Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation

Hugh W. Holbrook* Sandeep K. Singhal David R. Cheriton

Department of Computer Science
Stanford University

Abstract

Reliable multicast communication is important in large-scale distributed applications. For example, reliable multicast is used to transmit terrain and environmental updates in distributed simulations. To date, proposed protocols have not supported these applications' requirements, which include wide-area data distribution, low-latency packet loss detection and recovery, and minimal data and management overhead within fine-grained multicast groups, each containing a single data source.

In this paper, we introduce the notion of *Log-Based Receiver-reliable Multicast* (LBRM) communication, and we describe and evaluate a collection of log-based receiver reliable multicast optimizations that provide an efficient, scalable protocol for high-performance simulation applications. We argue that these techniques provide value to a broader range of applications and that the receiver-reliable model is an appropriate one for communication in general.

1 Introduction

Multicast sources in certain distributed applications have relatively low update rates yet have receivers that expect low delay in receiving updates, even in the face of loss. For instance, the dynamic terrain in a distributed virtual reality system such as *Distributed Interactive Simulation* (DIS) [11, 19] is normally updated infrequently, yet hosts need to be notified of updates within a fraction of a second or less to avoid perceptible skew among the views of participants. In DIS, terrain includes natural entities such as hills and trees, as well as cultural artifacts like bridges and buildings. Consider the example of a bridge that is destroyed during a virtual military exercise. The bridge is completely static for some considerable length of time, but once it is destroyed, each tank within visual range should "see" the bridge as destroyed shortly afterward. A tank with stale information about the bridge might try to drive over it, for instance.

*[holbrook,singhal,cheriton]@cs.Stanford.EDU

Dynamic terrain in DIS is a specific case of the distributed cache update problem. The cached data may not change for long periods of time, yet when it does, all cached copies need to be notified quickly.

We use the term *freshness* to refer to the degree that a receiver's state is up to date with the source. We are interested in providing reliable multicast to applications where the update rate is low, but a high degree of freshness is required even if packets are lost. For example, the tank needs an up-to-date view of the bridge, even though the bridge may not change frequently.

High-frequency transmission of state updates guarantees freshness in real-time protocols like *vat* [16]. However, it is not feasible to require DIS terrain entities to send frequent updates because of the excessive network traffic that would result. For example, next generation DIS systems may include hundreds of thousands of terrain objects. Similar scale may arise with entertainment applications over the Internet. If each entity generated a packet every 0.25 seconds, the network would be unnecessarily congested.

In many internetwork configurations, congestion occurs because of the limited bandwidth available on the *tail circuits* connecting individual sites to the backbone network, as illustrated in Figure 1. These tail circuits are likely to remain bottlenecks for the foreseeable future because they are expensive like the backbone lines (and unlike the local site lines), but unlike the backbone lines, they are only cost-amortized over a single site, or a small number of sites.

In current DIS, most of the bandwidth is consumed by periodic *appearance PDUs* that apply the real-time approach of continual state transmission for highly active entities such as airplanes, tanks, and jeeps. *Dead*

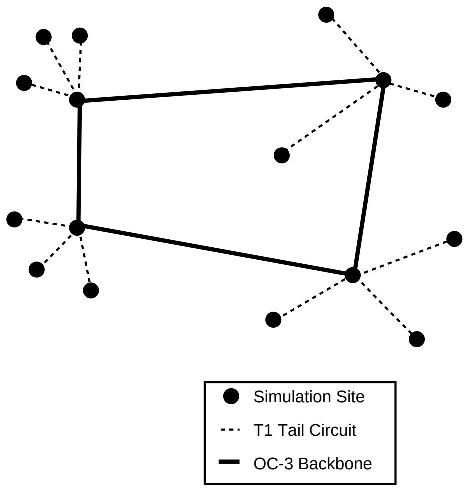

Figure 1: Typical WAN Topology, Showing the Tail-Circuit Bottleneck

reckoning [17] at each receiver dramatically reduces the bandwidth demands of dynamic entities, but the naturally high update rate of these entities still requires a large amount of communication.

Positive acknowledgement protocols such as the Chang and Maxemchuk algorithm [5] are unsuitable for the type of multicast traffic we have in mind for several reasons. First, a positive acknowledgement scheme used with multicast can lead to an acknowledgment implosion at the source and significant network load. Second, positive acknowledgement requires that the source know the identity of the receivers or subscribers. In many of the cases of interest, the source does not know about all of the receivers. Finally, positive acknowledgement protocols delay reception of subsequent packets until a missing packet has been recovered. This approach conflicts with the real-time requirement of favoring immediate reception of the latest data over waiting for retransmission of an earlier missing packet.

In this paper, we describe our *Log-Based Receiver-reliable Multicast (LBRM)* protocol that provides scalable, timely dissemination of state updates, meeting the needs of multicast sources like DIS terrain entities. We also describe several optimizations to the basic LBRM approach that further reduce the network load, the expected delay to recover a lost packet, and the load on receivers. In discussing related work, we contrast this protocol with the highly successful *wb* multicast session protocol [8] and argue that *wb* is inadequate for the type of large-scale applications that we are considering.

The next section describes the Log-Based Receiver-reliable multicast protocol and three optimizations that improve the protocol's performance. Section 3 presents our preliminary experience with an implementation of the protocol. Section 4 describes applications of the receiver-reliable multicast techniques. Section 5 compares sender-reliable and receiver-reliable techniques, and Section 6 compares and contrasts our techniques with previous work in the area. We conclude with a summary of the LBRM approach and directions for future research.

2 Log-based Receiver-Reliability

In the basic receiver-reliable multicast protocol, the source includes a sequence number in each packet and defines a *Maximum Idle Time* (MaxIT) bound. The source guarantees that it will transmit a packet at least once every MaxIT interval. If the application provides no data to send within MaxIT, the protocol generates keep-alive or *heartbeat* packets that repeat the previous sequence number (but not the associated data). A receiver recognizes that it has lost a packet either when it detects a gap in the sequence numbers of received packets, or when it has not received a packet for MaxIT.

The application chooses MaxIT according to the freshness requirement of the data being disseminated. Shortening MaxIT results in fresher data, but more network traffic. For entities with strict real-time delivery requirements, MaxIT must be small. For DIS terrain entities that change infrequently, recent research [3] suggests that a 1/4 second MaxIT is required to provide acceptable real-time visual performance.

In the *Log-Based Receiver-reliable Multicast* (LBRM) approach, illustrated in Figure 2, reliability is provided by a *logging server* that logs all transmitted packets from the source. When a receiver detects a lost packet,

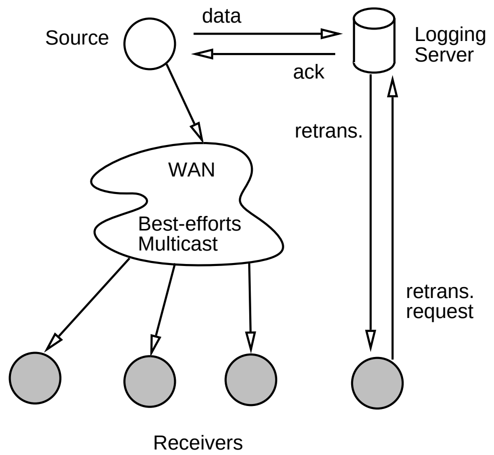

Figure 2: Log-based receiver-reliability

it requests the missing packet from the logging server.

The logging server need not be co-located with the source host, but if the two are separated, then the source must retain the data until it has received a positive acknowledgement from the logging server. The logging server may be replicated to provide greater reliability, as discussed in Section 2.2.3.

The use of a logging server for reliability generalizes the buffering of outstanding data performed by the sender in a conventional transport protocol. In TCP, the buffered data at the sender is effectively a log of transmissions, from which acknowledged packets have been flushed.

The protocol is *receiver*-reliable, in that each receiving application defines its own reliability requirements. The sender merely makes it possible, via the logging service, for the receiver to recover a lost packet.[1] Receivers are not obligated by the protocol to retrieve every lost packet, and the sender does not check to confirm that each receiver has every packet. Moreover, message causality and ordering are strictly an application-level concern for the receiver [6].

The length of time that the logging server must store a packet is application-specific. Some applications may only store packets until their useful lifetime has expired. Other applications with stronger persistence needs may log all packets, writing them to disk once in-memory buffers are full.

In the LBRM approach, three resources must be carefully managed: network bandwidth (particularly on the congested tail circuits), CPU load on the receiving hosts, and server load.

In the following sections, we present three techniques for managing network, receiver, and server load in large-scale, wide-area, distributed simulations using LBRM. First, we discuss a variable heartbeat mechanism that provides fast detection of packet loss while, in the DIS environment, using 1/50 of the heartbeat bandwidth of the basic receiver-reliable approach. Second, we describe a distributed logging service that reduces the number of retransmission requests, resulting in lower network and server load. Last, we describe a statistical loss-detection mechanism that dynamically chooses between multicast and unicast retransmission strategies to improve recovery time and minimize network bandwidth with large numbers of sites.

2.1 Variable Heartbeat for Real-time Delivery

The variable heartbeat scheme clusters heartbeat transmissions in the interval immediately following a data transmission, rather than spreading them out evenly across the idle period between transmissions. Using this technique, we provide sub-second loss detection of isolated packet losses. For longer periods of network lossiness, the loss detection interval is bounded by the length of the lossiness. However, we do not require a high heartbeat rate when the channel is idle.

In our scheme, each sender maintains an inter-heartbeat time h, which is the interval between the previous packet (heartbeat or data) transmission and the next heartbeat transmission. This interval is reset every time the application sends a data packet. Every sender also has a minimum inter-heartbeat time h_{min} and a maximum inter-heartbeat time h_{max}.

When the sender transmits a data packet, it initializes the inter-heartbeat time h to h_{min}. After every subsequent heartbeat packet is sent, the value of h is doubled.[2] The inter-heartbeat interval increases until it reaches h_{max}. When another data packet is sent, h is immediately reset to h_{min}. A sample distribution of heartbeat packets is shown in Figure 3.

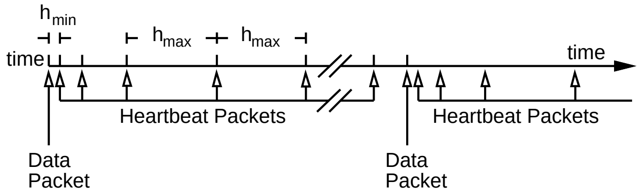

Figure 3: Timeline showing the distribution of heartbeat packets between data packet transmissions

2.1.1 Performance Analysis

The variable heartbeat algorithm provides loss detection within a period of h_{min} after single-packet losses.[3] After longer periods of network loss, it bounds the loss detection time to twice the loss period duration.

Consider a simple "burst" model of congestion, where congestion is parameterized in terms of its duration. Suppose that the network experiences a burst congestion period of duration t_{burst} during which a given host receives no packets. Assume further that before and after the burst loss period, the host receives 100% of the sender's transmissions.

[1] The receiver can estimate how much information it has lost either based on the number of data packets it is missing or based on the amount of time that has elapsed since the last packet was received.

[2] Note that in general, h could increase by any *backoff* multiple, 2, 3, 5, or by any arbitrary function.

[3] We define the loss detection interval to be the period between when the packet would normally have arrived and when the loss was detected at the receiver. For the purposes of this section, we assume that network delay is independent of load and that the delay between two hosts is invariant. While the effect of network delay is important, it would complicate the discussion without seriously affecting the basic analysis.

Consider a data packet that is sent at the start of the burst loss period. If the burst error length is small (less than h_{min}), (i.e., an isolated loss), then the lost packet is discovered when the first heartbeat packet arrives after h_{min}. If the burst error is longer, then a heartbeat will arrive no longer than t_{burst} after the network returns to normal. In this case, the maximum time between data packet transmission and receiver discovery of packet loss is $2 \times t_{burst}$ [4] (or h_{max}, whichever is smaller). Although h_{max} represents the maximum delay before a receiver learns of a lost data packet, this maximum delay is only encountered under extremely poor network conditions.

Thus, isolated losses and transient errors are discovered quickly and longer burst errors are discovered in time bounded by $\min(t_{burst}, h_{max})$.

2.1.2 Comparison to a Fixed Heartbeat Scheme

The number of packets generated by the variable heartbeat protocol is always less than that of a fixed-heartbeat scheme, when h_{min} is set to the fixed heartbeat interval. When the basic transmission rate is low, as is the case for DIS terrain entities, the variable heartbeat scheme has a significant advantage.

Consider a DIS scenario, loosely based on the US military's current simulation plans [2]. The scenario involves 100,000 dynamic entities (tanks, planes, ships, infantry), and an equal number of aggregate terrain entities (rocks, trees, fences, bridges). If each aggregate terrain entity has a real-time update requirement of 1/4 second, then with a fixed heartbeat, each would generate 4 packets per second, for a total of 400,000 packets per second. In current DIS simulations, dynamic entities generate one packet per second, on average [20], which results in 100,000 packets per second. Although terrain entities desire a 1/4 second freshness guarantee, they generally change state infrequently. If we estimate that the state changes once every two minutes, then the periodic heartbeats account for effectively all of the terrain updates and for 4/5 of the simulation's 500,000 packets per second.

Assuming a 1/4 second desired recovery time, Figure 4 compares the variable heartbeat rate to the fixed-heartbeat protocol as a function of the interval between data packets, dt. As dt increases, the variable heartbeat transmission rate approaches $1/h_{max}$, but the fixed-rate protocol continues to approach $1/h_{min}$, unaffected by the increase in dt. If $dt < 1/h_{min}$, no heartbeats are transmitted under either scheme, as every heartbeat packet is preempted by the next data packet.

Figure 5 shows the ratio of the heartbeat rates of the two approaches, using the same algorithm parameters as in Figure 4. As the interval between data packets grows large relative to h_{min}, the savings of the variable heartbeat scheme also grows. The marked point on the graph corresponds to a scenario in which the update rate is once every 120 seconds. At this point the variable heartbeat reduces heartbeat bandwidth by a factor of 53.4 over a fixed heartbeat.

[4] If h is increasing by a multiple k other than 2, then the loss recovery time would be $k \cdot t_{burst}$.

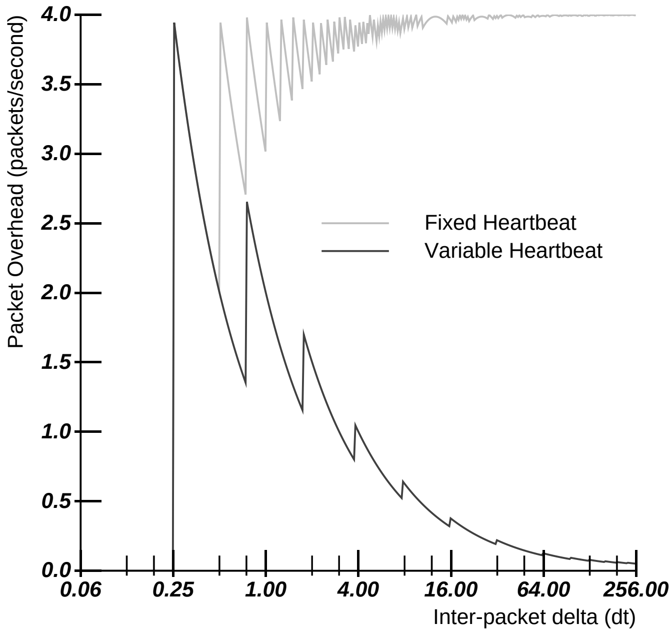

Figure 4: Fixed and Variable Heartbeat Overhead Rates ($h_{min} = 0.25, h_{max} = 32$, backoff $= 2$)

Table 1 shows how this ratio is affected by the *backoff* parameter, keeping all other algorithm parameters constant. Increasing the backoff parameter results in a larger loss detection interval when $t_{burst} > h_{min}$, but it does yield additional bandwidth savings. The reduction in overhead is moderately sensitive to the chosen backoff value.

Backoff	Overhead ($\frac{\text{Fixed}}{\text{Variable}}$)
1.5	34.4
2.0	53.3
2.5	65.8
3.0	74.8
3.5	81.7
4.0	87.3

Table 1: Ratio of Fixed Heartbeat Overhead to Variable Heartbeat Overhead as the Backoff Parameter Changes

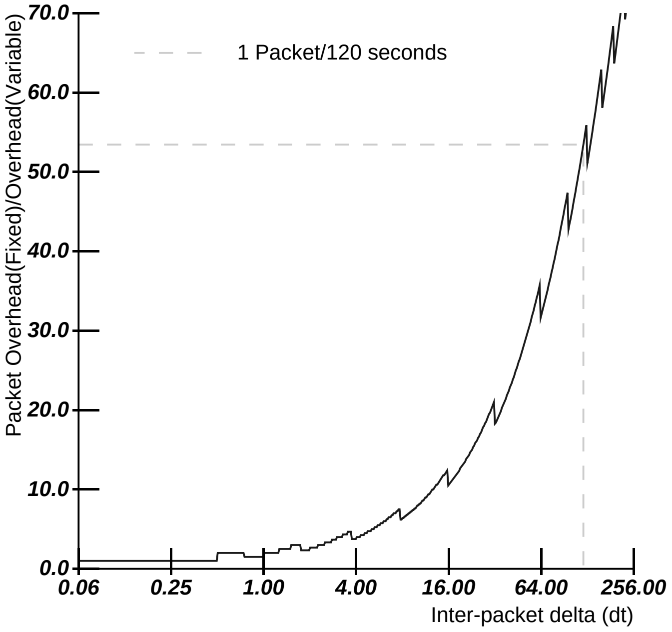

Figure 5: Overhead(Fixed)/Overhead(Variable). ($h_{min} = 0.25, h_{max} = 32$, backoff $= 2$)

2.2 Distributed Logging

As illustrated in Figure 6, *distributed logging* adds logging capabilities at client sites. For our purposes, a *site* is a topologically localized part of the network that is defined by the particular multicast application—it might be a set of hosts on a network tail circuit, a LAN, or even a single host.

2.2.1 Secondary Logging Servers

Each site's logging server logs packets from the multicast source. These logging servers call back to the primary logging server to retrieve a lost packet. Each receiving application requests retransmissions from its local *secondary* logging service, rather than directly from the source's *primary* log service.[5] In this way, only one retransmission request to the primary logging server originates from each site, rather than one from each group member, as illustrated in Figure 7. In cases where a packet is lost within a site, recovery can be handled by the local logging server; in this case, recovery introduces no load on the WAN or tail circuits.

The secondary logging service is purely an optimization over having each node directly access the primary logging server. If the secondary logging service fails, a receiver requests retransmissions directly from the primary (or next-higher-level) logging server. Moreover, the multicast source only receives an acknowledgement from the primary server signaling that the data packet can be discarded; the source is unaware of the secondary services' existence.

[5] When multicast sources are located at many sites, as is the case in DIS, a single logging process may serve as the primary logger for one group and as the secondary logger for another.

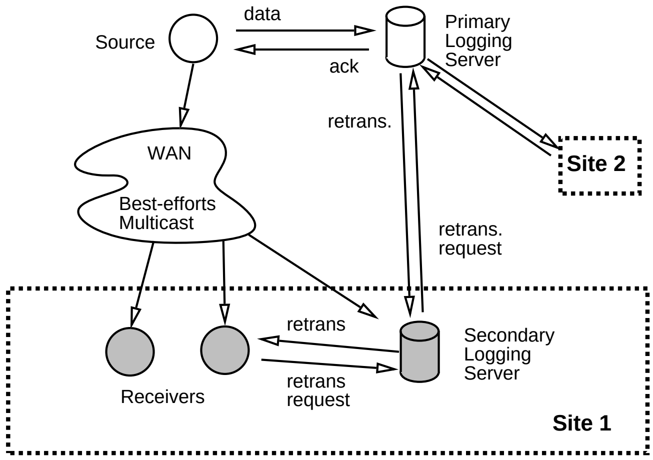

Figure 6: Distributed Logging Service Architecture

A dedicated machine at each site may be specially configured for the logging task in the same way that a file server or a compute server is configured for its task. Such a designated logger might have a fast network connection and a large memory and disk. An alternative implementation could provide distributed logging at each site by rotating the role of log server among the local hosts in order to distribute the load, similar to the Chang and Maxemchuk [5] algorithm, except that the multicast traffic originates from a source outside the virtual ring.

A variety of resource discovery techniques are possible for locating a secondary logging server. In our implementation, each host uses a series of scoped multicast discovery queries to locate a nearby logging service. If no sufficiently close logging service is found, then the receiver may either initiate a secondary logging process locally or take steps to initiate one on another nearby machine. Alternatively, each host may be statically configured with the location of the site's logging servers, much like it is configured with the location of the local name server and time server.

A secondary logging server may decide to re-multicast a packet, rather then sending point-to-point retransmissions, if it decides that a significant number of clients have lost the packet. This decision is based on the number of multicast requests it has received and whether or not the secondary logger itself received the packet. By setting the TTL (time-to-live) field in the retransmis-

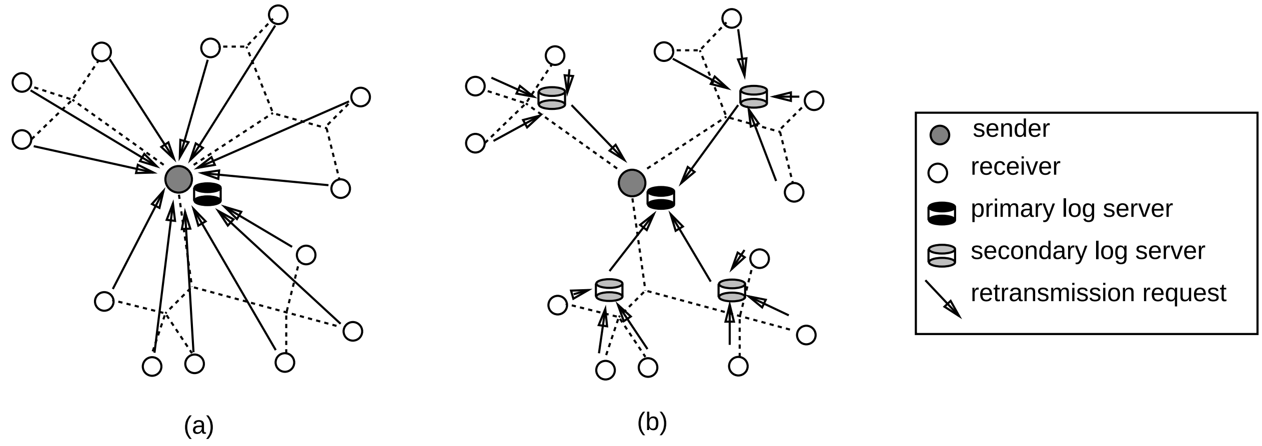

Figure 7: Retransmission Requests Under (a) Centralized Logging and (b) Distributed Logging

sion to an appropriately chosen value, the local server can limit the retransmission's scope to the local site. Within a site, we do not consider NACK implosion to be a problem because each secondary logger is presumed to serve a reasonably small number of receivers and because intra-site links are assumed to be relatively uncongested.

2.2.2 Performance Analysis

Distributed logging reduces NACK and retransmission bandwidth, as well as retransmission latency, when compared to a centralized packet recovery scheme. In order to quantify these effects in the distributed simulation environment, we assume that a particular receiver-reliable terrain update group has at least $1,000$ subscribing entities. (In "hot-spot" areas of the simulation, this number is likely to be even higher, and proposed non-DIS applications of reliable multicast—such as dissemination of stock quotes or traffic reports—have potential audiences in the hundreds of thousands.) The $1,000$ subscribers are distributed across 50 sites with 20 participating receivers at each site. This is a slightly larger number of sites than is currently under consideration for DIS simulations but is slightly smaller than what we envision for other future wide-area dissemination applications.

Distributed logging reduces NACK and retransmission bandwidth when a large fraction of the group loses packets. Figure 1 illustrates a common situation in the DIS network topology where congestion on the incoming bottleneck (T1) tail circuit causes packet loss at an entire site. Distributed logging cuts the number of NACKS transmitted across the tail circuit and the WAN from 20 (one per receiver at the site) to 1 (from the site's secondary logging server). The reduction in NACK requests proportionately reduces the load upon the primary logging server. Section 2.3 discusses a NACK reduction technique designed for groups containing many subscribing sites.

Distributed logging also reduces retransmission latency for packets that are lost within a site, because the round-trip time to the secondary logging server is much shorter than that to the primary logging server. This effect is becoming increasingly critical as DIS sites become more widely distributed. Some simple experimentation with the *ping* program reveals that a secondary logging server that is a few miles away and through a small number of gateways might typically be at a distance of 3-4 milliseconds RTT from the source, while a primary logging server located $1,500$ miles away across 10 internetworking routers might be at a distance of 80 milliseconds RTT. By getting a retransmission from the local logging server, we can reduce the retransmission latency by an order of magnitude.

LAN bandwidth and local resources are expected to remain relatively plentiful and cheap, while tail circuit (LAN-WAN) connections are likely to become more costly and/or more congested in the future. Distributed logging recognizes and exploits this trend.

2.2.3 Recovery from Primary Log Failure

If the primary logging server fails, receivers may not be able to recover all lost messages. To provide greater fault tolerance for these situations, we replicate the primary logging server. As before, the source reliably transmits packets to the primary logging server, who acknowledges their receipt. The primary logging server, in turn, is responsible for reliably transmitting updates to the replicated servers.

When the primary logger acknowledges a packet to the source, it includes two sequence numbers: a primary logger sequence number and a replicated logger sequence number. When an acknowledgement arrives from the primary logger, the sender's application may continue processing. However, in order to guarantee fault-tolerance, it cannot discard the data until it knows that a replica has received the packet.

When a primary logger failure occurs, the source locates the logging server replica holding the most up-to-date packets—that is, the replica associated with the most recent replicated logger sequence number. The source reliably transmits to the replica any packets being held in its buffer. From that point, the replica is guaranteed to hold a complete packet history and can proceed as the new primary logging server. Complete log failure therefore can only occur when two logging servers (the primary and the most up-to-date replica) fail simultaneously.

To support failure recovery, each secondary logger and each receiver that directly accesses the primary logger treats the primary logger address as a cached value. Whenever the primary logger becomes inaccessible, the receivers contact the source who can furnish the identity of the current primary logger.

2.3 Statistical Acknowledgement for Selecting a Retransmission Strategy

Statistical acknowledgement is used to select between multicast retransmission and possibly multiple unicast transmissions when packets are lost. When a packet is lost by a number of sites, multicasting the retransmission immediately uses less network bandwidth and results in faster delivery time than servicing individual retransmission requests via unicast. However, at the other extreme, if a packet is lost at a single site, it is more efficient to unicast to that single site and not load the network and other sites with a multicast retransmission.

With statistical acknowledgement, the source *probabilistically selects* a small random set of secondary logging servers to acknowledge each transmitted packet (by a unicast packet sent back to the source). The source chooses between multicast retransmission and unicast transmission based on the number of acknowledgements it receives from this set. The source selects a new set of acknowledging log servers periodically, i.e. every new *epoch*. The following subsections elaborate.

2.3.1 Epochs

The multicast transmission is divided into *epochs*. Before the start of each epoch, the source chooses a number of positive acknowledgements (k) that are desired for each data packet, where k is a typically small number relative to the total number of sites; analysis suggests that between 5 and 20 ACKs is appropriate. The source computes a *logger acknowledgement probability* p_{ack} based on an estimate of the number of active secondary loggers, N_{sl}. (The next section presents an algorithm for estimating N_{sl}.) The source transmits the newly selected value of p_{ack} and the new epoch number in an *Acker Selection Packet*. Each secondary logger responds to the Acker Selection Packet with probability p_{ack}. The responding loggers are the *Designated Ackers* which then unicast to the source an acknowledgement for each packet of the epoch they receive. Either all data packets include the current epoch number so that Designated Ackers know which packets they must acknowledge or else the source keeps track of the Designated Ackers for an epoch and expects some overlap in acking between epochs.

After receiving responses from each new Designated Acker, the sender knows exactly how many acknowledgements to expect for each data packet in the epoch. The source then switches to the new epoch for newly transmitted data packets.

Figure 8 shows the packets sent at the beginning of a new epoch. An initial Acker Selection Packet is sent and three designated ackers respond. Next, the source sends data packet #33 in the new epoch, but only receives two out of three ACKs. Consequently, the source immediately re-multicasts the packet, and this time receives the expected three ACKs.

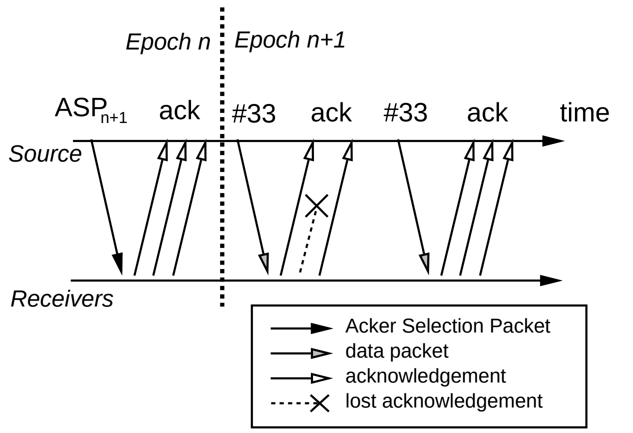

Figure 8: Timeline showing the operation of probabilistic selection and statistical acking.

After the source sends an Acker Selection Packet, it must at some point stop waiting for ACKs from the new Designated Ackers, and start the new epoch. The sender needs to wait for a period that is long enough to allow most secondary loggers to respond. Future ACKs from secondary loggers that do not respond within this interval are not considered. This interval should be long enough to include ACKs from all but the most highly delayed members of the multicast group.

2.3.2 Retransmission Strategy

When a source receives acknowledgements from all its Designated Ackers, it assumes that multicast retransmission is not needed, and waits for individual retransmission requests. The primary logger can resort to a multicast retransmission if it receives a significant number of retransmission requests. For example, if there are 20 Designated Ackers in a configuration with 500 sites, it is possible, although unlikely, to receive all the acknowledgements yet have 480 sites that missed the data.

When a source is missing acknowledgements from one or more of its Designated Ackers, it selects to immediately multicast a retransmission if the number of missing acknowledgements represents a significant number of sites. For example, with a 500 site configuration, each Designated Acker represents 25 sites so multicast is warranted if even a single acknowledgement is lost. However, with a 20 site configuration, it is feasible for each logging server to acknowledge.

This strategy is viewed as an optimization over multicasting every retransmission, which is the safest approach otherwise, given the cost of individual unicasts when packet loss occurs high in the multicast distribution tree. We recognize that multicast retransmissions will occur with this scheme that are not warranted. For example, missing acknowledgements arise when a secondary logger fails, becomes disconnected from the network, or leaves the multicast group. However, such events are rare, and their effects are limited to the current epoch.

Packet buffering and round-trip time estimation use techniques similar to those in TCP and other transport protocols. The source must retain each data packet for an interval t_{wait} after sending before it can determine if the packet should be re-multicast. During this interval, it collects and counts ACKs received from the Designated Ackers. If t_{wait} is too short, the sender may be led to believe that a packet is lost, when in fact its ACKs are merely delayed. If t_{wait} is too long, however, the sender unnecessarily delays the detection of lost packets, increasing the chance of a NACK implosion. An initial value for t_{wait} can be chosen at the start of an epoch based on the value during the previous epoch or based on some prior knowledge. Thereafter, the sender adjusts t_{wait} on every data packet using an exponentially-converging round-trip estimator:

$$t'_{wait} = \alpha \times rtt_{new} + (1 - \alpha) \times t_{wait}.$$

The value of rtt_{new} is chosen to be the time at which the last ACK to a data packet arrives, up to time $2 \times t_{wait}$ after each Acker Selection Packet. The $2 \times t_{wait}$ time limit allows the source to, at some point, assert that an ACK was lost. If t_{wait} is less than the minimum heartbeat time, h_{min}, then a transmission error is discovered before any retransmission requests are sent. If, instead, $t_{wait} > h_{min}$, the secondary logging servers should delay their retransmission requests until the primary logging server has had a chance to re-multicast the packet. This occurs at $t_{wait} - h_{min}$ after the first heartbeat has arrived.

2.3.3 Group Size Estimation

Our basic algorithm for generating an initial secondary logging server count N_{sl} is based a modification of a technique described by Bolot, Turletti, and Wakeman [4]. We build upon their basic protocol in which a sender initiates a series of rounds wherein the secondary loggers are probed, using an increasing ACK probability p_{ack} to avoid causing an ACK implosion on the sender. Once *enough* ACKs have been received to make a fairly confident estimate of N_{sl}, the probing stops. As a modest extension to the Bolot scheme, the last p_{ack} probe may be repeated several times to increase the accuracy of the N_{sl} estimate. Each successive probe increases the accuracy, as shown in Table 2.

Probe Count	Standard Deviation of N_sl Estimate	
1	$\sqrt{\frac{N(1-p_{ack})}{p_{ack}}}$	$= 1.000\sigma_1$
2	$\sigma_1/\sqrt{2}$	$= 0.707\sigma_1$
3	$\sigma_1/\sqrt{3}$	$= 0.577\sigma_1$
4	$\sigma_1/\sqrt{4}$	$= 0.500\sigma_1$
5	$\sigma_1/\sqrt{5}$	$= 0.447\sigma_1$

Table 2: Accuracy of N_{sl} Estimation as Number of Probes Increases for Actual N Secondary Loggers and Acknowledgement Probability p_{ack}

Bolot, Turletti, and Wakeman's protocol demands that the secondary logger estimation algorithm be repeated periodically. However, with the statistical acknowledgement scheme, the N_{sl} estimation need only be run at the beginning of a multicast transmission until the number of secondary loggers has stablized. Afterward, responses to each Acker Selection Packet effectively serve as a secondary logger probe that is used to continually readjust N_{sl}.

The following algorithm dynamically maintains the N_{sl} estimate. Assume roughly k ACKs are expected on each Acker Selection Packet, the sender has an initial group size estimate of N_{sl} loggers, and the Acker Selection Packet carries an ACK probability of p_{ack} =

k/N_{sl}. After transmitting a data packet, the source receives k' ACKs. Using the following formula, the source revises its N_{sl} estimate:

$$N'_{sl} = (1 - \alpha) \times N_{sl} + \alpha \times k'/p_{ack}$$

where α is some small number, say 1/8. The new ACK probability is $p'_{ack} = k/N'_{sl}$. This approach is similar to Jacobson's TCP Round-Trip-Time estimator [12]. Statistical variations in k' cause minimal variation in N_{sl} or p_{ack}, but the algorithm dynamically adjusts as secondary loggers enter and leave the group.

Due to software or hardware faults, a logger might disrupt the system by, for example, responding to every Acker Selection Packet. The source can easily track these faults by keeping a histogram or a timed "hotlist" of recently-active Designated Ackers. Once a faulty logger has been identified, its future ACKs can be ignored.

2.3.4 Scalability Considerations

Statistical acknowledgement quickly detects widespread packet loss and retransmits the lost data within one round-trip time. In most cases, this will prevent every logging server from simultaneously requesting retransmissions from the sender. For future DIS networks that encompass sites in the US and in Europe, 50 sites is a conservative estimate. Without these techniques, the potential for NACK implosion in networks of this size is severe. Statistical acknowledgement prevents the implosion in the case where isolated packets are lost on the sender's outgoing tail circuit due to one-time transmission errors or due to short-term congestion.

3 Implementation Experience

We have implemented the Log-Based Receiver-reliable Multicast protocol with the variable heartbeat and distributed logging extensions in the Unix environment. The scoped multicast technique discussed in Section 2.2.1 is used for logging server discovery. Our implementation does not yet implement statistical acknowledgment or deal with logging server failures.

The loggers are stand-alone Unix processes. The sender and receiver protocols are C++ class libraries linked into the LBRM application. The entire implementation is 4,867 lines of code. Much of the code is reusable across different components of the system because of the recursive nature of the distributed logging architecture. Our implementation experience indicates that the overall design complexity is modest.

We measured the response time and CPU load of a local secondary logger responding to logging requests. All measurements were made on an IBM RS/6000 model 370, rated at 70 integer SPEC marks, running AIX 3.2.5, with a 10 Mbit Ethernet adapter. The involved CPUs and the network were otherwise unloaded.

Table 3 shows measurements of the response time to request and retrieve a 128-byte packet from a logging server located on the local Ethernet. This is effectively the cost of an RPC to the logging server. This measurement includes the effects of server processing, network transmission over the 10 Mbit Ethernet, and network interrupt handling and context switching. These measurements suggest that loss detection (i.e. the 250 msec heartbeat packet) and network transmission (for requests to remote servers), rather than server processing, dominate the packet loss and recovery latency.

Operation	Time (μsecs)
Server Request Processing	102
Ethernet Transmission	390
Network Interrupts, Context Switch, Misc.	1090
Total	1582

Table 3: Secondary Logging Server Response Time

We also measured the maximum rate at which a logging server could respond to retransmission requests without dropping any. A server can receive, process, and reply to one request every 630 microseconds, or 1587 requests per second. The per-packet servicing time is composed primarily of server processing time and OS time spent servicing each network interrupt. No process-to-process context switch occurs if the server is saturated because the server is constantly running on the CPU. These measurements indicate that a server with hundreds of clients is not unduly loaded. The server can receive and process 100 requests for in-memory packets in 0.063 seconds. Our investigation indicates that logging server load is not severe, even if hundreds of nearly simultaneous retransmission requests arrive.

4 Applications of LBRM

The scalability and fast loss recovery of the LBRM protocol make it well-suited to applications in which each host actively disseminates time-sensitive information and each receiver is interested in a subset of sources. In this section, we describe some of these applications, besides Distributed Interactive Simulation, including one that we have implemented in a preliminary fashion.

4.1 Traffic Report and Stock Quote Dissemination

Reliable multicast is well-suited for applications in which clients obtain and cache data from a server. Whenever the server updates the data, it needs to reliably invalidate the caches held by the clients and possibly refresh those caches with updated information. Examples of such information dissemination applications arise for distributing real-time stock quotes to brokers' terminals (and eventually to the public at large) and providing up-to-date traffic report maps to a client display mounted in an automobile dashboard.

4.2 File Caching

LBRM is an alternative to *leasing* [9] for fault-tolerant distributed file caching. Rather than having explicit leases on the files in its cache, each client subscribes to a LBRM channel from the server on which to (reliably) receive invalidation notifications. If the client detects a failure of its connection to the server (by the absence of heartbeats or other traffic), it invalidates its cache; this action occurs in time comparable to a lease timeout. LBRM eliminates much of the bookkeeping and maintenance of timeouts required in the leasing approach and instead relies on the reliability and failure notification of the LBRM server channel, one per fileserver.

4.3 Cached WWW Page Invalidation

World-Wide-Web browsers, such as Mosaic™, keep a cache of recently visited HTML pages, but those caches are not automatically invalidated or refreshed when the source (server) document changes. HTML 3.0 supports dynamically changing HTML pages by allowing clients to periodically poll servers for updates or by maintaining an open TCP connection between the server and each client. However, these approaches to page invalidation offer limited scalability because they place a heavy load on both the server and the network.

To demonstrate the applicability of the LBRM algorithm in this environment, we put a version of the protocol into the Mosaic browser. Each HTML file is associated with a multicast address. When the Mosaic client displays this HTML file, it subscribes to the associated multicast address. Whenever the HTTP server detects that one of the local HTML documents has been modified, it reliably transmits an update announcement to the associated multicast group. In response to the invalidation message, the client highlights the `RELOAD` button, informing the user to reload the page from the server. A simple extension allows automatic dissemination of the updated document to the multicast group. Appendix A describes the implementation in more detail.

4.4 Factory Automation

In a factory automation system, sensors on the factory floor capture the status of equipment. Data from these sensors must be transmitted reliably to the many systems that monitor factory performance, maintenance, etc. LBRM is particularly well-suited to this application domain [10]:

Factory automation typically requires that all transactions and tasks are logged for accurate record-keeping. LBRM already provides this logging as part of the lost packet recovery mechanism. Moreover, LBRM supports simple data sensors because it imposes minimal buffering and computation requirements on those sources.

Factory automation systems must also allow users to dynamically reconfigure the system to change how data flows throughout it. Receiver-reliable multicast eliminates receiver lists froom source hosts and thereby allows new components to be introduced dynamically into the system without the need for an explicit connection setup phase.

Finally, these factory systems are making increased use of mobile devices to allow workers to monitor production from the factory floor. As a rule, these devices experience intermittent network connectivity, yet they still need to receive data reliably. LBRM is well-suited for this environment. When a mobile host reconnects, it can recover any lost data from a logging server without interfering with the other receivers or affecting the on-going data flow from the source.

5 Receiver-Reliable Versus Sender-Reliable Communication

Receiver-reliable communication is better suited for real-time multicast applications than conventional sender-reliable multicast for several reasons. A *receiver-reliable* protocol allows a receiver to know whether or not it has received all packets sent before some relatively recent time t, and it provides a means to recover missing packets; the application chooses whether or not to get these packets. It neither guarantees that packets are delivered to all receivers nor guarantees that they are delivered in-order. Imposing these properties would incur delay, even if the application does not want it.

Receiver-reliability is frequently appropriate when multicast communication is used for notification or dissemination of information. In these dissemination-oriented systems, the server multicasts information to a set of clients asynchronously, rather than responding synchronously to requests from the clients. The sender makes the information available, but only the receiver is really concerned with getting the data. By analogy, television is such a dissemination model. The television station does not need or want positive confirmation from each receiver. Rather, the receiver is responsible for detecting adequate reception and "complaining" if it is inadequate. Many multicast applications have a similar nature. The receiver needs to know whether the information it has is up-to-date (or at least sufficiently recent to satisfy its application semantics).

Conventional transport protocols provide *sender-reliable* communication in which the sender knows that the receivers have received the transmitted data, at least up to some time in the recent past. The "knowledge" is provided by positive acknowledgements sent from the receiver to the sender, confirming receipt of the data.

However, the receipt of a positive acknowledgment from the transport level of the remote client does not guarantee end-to-end reliability. Application-level reliability is provided only by an acknowledgement that was generated after the data was received and processed at the receiver's application level [15]. In a remote procedure call (RPC) system, this end-to-end requirement is provided by receipt of the return packet for the call.

Acknowledgments in conventional transport protocols are really resource management actions to assist the sender in managing its resources and processing. In particular, an acknowledgement primarily signals that the sender's transport module can reclaim the buffer space holding the data for potential retransmission. The timing of the acknowledgement, along with other information in the packet such as the flow control window size, help the sender regulate its rate of transmission to avoid wasting processing time and network bandwidth by sending packets too quickly or too slowly.

As part of our future work, we are exploring the use of the selective acking mechanism as a resource management tool; in particular, we are looking into using statistical acknowledgement information to slow down the sender during periods of high loss.

6 Related Work

Several researchers have proposed reliable multicast protocols, but most of these are not well-suited for large-scale information dissemination applications. Cheriton and Zwaenepoel's [7] k-reliable multicast simply generalizes the conventional *all-reliable* semantics that are usually implemented using *sender-reliable* techniques. Chang and Maxemchuk [5] describe several techniques to reduce acknowledgements at the cost of increased delay in lost packet detection. Causally and totally ordered multicast protocols such as those in Isis [14] and MTP [1] introduce additional latency to enforce ordering of updates across multiple sources, so they are unsuitable for use in real-time applications such as DIS.

Our work is closest to that of Floyd, Jacobson, Liu, McCanne and Zhang [13, 8] on lightweight multicast sessions and the *wb* reliable multicast protocol. The *wb* protocol was initially designed to support a shared whiteboard application.

A major difference between LBRM and *wb* lies in their approaches to packet recovery for large multicast groups. LBRM takes an organized approach to recovery, while *wb* is fundamentally unorganized. LBRM organizes retransmission requests into a hierarchy, by having a receiver request retransmissions only from ancestors in the logging hierarchy. *Wb* imposes no such structure; a receiver requests lost packets from everyone in the group, and anyone with the packet may respond. Consequently, the *wb* recovery algorithm is highly fault tolerant: if any reachable member of the group has the lost packet, then the requestor eventually gets it. However, this fault tolerance comes at the cost of redundant retransmission multicasts. Whenever a host loses a packet, at least one repair request is multicast to the entire group, even if the losses are limited to hosts within a single site. Moreover, between one and five responses are typically multicast to the entire group for each repair request.

A problem with *wb*-style recovery is that it introduces the "crying baby" problem in the case of a single packet drop. That is, if a single link to one member of the group has a high error rate, then all members of the multicast group must contend with a multicast request and one or more multicast responses. For example, a receiver may be behind a poor wireless connection or a congested SLIP line or have faulty network hardware. LBRM does not suffer from the crying baby problem because retransmission requests and repairs are not multicast unless a number of receivers lost the packet.

LBRM improves recovery time compared with *wb* by organizing packet recovery into a hierarchy. After a loss is detected, an LBRM receiver immediately requests a packet from its local logging server. The total recovery delay equals the RTT to the nearest logger in the hierarchy that has the packet, plus processing delays at the loggers. In *wb*, because repairs are multicast to the entire group, a receiver must delay its retransmission request for a time proportional to the RTT delay to the source (in order to avoid duplicate requests), even if the loss occurred locally. Responders also delay their responses to avoid duplicate responses. In *wb*, the last receiver to lose a packet recovers from a loss in approximately $3 \times RTT$ (where RTT measures the network round trip time between the receiver and the packet source).[6]

Organizing the packet recovery in LBRM comes at

[6] It should be possible to improve *wb*'s recovery time for local losses by adding locally-scoped multicast retransmission requests.

some cost in complexity. A logger process must be maintained at every group member's site, and added protocol complexity is required to handle logger failure. There is also an additional protocol for the sender to communicate with the primary logger. A *wb*-style reliable multicast group does not require these mechanisms. However, based on our implementation experience, the protocol complexity is not significant in practice.

In a group with a low data rate, *wb* does not provide fast loss detection, but rather, it relies on periodic multicast session messages occurring at fixed intervals to discover losses—similar to the basic "fixed heartbeat" scheme discussed in Section 2. However, *wb* could be extended to use our variable heartbeat technique for session messages, providing comparable loss detection performance.

7 Conclusion

Our Log-Based Receiver-reliable Multicast (LBRM) protocol supports large-scale information dissemination in applications like DIS that demand low-latency data recovery. We have described three techniques that benefit real-time dissemination applications:

- A variable heartbeat rate bounds the time to detect a lost packet, and introducees a limited overhead. The technique reduces heartbeat packet overhead by roughly a factor of 50 over a fixed heartbeat scheme for an expected DIS scenario.

- Distributed logging provides fast recovery of lost packets and provides a scalable solution to the NACK implosion problem. For example, in the expected DIS scenario, NACK traffic at the primary logging server is reduced from 20 per site to 1 per site. Lost packets are recovered in one RTT from the nearest logging server in the hierarchy that received the packet.

- Statistical acknowledgements allow the sender to dynamically determine whether packet losses are best served by immediately re-multicasting the data or by unicast retransmission by the primary logging server. The statistical technique minimizes recovery delay without imposing significant bandwidth overhead.

The receiver-reliable communication design allows the source to release buffering resources as soon as data is acknowledged by the primary logging server; consequently, the source is isolated from local network behavior at the receivers. The protocol provides meaningful end-to-end reliability semantics in which the receiving host can determine with confidence whether all data has arrived.

Although our protocol is targeted primarily at high-performance simulation applications, we have identified several other potential application areas for our work, including stock quote dissemination to brokers and the public, traffic reports to on-board displays in automobiles, and factory automation and monitoring. These applications benefit from LBRM's built-in support for data logging, dynamic reconfiguration, and intermittent connectivity.

We are considering several promising directions for further research:

- A separate multicast channel could be used for retransmissions. The sender would retransmit every packet on the retransmission channel n times, using an exponential backoff scheme similar to that used for heartbeat packets. A client would recover a lost transmission by subscribing to the retransmission channel, rather than requesting the packet. Logging servers would provide retransmissions of packets that were no longer being transmitted on the retransmission channel. In order to use this technique for real-time data, fast multicast group subscription would be required.

- A multi-level hierarchy of logging servers may be used to further reduce NACK bandwidth in large groups.

- For small packets, it might be cost-effective to retransmit the original packet instead of an empty heartbeat packet. This would reduce retransmission requests.

- The use of LBRM for WWW page caching and for file caching are specific examples of a more general use of LBRM for cache invalidation of all kinds; we are investigating further generalization of this use of LBRM.

- Our work on the HTML browser could be extended to use the full set of LBRM optimizations. LBRM could be implemented as a dynamic protocol module in an extensible browser such as HotJava [18].

In general, there is a need for multicast protocols that allow receivers to determine how much loss they are experiencing, recover missing data if they desire, and recover the data within reasonable delay bounds relative to the time of original transmission. As the bandwidth available for multicast in the wide area grows and as more DIS-like multicast applications are deployed, we expect to see an increasing need for receiver-reliable protocols, such as the LBRM protocol we have presented.

Acknowledgements

The authors were supported by ARPA under contract DABT63-91-K-0001 and by an equipment grant from IBM. Hugh Holbrook was supported by a USAF Graduate Laboratory Fellowship. Sandeep Singhal was supported by a Fannie and John Hertz Foundation Fellowship. Matt Zelesko and Jonathan Stone provided helpful discussions that contributed to the development of the ideas in this paper.

A Appendix: HTML Document Invalidation Protocol

This appendix describes the implementation of our HTML page invalidation protocol mentioned in Section 4.3.

Each HTML file includes a comment in the first line to associate a multicast address for invalidation messages. For example, the line

```
<!MULTICAST.234.12.29.72.>
```

associates the file with multicast address 234.12.29.72. Typically, all files from a single server would share the same multicast address, though this is not a requirement. When the Mosaic client first displays this HTML file, it enters a subscription to the multicast address. This subscription is retained as long as the HTML file remains in the client's cache.

Invalidation messages from the HTTP server take the following form:

```
TRANS:17.0:UPDATE:
http://www-DSG.Stanford.EDU/groupMembers.html
```

This packet, which is the initial transmission of sequence number 17 to this multicast group, invalidates the file `groupMembers.html` in client caches. When updates are not being transmitted, the server transmits Heartbeat packets to the multicast group. For example, the packet

```
TRANS:17.12:HEARTBEAT
```

is the 12th heartbeat transmitted after update sequence 17.

When an update packet arrives, the client sets an invalidation flag for the associated cached page. This flag determines whether to highlight the RELOAD button for the current displayed page. The flag is cleared when the document has been reloaded from the server.

Whenever the client detects that one or more updates were lost, it starts a short retransmission request timer. This delay allows out-of-order packets to arrive, and it prevents NACK implosion at the source. After that timer expires, the client requests any missing updates from a logging process at the server host. The logger's response packet contains a list of retransmissions. For example, a retransmission of update 17 would contain the tag `RETRANS` instead of `TRANS`.

References

[1] Susan M. Armstrong, Alan O. Freier, and Keith A. Marzullo. "Multicast transport protocol." In *Internet Requests for Comments (RFC 1301)*, February 1992.

[2] ARPA. "Stow 97 program plan.", May 1994.

[3] Kevin W. Arthur, Kellogg S. Booth, and Colin Ware. "Evaluating 3d task performance for fish tank virtual worlds." *ACM Transactions on Information Systems*, 11(3):239–265, July 1993.

[4] Jean-Chrysostome Bolot, Thierry Turletti, and Ian Wakeman. "Scalable feedback control for multicast video distribution in the internet." In *Proceedings of SIGCOMM 1994*, pages 139–146, London, England, August 1994. ACM SIGCOMM.

[5] Jo-Mei Chang and N. F. Maxemchuk. "Reliable broadcast protocols." *ACM Transactions on Computer Systems*, 2(3):251–273, August 1984.

[6] David R. Cheriton and Dale Skeen. "Understanding the limitations of causally and totally ordered communication." In *Proceedings of the 14th Symposium on Operating Systems Principles*, Asheville, NC, December 1993. ACM SIGOPS.

[7] David R. Cheriton and Willy Zwaenepoel. "Distributed process groups in the v kernel." *ACM Transactions on Computer Systems*, 3(2):77–107, May 1985.

[8] Sally Floyd, Van Jacobson, Charley Liu, Steven McCanne, and Lixia Zhang. "A reliable multicast framework for light-weight sessions and application-level framing." In *Proceedings of SIGCOMM 1995*, Boston, MA, August 1995. ACM SIGCOMM.

[9] Carey G. Gray and David R. Cheriton. "Leases: An efficient fault-tolerant mechanism for file cache consistency." In *Proceedings of the 12th Symposium on Operating Systems Principles*, pages 202–210, Litchfield Park, AZ, December 1989. ACM SIGOPS.

[10] Kieran Harty. Personal communication, December 1994. Dr. Harty is Manager of Advanced Technology at Teknekron Software Systems in Palo Alto, CA.

[11] Institute for Simulation and Training. "Standard for distributed interactive simulation—application protocols (version 2.0.4 draft ieee standard)." Technical Report IST–CR–94–50, University of Central Florida, Orlando, Florida, March 1994.

[12] Van Jacobson. "Congestion avoidance and control." In *Proceedings of SIGCOMM 1988*, pages 314–329, Stanford, CA, August 1988. ACM SIGCOMM.

[13] Van Jacobson. *Multimedia Conferencing on the Internet.* SIGCOMM 1994 Tutorial Notes. August 1994.

[14] T. A. Joseph and K. P. Birman. "Reliable broadcast protocols." In S. Mullender, editor, *Distributed Systems*, pages 293–318. ACM Press, Addison-Wesley, 1989.

[15] J. H. Saltzer, D. P. Reed, and D. D. Clark. "End-to-end arguments in system design." *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

[16] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacoboson. "Rtp: A transport protocol for real-time applications." Internet draft, Work in Progress, November 1994.

[17] Sandeep K. Singhal and David R. Cheriton. "Exploiting position history for efficient remote rendering in networked virtual reality." *Presence: Teleoperators and Virtual Environments*, 4(2):169–193, Spring 1995.

[18] Sun Microsystems. "The hotjava™ browser: A white paper." Available from `http://java.sun.com/`., 1995.

[19] Jack A. Thorpe. "The new technology of large scale simulator networking: Implications for mastering the art of warfighting." In *Proceedings of the 9th Interservice/Industry Training System Conference*, pages 157–160, Washington, DC, November-December 1987.

[20] Adam H. Whitlock. "Draft estimate of bandwidth demand for stow-97." Distributed by the Naval Research and Development Center (NRaD), May 1995.