playout buffering can be quite large (e.g. seconds) without any adverse effect on the perceived video quality. It hardly matters to a viewer if the display of a movie starts a few seconds later if a continuous, error-free playback can be guaranteed from that point on.

The size of the playout buffer is proportional to the data rate. For example, buffering 3 frames of NTSC compressed video (5 - 10 Mbps), requires a playout buffer of 0.0625 - 0.125 Mbytes. For compressed HDTV (about 20 Mbps), the size of the playout buffer for three frames is 0.25 Mbytes. These numbers are well within the capabilities of modern workstations. It remains to be seen, however, if they are within the capabilities of television set-top boxes. We are optimistic that this will not be a problem: some high-end television sets already have the capability to store frames for special effects (freeze-frame, slow motion, etc.), which indicates that such features may soon become common.

3.2.2 Implicit retransmission buffer expiration

The periodic nature of the data in CM allows the sender to estimate how long to keep data in its retransmission buffers before discarding it. Therefore, an explicit acknowledgment from the receiver is not required. The sender can simply maintain a retransmission buffer with the same number of slots as the receiver's playout buffer. Whenever new data is sent the oldest data in the retransmission buffer is discarded because it has expired. Implicit data expiration allows the sender to release its retransmission buffers when data expires without explicit indication from the receiver. It also helps avoid late retransmissions: if a retransmission request is delayed and arrives after the data has been discarded, no retransmission is sent.

3.2.3 Gap-based loss detection

Loss detection is typically done with time-outs. In time-out loss detection, each packet is associated with a timer at the sender. If the timer expires before an acknowledgment is received, the packet is retransmitted. The time-out values are typically large (several times the RTD), and therefore time-out loss detection is not appropriate for continuous media applications. We believe that gap-based loss detection at the receiver is more appropriate [15]. It is important to note, however, that gap-based loss detection is applicable only if the underlying network preserves packet sequence.

In gap-based loss detection, each packet contains a sequence number, as depicted in Figure 3. A gap is detected when a packet arrives with a sequence number higher than expected. Gap-based loss detection has some important differences from time-out loss detection:

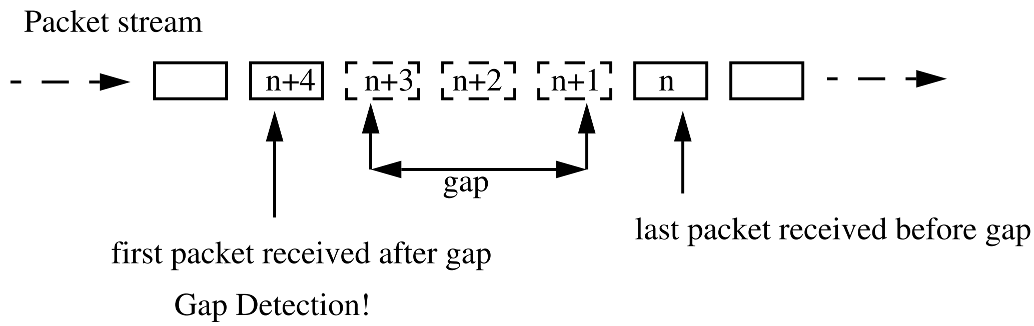

Figure 3: Gap detection at the receiver

- a gap is detected only after another packet is received, thus gab-based loss detection is more suitable for applications that send data frequently (e.g. CM applications)
- loss detection is much faster than timer-based loss detection, if the loss is limited to a few packets
- no per-packet timers at the sender are required

Gap-based loss detection may take longer than time-out loss detection during prolonged network loss periods. However, it is questionable whether a retransmission request should be sent before the network loss period is over. With gap-based loss detection, loss is detected after the reception of a packet, so there is a good possibility that the network loss period is over (assuming no packet-discarding strategies are employed at the network[25]).

3.2.4 Conditional retransmission

Since data in continuous media has a limited lifetime, the protocol must employ a mechanism to abort retransmission when the time left before presentation is less than the RTD. This may happen after multiple retransmission attempts have failed, or if the network latency has become too large. Late retransmissions not only waste network bandwidth and CPU cycles, but contribute unnecessarily to congestion and may delay new data. To avoid late retransmissions, the mechanism must keep an estimate of the round-trip delay and the presentation time of the data at the receiver and must ensure that the presentation time is always greater than the current RTD estimate before requesting a retransmission.

3.2.5 Data integrity information delivery

The mechanism does not guarantee that data delivered to the application will be error-free. The mechanism, however, maintains explicit information about the integrity of the received data, which can be delivered to the application. This information includes an indication of missing data and location. We believe that such information is valuable to the application in deciding how to handle incomplete data, e.g. in applying concealment.

3.3 Complexity

The error control mechanism is a variation of selective repeat and therefore its complexity similar to the complexity of other selective repeat mechanisms [6, 7, 19]. The appropriateness of this type of error control for high-speed networks has been already demonstrated. Our mechanism includes some additional complexity, which we discuss next:

3.3.1 Retransmission decision:

The information required for the retransmission decision at the receiver are estimates of the RTD of the connection and the period of the continuous media. The RTD can be estimated by periodically bouncing a timestamped packet between the receiver and the sender. The new RTD estimate is derived after applying a smoothing function, e.g. $RTD = (a * RTD_{new}) + (1 - a) * RTD_{old}$. We expect that this simple smoothing function will suffice in connection-oriented ATM networks, where the RTD variance is expected to be low. The period of the CM stream will be one of the parameters passed to the transport protocol by the application during connection setup, in order to determine the size of the playout buffer.

3.3.2 Retransmission and playout buffer allocation and management:

The size of the retransmission and playout buffers depends on the RTD and the period of the continuous medium. The size, however, is expected to be small (see “Playout buffering” on page 5). The buffer management operations are the same as those used in other selective repeat protocols. The only difference is the increased number of buffer slots, which are managed as a FIFO queue. The state and the number of operations required to maintain a FIFO queue are small: the state consists of a head and tail pointer, and the operations take around 10 instructions.

3.3.3 Playout buffer status update and delivery to application

The playout buffer status consists of a bitmap marking the presence/absence of packets. The bitmap is similar to the bitmap used in other selective repeat protocols, except that one bitmap per buffer slot is required. In contrast to other protocols, the bitmap along with other information must be made available to the application. This requires some protocol processing to prepare a data structure containing a flag indicating loss/no loss, the loss bitmap and the packet size (assuming a constant packet size). A more important consequence is that the application interface may have to be modified to support status delivery. Ideally, the application interface must support data and status delivery in a single system call. However, this is not always possible, e.g. in the BSD socket interface. We consider this a limitation of the application interface, not the protocol.

3.4 Status and evaluation plan

We have completed the design of a transport protocol with the above error control mechanism and we have implemented the protocol in the SunOS kernel, on top of IP. The protocol is being ported to NetBSD on top of ATM. The protocol attempts one retransmission, which we expect should suffice in most cases. However, we plan to enhance it to attempt multiple retransmissions. The user selectable parameters include the size of the retransmission and playout buffers, the packet size and the period of the continuous media. We are planning to evaluate the protocol as follows:

1. Instrument the protocol to measure packets lost, packets recovered and record the end-to-end latency.
2. Implement a tool to emulate network delay and loss. The tool will consist of a machine with a kernel modified to receive, delay and/or drop and forward packets of the protocol. The tool will be used as shown in Figure 4 to emulate different network delays and loss models.
3. Conduct experiments with different the network delays and loss probabilities. Investigate trade-offs between the playout buffer size, RTD, loss and latency. For example, we want to determine the required playout buffering required for a given RTD and loss rate.
4. With the help of researchers at NEC in Princeton, NJ, we will assess the effectiveness of the protocol in improving concealment with the following experiment: transmit an MPEG stream; for each received frame record the number and location of lost and recovered packets; perform off-line concealment[1] of the original MPEG stream to produce two versions, one excluding and another including data recovered by the protocol; display the two streams for subjective evaluation.

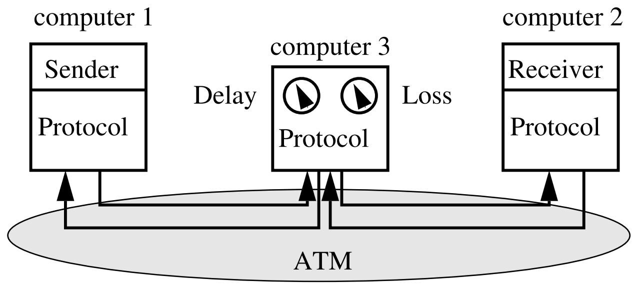

Figure 4: Protocol evaluation environment

3.5 Related work

Little work has been done in designing error control mechanisms for CM applications. In most CM transport protocols, errors are generally not recovered [23]. In [11] work has been done in developing an end-to-end model of packet voice transmission. However, in voice transmission the authors assume that the control time in talkspurts can be extended without any adverse effect on playback. The same assumption is not true for video, where once playback begins the frames must be displayed at fixed intervals. We believe that variable frame rate video is annoying to the users and therefore we do not advocate this option.

Other researchers have relied on FEC for error control for multimedia applications [27]. However, as discussed earlier, we consider retransmission-based error control as preferable to FEC because of its lower cost.

4. IMPLOSION IN MULTICAST ERROR CONTROL

In this section we first describe implosion and how it occurs in multicast error control. Then, using a simple analytic model we derive an estimate of the cost of implosion and argue that it is very high.

4.1 Motivation and problem statement

Multicast networks can achieve optimal delivery of messages from a source to multiple destinations by delaying the replication of a message until it has to traverse separate paths. This results in only one message traversing any particular link in the network. The same is not true, however, for messages returned from the destinations back to the source. Ideally, such messages should be merged at the branching points as they traverse the multicast tree in the reverse direction, so that a single composite message is finally delivered to the sender. However, due to different propagation and processing delays at the receivers, it is hard to synchronize messages at the branching points. In addition, merging several messages into one is a complex operation and typically requires software processing, in contrast to replicating a single message which can be easily done in hardware. For these and other reasons (e.g. security), messages sent by the receivers are usually delivered separately to the sender.

As the number of receivers increases, so does the number of messages returned to the sender. If the number of messages exceeds some threshold, two things may happen: (1) the sender may be swamped with messages and communication will be severely impaired or even halted, or (2) the message bandwidth at some link may exceed the available network bandwidth. This problem is called *message implosion*. An example of message implosion is depicted on the left side of Figure 5, in the context of error control. The

1. concealment is currently done in software and is therefore too expensive to do in real-time

figure shows the multicast tree for an one-to-many connection at an instant after a packet was lost on the left branch of the tree. The loss triggers retransmission requests from all receivers in the left subtree, causing implosion at the sender.

Messages that can cause implosion can be initiated by entities residing at any communication layer which sends messages back to the sender. However, we focus on implosion control in error control at the transport layer. The types of messages that can cause implosion in error control include acknowledgments, retransmission requests and redundant retransmissions. It is important to note that many of these messages may be synchronized (e.g. acknowledgments or retransmission requests) which further exacerbates the problem.

In multipoint connections, error control may cause not only implosion at the sender, but also redundancy at the receivers. This is shown at the right side of Figure 5, where some receivers receive redundant

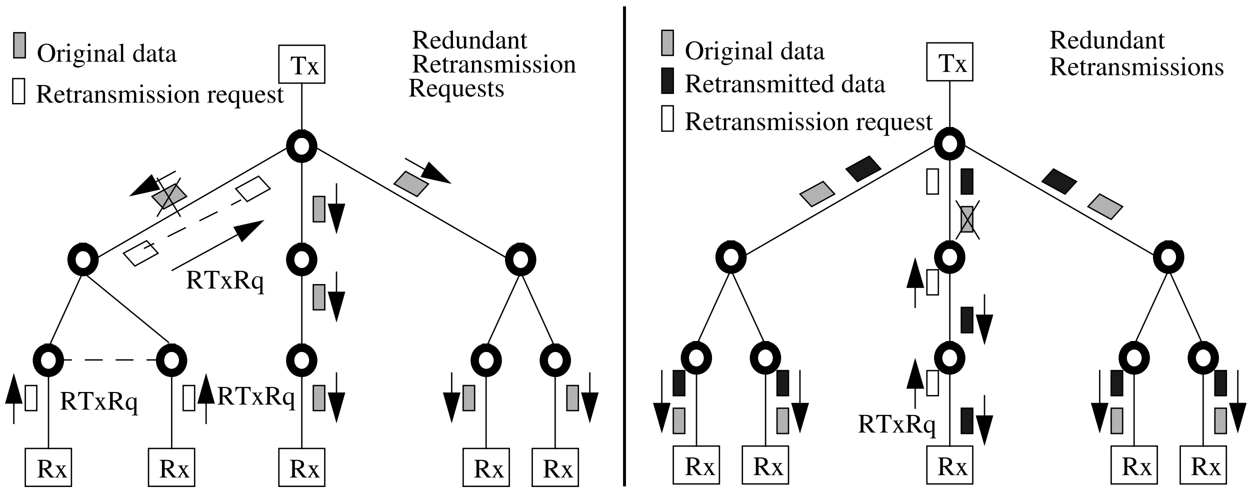

Figure 5: Redundancy in retransmission requests and retransmissions

retransmissions. Although both implosion at the sender and redundancy at the receivers are very important problems, implosion is more serious because a single endpoint can find itself being bombarded by messages from several other endpoints. Also, solving the redundancy problem requires sending retransmissions over either point-to-point connections which can be wasteful, or over dynamic multicast subgroups (created to send one retransmission and then torn down) which is too complex. Therefore, the less-than-optimal solutions ignore the redundant retransmissions problem. The near-optimal solution, however, does address the problem.

4.2 Cost of implosion in error control

Before presenting solutions to the implosion problem, we first have to convince ourselves that the problem is worth solving. We do this by estimating the cost of implosion in terms of messages, using a simple analysis. As a disclaimer, we point out that the analysis is not meant to accurately estimate the cost of implosion, but rather to convince us that there indeed exists a serious problem. Therefore we make assumptions that are meant to simplify the analysis, not accurately reflect real life situations. However, we believe that the analysis is accurate enough to reveal the existence of a problem, if one exists.

We assume a one-to-many uniform multicast tree where the endpoints lie at the leaves and all the intermediate nodes are switches. Switches receive a packet from the upstream link, replicate it and send it to the appropriate downstream links. We assume equal loss probability at the switches, which we define as the probability of a switch not receiving a packet. Therefore, a lost packet results in loss at all the receivers in the sub-tree rooted at the first switch that missed the packet. Assume the probability a switch will miss a packet to be p. Let the height of the multicast tree (counting only the switches) be h, and assume each switch has the same fan-out k. Then for each new packet sent the number of generated retransmission requests is:

$$n = k^h \cdot \left(1 - (1-p)^h\right)$$

By substituting values and plotting the results, we obtain the set of graphs in Figure 6. The graphs

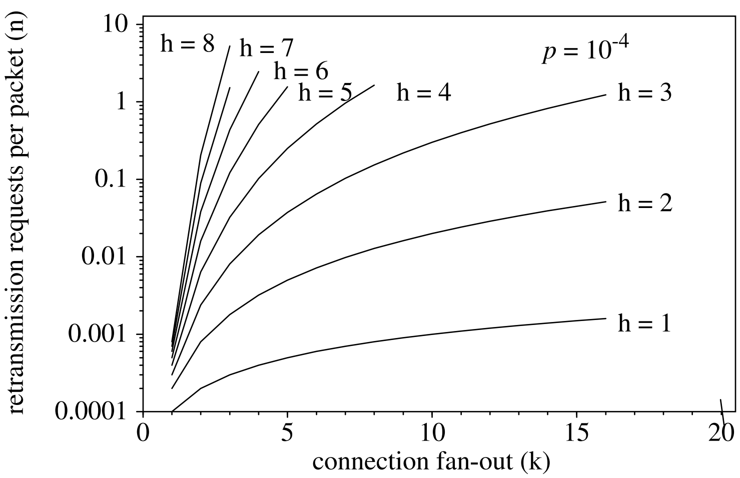

Figure 6: Retransmission requests reaching transmitter

show the number of retransmission requests received by the transmitter for every packet it sends out, versus the fan-out at each switch. For all graphs the loss probability is set to 10^{-4}. The graphs show that the number of the received retransmission requests increase at an alarming rate as a function of the height of the tree and the connection fan-out. For example, in a connection with height = 4 and fan-out = 4 (which contains in 256 endpoints), the transmitter receives one retransmission request for approximately every 10 data packets it sends out. This means that in the multipoint case the sender will see an increase of 3 orders of magnitude in the number of control packets compared to the point-to-point case. Such an increase will drastically reduce performance due to increased protocol processing and the number of OS interrupts. It is important to note that the situation is very similar with flow control, or with other mechanisms generating feedback. In flow control for example, changes in the load of a link may cause all downstream receivers to send messages back to the sender creating a situation similar to packet loss.

5. CONTROLING IMPLOSION IN MULTICAST ERROR CONTROL

In thissection we define the properties of optimal implosion control and argue that implementing an optimal solution is very hard. Thus we proceed to define a taxonomy of less than optimal, but easier to implement solutions, one of which comes very close to being optimal. Later sections present examples of such solutions.

5.1 Optimal solution to implosion

We define an optimal solution to implosion as one that makes a multipoint connection appear to the sender as a point-to-point connection 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 the same time as in a point-to-point connection. The sender is presented with the illusion that it communicates with a super-receiver which aggregates the messages from all receivers. An optimal solution must possess certain properties in terms of messages and latency, which we define as the following:

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 endpoint that first discovers loss and a retransmission is sent immediately by the endpoint closest to the requesting endpoint

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 the receivers, the number of messages is minimal and does not contribute to implosion

A significant amount of work has been done in optimizing the implementation of point-to-point transport protocols [13]. 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. It is important that the efficiency gained in optimizing point-to-point protocols finds its way to multipoint protocols and the efficiency is not compromised by the implosion control mechanism. However, designing and implementing an optimal implosion solution 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 without relying on explicit message exchange.

Despite its complexity, defining an optimal solution to the implosion problem 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.

5.2 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 will be 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 required 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 a key *initial* assumption: we assume 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.

5.3 Implosion control taxonomy

A multipoint connection has 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 created 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, and (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), or not allowed for security reasons.

Receiver-controlled implosion

In receiver-controlled implosion the secondary channel must allow messages sent by any receiver to reach all other receivers so that receivers can communicate to control implosion. Since messages reach everyone, receivers must be very cautious before sending. For example, on loss each receiver may calculate its own back-off delay, listen for requests from others while its timer is running and transmit a request only if no-one else did so. 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 and many-to-many connections (e.g. IP Multicast, ATM many-to-many connections).

In both sender and receiver-controlled implosion, we assume that retransmissions are sent over the pri-

---

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.

mary channel and reach all receivers even if they 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 handle, whereas in receiver-controlled implosion the number of messages must be reduced below the maximum the sender or any receiver can handle.

Hierarchical implosion control

In large multicast connections a natural approach is to use a hierarchical solution. For example, the receivers can be organized into 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 avoids sending retransmissions to the whole group. Parents can use the secondary channel for retransmissions if it is bidirectional, or they can insert retransmissions into the primary channel.

In the following three sections we investigate a solutions based on each category. In each case we present an outline of the mechanism and point out specific trade-offs. In addition, as part of the hierarchical solution, we examine message consolidation.

6. SENDER-CONTROLLED IMPLOSION

In this section we begin by presenting a basic mechanism for sender-controlled implosion. We assume an environment where the sender is solely responsible for controlling implosion because communication among receivers is either not possible or not desirable (e.g. in voting systems). Following the description of the basic mechanism, we discuss trade-offs and propose a set of enhancements to the basic design. Finally, we present our evaluation plan.

Note that receivers are isolated by definition in one-to-many bidirectional connections in connection-oriented networks like ATM. Such connections are suitable for multimedia dissemination. To the best of our knowledge, no implosion control solutions have been proposed so far for these types of connections.

6.1 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 retransmission initiated by the previous receiver. Such a linear chain ensures that at most one request reaches the sender at the expense of introducing very high latency (when the receiver set is large). However, some redundancy can typically be tolerated at the sender and typically not every receiver will send a request. Thus instead of a linear chain, receivers can be bundled into groups where the whole group is allowed to send requests simultaneously. Grouping makes the chain shorter and fatter, reducing latency at the expense of generating redundant requests, which increase from O(1) in linear chaining, to O (maximum group size). Therefore, it is critical that the sender is allowed to control the size of each group to ensure that it stays below the sender's implosion threshold.

We now present a basic mechanism based on the above observations. The receivers are grouped into a chain of "buckets" based on their RTD from the sender, as shown in Figure 7. Grouping receivers based on

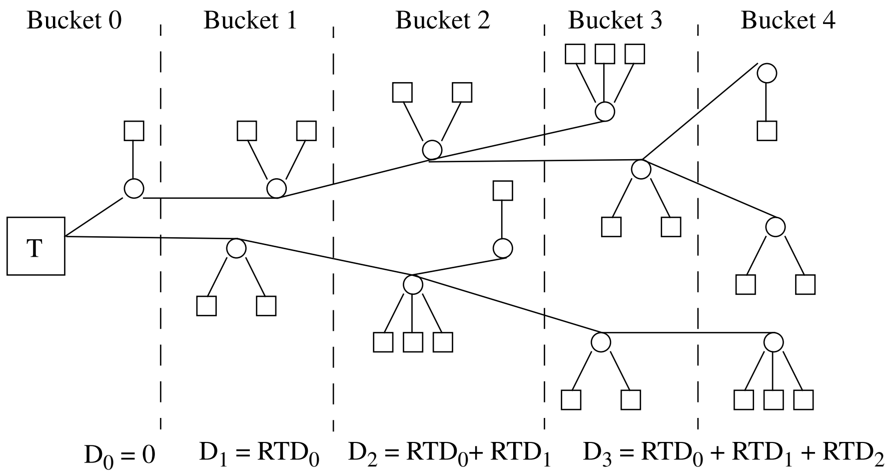

Figure 7: Grouping endpoints into buckets

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 the 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.

6.2 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 "4.3: Optimal solution to implosion" on page 12). 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. An important consequence of latency is that the retransmission buffers at the sender may be large. The mechanism in its present form does not scale well and the maximum number of receivers is bounded by the application latency requirements and the sender's implosion threshold. However, since the sender has full control of the receivers, the probability of implosion is minimized. 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.

6.3 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 most of the time 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 is what happens during congestion, which is expected to be the main cause of packet loss. 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.

6.4 Evaluation plan

Presently, wide-area ATM networks with the capacity for large scale connections are not available, which prevents us from evaluating the implosion control mechanisms in a real experimental environment. However, we believe we can still gain insight into the trade-offs by designing, implementing and simulating the mechanisms. Specifically, we propose to fully implement the sender and receiver mechanisms in a transport protocol, model their behavior and use the model to simulate implosion control in a large scale connection as described next:

1. Create a complete specification and implementation of a protocol incorporating sender-controlled implosion on our local ATM network. The implementation will be tested using one sender and as many receivers as possible in our local environment (we expect to use between 10 and 20 receivers).
2. Instrument the implementation to measure the latency added by the implosion control mechanisms at the sender and the receivers. The most important measurements are: (a) the latency to send a retransmission after receiving a request, (b) the number of redundant requests and the associated processing overhead (to find the sender's implosion threshold) and (c) the processing overhead of the implosion mechanisms at the sender and receivers.
3. Create a simulation of a wide-area network using generally accepted values for bandwidth, latency and error rate, where connections with a large number of receivers can be set up in various topologies (LANs, MANs, WANs). Use the measurements obtained from instrumentation to simulate the sender and the receivers.
4. Using the simulation, experiment with the various implosion control mechanisms, different network loss characteristics, and receiver topology. Obtain results for average and maximum latency, error and implosion control effectiveness, buffer requirements, and processing overhead.
5. Fine-tune and optimize the implementation based on knowledge gained from the simulation.

At the end of our evaluation we expect to be able characterize the trade-offs of the proposed implosion control mechanisms in the ATM environment. Based on our findings, we will fine-tune the implementation of the protocol. The protocol will be documented and distributed to interested parties for experimentation.

7. RECEIVER-CONTROLLED IMPLOSION

In contrast to sender-controlled implosion, there has been activity on receiver-controlled implosion in the last few years. One of the most significant is the receiver-controlled implosion mechanism introduced in XTP in the form of *damping* and *slotting* heuristics [26]. Recently, an enhancement to the original XTP mechanism was presented for the Internet [17]. We are not yet aware of systematic evaluations either of the original XTP mechanism or the Internet version, although we expect them soon. However, we expect future evaluations to be focused on the Internet environment, whereas we are mostly interested in ATM.

The significance and popularity of the mechanisms coupled with the lack of a systematic evaluations, has prompted us to chose to carry out our own evaluations in the context of ATM. We expect that in connection-oriented environments the issues and cost factors will be different than those in connectionless environments. For example, the cost of setting up and maintaining many-to-many connections is drastically different: although many-to-many connections are readily supported on the Internet with IP Multicast [10], they are currently very expensive in ATM, requiring a one-to-many connection from each endpoint.

In this section we begin by describing the operation of the Internet mechanism, followed by the performance characteristics and trade-offs. We conclude with our evaluation plan.

7.1 Basic ideas

The receiver-controlled implosion mechanism presented here assumes that receivers are capable of sending retransmission requests and retransmit packets. Endpoints (senders and receivers) are connected via a many-to-many connection, thus everyone receives all requests and retransmissions. The mechanism requires endpoints to maintain RTD estimates between themselves and every other endpoint in the connection. The mechanism attempts to minimize the number of messages and latency by using two timers at each endpoint: a timer for sending requests and a timer for sending retransmissions. Intuitively, the sequence of events depicted in Figure 8 is as follows: on loss, each receiver sets its request timer to a value

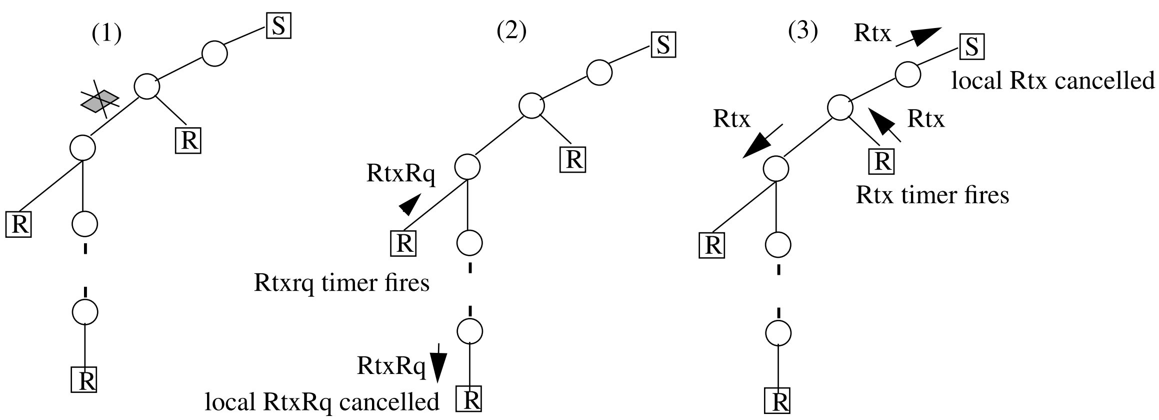

Figure 8: Receiver-controlled implosion phases

proportional to the RTD between itself and the original sender. This causes receivers closer to the sender to send requests faster than other receivers. If these requests are received by other receivers before their timer expires, they cancel similar pending requests. All endpoints that can retransmit the requested data start a retransmission timer with a value proportional to the RTD between them and the requesting receiver. This causes the retransmission to be sent by the endpoint closest to the requesting receiver. The retransmissions also cancel similar retransmissions pending at other endpoints. The mechanism's actions are outlined next:

Receiver actions:

1. If a loss is detected at receiver R, set T to a random value i between p and 2p, where p is the propagation time from the sender to R
2. If a retransmission for the missing packet is received during this interval, cancel the request
3. If someone else's retransmission request is received during this interval, double i and pick a new delay interval between i and 2i
4. If T expires, multicast the request, double i and set T to a new delay interval between i and 2i again

Sender actions:

1. Upon reception of a retransmission request, delay random time between 0 and p (where p is the propagation delay between requesting receiver and sender)
2. If a retransmission is sent by someone else, cancel the timer
3. If the timer expires, multicast the retransmission

The mechanism differs from XTP in the selection of the request and retransmission back-off intervals: XTP chooses random values, while in this mechanism the intervals are set based on the propagation delay of each receiver from the sender.

7.2 Performance and trade-offs

The use of a many-to-many connection as the secondary channel violates the *isolation* and the *redundancy* optimality criteria (see "4.3: Optimal solution to implosion" on page 12). The *"recovery in one RTD"* criterion is also violated, because the mechanism introduces latency in the form of back-off delays in both sending requests and retransmissions. The extra latency can be estimated by observing that on average the first retransmission reaches the closest receiver at time $4 * p_{min}$[1] after the request is sent; if retransmission requests are dispatched immediately the latency is $2*p_{min}$. This, however, is a very good result compared to sender-controlled implosion where latency is O (number of buckets).

The performance of the mechanism is closely dependent on the receiver topology and the location where a packet was lost. In the best case, a single receiver close to the sender detects loss first and its request arrives in time to cancel requests from other receivers. Thus the mechanism works best if receiver RTDs are dissimilar. In the worst case, however, when receiver RTDs are similar, the mechanism does nothing to prevent implosion as shown in the following example[2]: assume that the sender and the receivers are connected in a star topology to a switch, as shown in Figure 9. The propagation delay between each endpoint and the switch is d. Assume a packet is lost at the switch. The receivers upon discovering the loss set their retransmission request timers to random (2d, 4d). Assume that the timer of some receiver expires after time 2d and a request is multicast. The request takes time 2d to arrive to the remaining receivers. However, by the time this request arrives at other receivers all timers will have expired with the result that all receivers send requests! Therefore, depending on topology the performance of the mechanism can vary from excellent to totally ineffective. Given the wide range in the theoretical performance, it is hard to predict what the performance will be in real-life scenarios without further evaluation.

The mechanism offers the following trade-offs which we plan to evaluate:

---

1. $(1.5 + 0.5 + 2)p_{min}$, where p_{min} is the propagation delay between the receiver whose request is received first, and the sender

2. To the best of our knowledge this problem has not been reported before

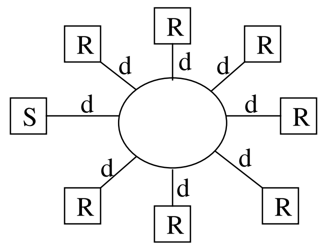

Figure 9: Star topology fools receiver-controlled implosion mechanism

- the mechanism reduces implosion by trading latency: the wider the time-out distribution the lower the chance of implosion.
- the mechanism trades absolute sender control (and therefore the guarantee that implosion will not occur) for distributed implosion processing, achieving better load balancing and scalability than sender-controlled implosion. In its present incarnation, the mechanism offers no means to the sender to request tighter implosion control.
- the mechanism trades complexity for faster recovery. Data is recovered from the closest endpoint at the expense of requiring endpoints to buffer data for retransmission, keep RTDs for other endpoints and execute implosion-related steps.

7.3 Evaluation plan

Our evaluation plan is similar to sender-controlled implosion: we plan to use a combination of implementation in ATM and simulation (see the previous section for details) to evaluate the mechanism's trade-offs identified above. Since many-to-many connections are more expensive in ATM than in the Internet, our investigation will also focus on characterizing the overall cost of the mechanism in terms of connection setup and maintenance.

8. 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 "4.5: Implosion control taxonomy" on page 13) 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. 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 followed by our evaluation plan.

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

8.1 MRT structure

MRTs form a tree hierarchy with the sender at the root. Each node of the tree is formed by a group of receivers. Each group has a designated receiver, called a Relay Node (RN). A RN is connected to all receivers in the group via a bidirectional one-to-many connection. Each RN is connected with the node's parent and children (if any), via point-to-point bidirectional connections. MRTs are constructed by the receivers. 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 10. It is important to note, however, 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 best-effort MRT.

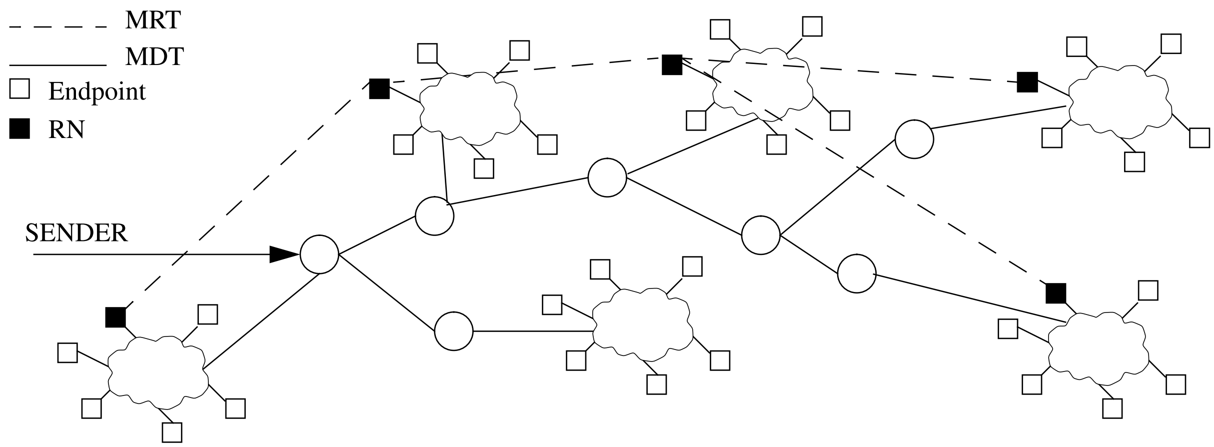

Figure 10: Multicast and recovery trees

8.2 MRT components

MRTs are composed of four components. These are:

1. 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, reforming the group after a failure.
2. Inter-group communication component: this component is responsible for communication between RN and its group. Typically this component is a reliable or unreliable one-to-many connection. The component is also responsible for detecting RN failure, if required.
3. MRT management component: this component deals with the interconnection and membership operations of RNs to form the MRT. It supplies each RN with information about its parent and children. It is also responsible for reforming the MRT in case of failure.
4. MRT communication component: this component is responsible for communication between RNs. Typically, this is a point-to-point connection.

8.3 Implosion control actions

The implosion control actions are described next. We present actions for two cases: (1) when loss occurs outside a node, and (2) when loss occurs within a node. For brevity, we omit some details like detection of failure.

Loss outside a group

1. 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.
2. 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.
3. If receivers in a group receive a retransmission request from RN before the above timer expires, 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

1. If some receivers but not RN experience loss, the back-off timer described in step 2 earlier expires and one or more requests are sent to RN. RN has the missing data (since it did not send a request) and retransmits the missing data to the receivers.
2. If some receivers including 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.

8.4 Consolidation

Some types of messages are not redundant and thus cannot be discarded. Therefore, 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 continues, until a single message is delivered to the sender. The structure of MRTs makes them highly suitable for consolidation by using RNs as consolidating nodes. Several issues arise in message consolidation. Examples include the following:

Selecting a method of message collection:

1. Periodic: the consolidating node sets a period to wait for messages. At the end of that period, the consolidating node combines all the messages it has received and forwards the consolidated message up the tree
2. Collecting a message from each receiver: the consolidating node waits for a message from every receiver before constructing the consolidated message
3. Sender initiated: The sender initiates message collection explicitly, specifying the method of message collection the consolidating nodes should use

Consolidation operations:

A set of operations need to be defined in order to specify how a node will perform message consolidation. Some example operations are the following:

1. MIN (x_1, x_2,...x_n): this operation for example can be used to consolidate fields indicating the highest packet sequence received by all receivers
2. MAX (x_1, x_2,...x_n) this may be used to indicate the maximum window size advertized by any receiver
3. OR (b_1, b_2,...b_n): this operation may be used to combine error bitmaps from all receivers to determine which packets have not been received by any receiver
4. AND (b_1, b_2,...b_n): this operation may be used to combine error bitmaps to determine which packets have been received by all receivers

8.5 Performance and trade-offs

The hierarchical solution to implosion adopted by MRTs is the natural solution for connections with a large number of receivers. The hierarchy allows MRTs to offer near-optimal implosion control performance, 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.

We characterize the performance of MRTs as near-optimal because performance comes very close to the optimal solution we defined earlier (see “4.3: Optimal solution to implosion” on page 12). 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 and only if the group size is large. Typically we expect to divide large groups into smaller groups before implosion becomes a problem. However, if the number of groups must be limited, large groups can control 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 node management protocols. However, MRTs do not interfere with data transfer: for example, operations like endpoint addition and deletion are mostly contained within the nodes. The RN tree is expected to be slow changing, e.g. it changes only when a new node is added or deleted, and even then, only the parent node is affected.

8.6 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 node management components we will examine algorithms from distributed systems. We will also investigate options and trade-offs on how RNs locate an appropriate parent and how receivers aggregate to form nodes. 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 ATM signalling protocol, or implementing them as network daemons.

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

We will characterize the complexity of MRTs in terms of the required connections, state, buffering and message exchange between the RNs and the receivers in a node, and the RNs with other RNs. Our evaluation will again utilize a combination of implementation and simulation similar to the one described earlier in sender-controlled implosion. At the end of our evaluation we will systematically document the trade-offs of MRTs and have efficient implementations of their components.

8.7 Related work

One of the first and most important works on reliable multicast is [5]. It is a centralized, reliable, totally ordered multicast protocol. [28] is an enhanced version of [5]. Implosion control is not addressed in the above protocols, or in many other multicast protocols [1, 4, 8, 18].

The first protocol to address implosion was XTP [26] with algorithms and heuristics known as the bucket algorithm, damping and slotting. These methods work only on many-to-many communication. A solution for one-to-many communication with small messages is presented in [9]. The solution is based on a multi-round approach where the sender actually allows responses from receivers to overflow its receive buffers and subsequently collects missed responses by multicasting the data multiple times. This is shown to reduce the total delay compared to allocating slots at the sender for receiver responses, at the expense of multicasting messages several times.

In all of the above solutions retransmissions are sent to the whole group. Other proposed solutions that avoid sending retransmissions to the whole group, are based on a hierarchical approach similar to ours [20]. However, the solution is integrated with the SNR protocol and it is suited only for reliable multicast. Moreover, the authors provide no details about how their hierarchy is formed and managed. In [21] three generic protocols were studied and it was confirmed that the performance of receiver initiated error control (NACK-based) is substantially higher than sender-initiated error control (ACK-based) in terms of host processing requirements.

9. CONCLUSIONS

In this thesis proposal we have presented experimental solutions for error control for multipoint and CM applications. The solutions are, a low-cost retransmission-based error recovery method for CM applications and a set of scalable error recovery methods for multicast applications that incorporate implosion control. We have discussed in detail the mechanisms and trade-offs of these solutions and presented evaluation plans for each solution.

The contributions of this research will be twofold: first, we will investigate the problems, the options and the proposed solutions in detail and produce systematic evaluations of the trade-offs and performance; second, we will produce practical implementations of the chosen solutions on our ATM testbed. We feel confident that we have the necessary technical background and resources which are at the cutting edge of the technology to help us achieve our goals.

REFERENCES

[1] Armstrong, S., Freier, A., Marzullo, K., “Multicast Transport Protocol,” RFC 1301, February 1992.

[2] Biersack, E., “Performance Evaluation of Forward Error Correction in ATM Networks,” ACM Sigcomm ‘92, vol. 22, No. 4, pp. 248-258, August 1992.

[3] Brady, P., “Effects of Transmission Delay on Conversational Behavior on Echo-free Telephone Circuits,” Bell System Technical Journal, Vol. 50, No. 1, pp.115 - 134, January 1971.

[4] Braudes, R., Zabele, S., “Requirements for Multicast Protocols,” RFC 1458, May 1993.

[5] Chang, J., Maxemchuck, N., “Reliable Broadcast Protocols,”, ACM transactions on Computer Systems, Vol. 2, No. 3, pp.251 - 275. August 1984.

[6] Cheriton, D., “VMTP: Versatile Message Transaction Protocol,” RFC 1045, Stanford University, Feb. 1988.

[7] Clark, D., Lambert, M., Zhang, L., “NETBLT: A Bulk Data Transfer Protocol,” RFC 998, MIT, 1987.

[8] Crowcroft, J., Paliwoda, K., “A Multicast Transport Protocol,” Proc. ACM Sigcomm ‘88, 1988.

[9] Danzig, P, “Flow Control for Limited Buffer Multicast,” IEEE Transactions on Software Engineering, Vol. 20, No. 1, pp. 1-12, Jan 1994.

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

[11] Dempsey, B., Liebeherr, J., and Weaver, A., “On Retransmission-based Error Control for Continuous Media Traffic in Packet-switching Networks,” Technical Report CS-94-09, Computer Science Department, University of Virginia, Charlottesville, Virginia, Feb. 1994.

[12] 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.

[13] 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.

[14] Garrett, M., Willinger, W., “Analysis, Modeling and Generation of Self-similar VBR Video Traffic,” SIGCOMM ‘95, pp. 269 - 280.

[15] Gong, F., Parulkar, G., An Application-oriented Error Control Scheme for a High-Speed Transport Protocol,” Technical Report WUCS-92-37, Washington University, 1992.

[16] Gopalakrishnan, R., Parulkar, G., “Efficient Quality of Service Support in Multimedia Computer Operating Systems,” Technical Report WUCS-94-26, Washington University, Nov 1994.

[17] Jacobson, V., Multimedia Conferencing on the Internet,” Tutorial notes, Sigcomm ‘94.

[18] Kaashoek, M., Tanenbaum, A., Hummel, and Bal, “An Efficient Reliable Broadcast Protocol”, Operating systems review, October, 1989.

[19] Netravali, A., Roome, W., Sabnani, K., “Design and Implementation of a High-Speed Transport Protocol, “IEEE Transactions on Communications, Nov. 1990.

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

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

[22] Postel, J., “Transmission Control Protocol - Darpa internet Protocol Program Specification.” RFC 793, September, 1981.

[23] Shulzrinne, H., Casner, S., Frederick, R., Jacobson, V., “RTP: a Transport Protocol for Real-Time Applications,” Internet Draft draft-ietf-avt-rtp-07, work in progress, March 1995.

[24] Sunderam, V., “PVM: A Framework for Parallel Distributed Computing,” Concurrency: Practice and Experience, Vol. 2, No. 4, pp. 315-339, December 1990.

[25] Turner, J., “Managing Bandwidth in ATM Networks with Bursty Traffic,” IEEE Network, pp. 50 - 58, September 1992.

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

[27] Yavatkar, R., Manoj, L., “Optimistic Strategies for Large Scale Dissemination of Multimedia Information,” Proceedings of ACM Multimedia ‘93, Anaheim, CA, pp. 1-8, Aug. 1993.

[28] Whetten, B., Kaplan, S., Montgomery, T., “A High Performance Totally Ordered Multicast Protocol”, INFOCOM ‘95.