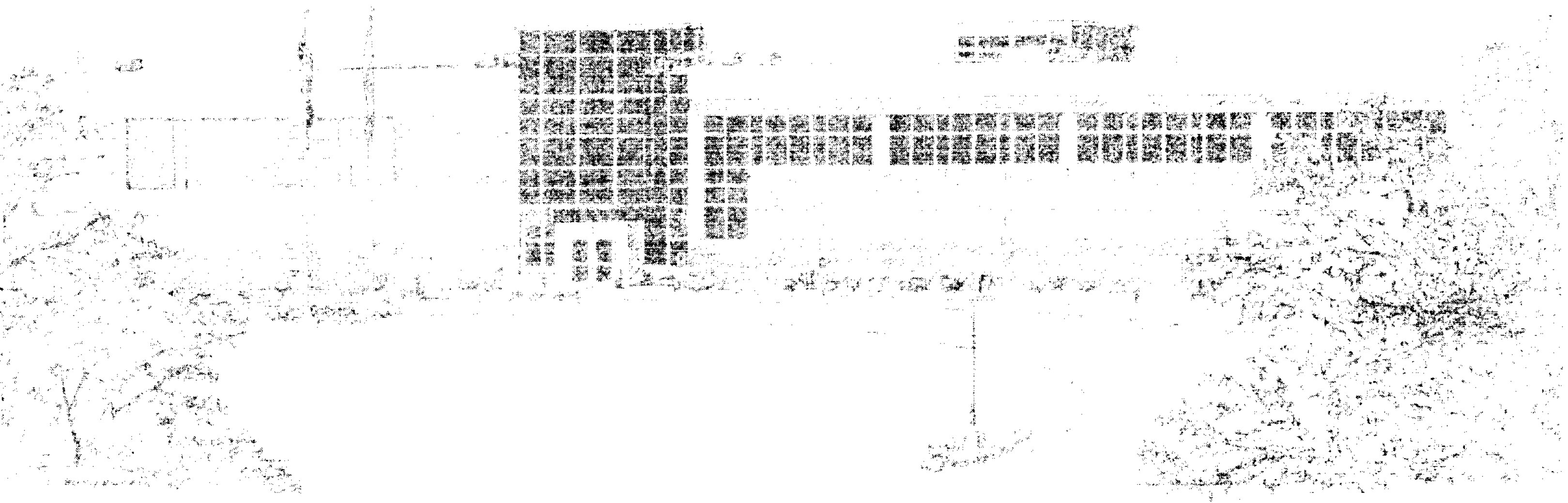
NASA-IVV-97-011
WVU-IVV-97-011
WVU-CS-TR-97-014

NASA/WVU Software IV & V Facility
Software Research Laboratory
Technical Report Series

A Loss Tolerant Rate Controller for Reliable Multicast

By Todd Montgomery



National Aeronautics and Space Administration

West Virginia University

NASA-IVV-97-011

NASA IV&V Facility, Fairmont, West Virginia

A Loss Tolerant Rate Controller for Reliable Multicast

Todd Montgomery

August 20, 1997

This research was funded under cooperative Agreement #NCC 2-979 at the NASA/WVU Software Research Laboratory.

A Loss Tolerant Rate Controller for Reliable Multicast

Todd Montgomery
West Virginia University and GlobalCast Communications, Inc.
tmont@{cs.wvu.edu, gcast.com}

Technical Report: NASA-IVV-97-011
September 8, 1997

Abstract

This paper describes the design, specification, and performance of a Loss Tolerant Rate Controller (LTRC) for use in controlling reliable multicast senders. The purpose of this rate controller is not to adapt to congestion (or loss) on a per loss report basis (such as per received negative acknowledgment), but instead to use loss report information and perceived state to decide more prudent courses of action for both the short and long term. The goal of this controller is to be responsive to congestion, but not overly reactive to spurious independent loss. Performance of the controller is verified through simulation results.

1. Introduction

Reliable multicast protocols face numerous problems when deployed in a large-scale internetwork. One of the principle problems is congestion control. The current Internet must reply primarily on end-to-end congestion control at the protocol layer. Congestion aware router components are becoming more widely deployed, but deployment in a large infrastructure like the Internet is a slow process. For reliable multicast to be accepted and embraced, it must address congestion control.

Congestion control can be implemented two basic ways for reliable multicast. Either the sender controls the rate for the whole group, or the receiver controls the rate at which it receives the data, usually by using multiple multicast groups. The first way, to control the sender, is essentially the technique TCP uses. Loss information, through timeouts, fast retransmits, or selective acknowledgments, causes the sender to slow its sending rate. The primary drawback to scaling this technique up to reliable multicast is that all of the receivers are effected by this rate change. One can argue that this is undesirable given that some receivers should not pay the price for congestion on the path to other receivers. For a loss tolerant data type, such as Audio/Video (A/V), this argument has much merit. However, for bulk-transfer, where the goal is to get 100% of a data object from the sender to a group of receivers, allowing the receivers to receive at different rates brings in a potential semantic problem. At the least, the faster receivers might have to wait for the slower receivers to receive the whole object before the sender can proceed to the next data object. A more complicated scheme may involve allowing different receivers to receive different objects at different rates, while thus disallowing receiver consistency to be taken into account in a scaleable way. For bulk transfer, this may or may not be a serious problem depending on the consistency needs of the application. Further complicating the problem is that certain data types do not lend themselves well to being split among varying groups so that receiving the pieces of the data object at different speeds is possible or desirable[1].

Separate from the issue of the congestion control method, but none-the-less a factor in it, is the method(s) of reliable delivery. As a generalization, the primary mechanisms of achieving reliable delivery can be broken down into two approaches, statistical and decisional. A statistical method involves intentionally sending redundant data that allows receivers to reconstruct the original message if a piece of the data is lost. Statistically, this improves the odds of being able to receive the original data. Forward Error Correction (FEC) is the primary example of this. A decisional method involves the receiver deciding how to receive a lost portion of data. This could be through requesting a retransmission, or by resorting to a third party to fill in the loss. In the presence of no loss, the use of redundancy causes the transmission to be less efficient. In fact, redundancy is only efficient if the redundancy level is close to the amount of loss. If the loss level is above the redundancy level, then the method is ineffective because the original data is not "decodable" from the amount of data received. In comparison, a decisional method is going to have an efficiency level that depends on the amount of the original data "retransmitted". A graph of this relationship is presented below. An optimization of these two

[1] The problem here stems primarily from the close relationship between ordering, consistency, and efficient use of data layout to achieve that relationship.

techniques is possible. [NBT97] presents an integrated technique that is highly efficient.

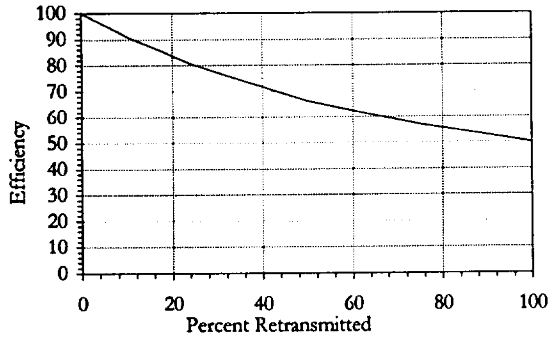


Figure 1: Decisional Efficiency

The point of this discussion is to stress the need for a request-response system for reliable delivery. Any congestion control scheme should take this into account. However, this has a hidden problem. Imagine the simple setup presented below with a single sender and two receivers. We will refer to this as the *local recovery rate problem.*

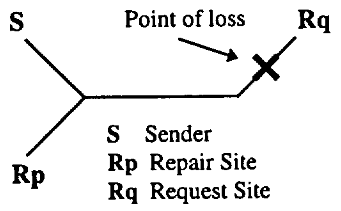


Figure 2: Local Recovery Problem

Here have a topology where three sites are distributed. We make a simplifying assumption that repair suppression (i.e. multiple repairs are not sent out for a single requests) is used. The case is that we have a sender (S), and two receivers, Rq and Rp. Rq experiences a loss on the link shown in the figure. Rq may or may not be just a single member of a multicast subtree that sends the request. The repair suppression (by SRM [FJLMZ95] or the use of DRs in RMTP [LP96]) dictates that Rp is the one to send the repair. The sender keeps sending at its own rate. It should be easy to see that the loss point (or congestion point) will not reduce its rate, in fact, it is increased by the repair being sent. This is a pathology if the link is persistently congested because the flow through the link increases with the loss experienced by Rq as more repairs are sent which create more loss. This continues until total congestion collapse occurs or other traffic through the link subsides. In order to account for this problem, the sender **must** reduce its rate because the repairing site, Rp, can not control the rate through the congested link that the sender is injecting. This is a very convincing argument for sender rate control if local recovery is allowed to occur. In fact, this argument should hold even if repairs (retransmissions) are allowed only from the sender and the repairs are not rate controlled along with the new data being injected.

Very few things in the world are mutually exclusive. It is possible, and desirable, to use both sender and receiver approaches for congestion control. A sender rate control component could optimize intra-group transmission, while a receiver component could optimize receivers into groups based on loss and available rate. A naive first approach for this would be to have receivers switch to auxiliary groups if loss levels reach certain thresholds. Such a system would try to keep the loss (and consequently the average rate) of the main group within certain limits. Thus the need and utility of a sender rate control component for reliable multicast should be clearly obvious.

Section 2 presents the primary requirements for a reliable multicast rate controller as well as design goals that should be taken into account. Section 3 presents a rate controller designed to meet those requirements. Section 4 presents simulation results of that rate controller. Then Section 5 presents some conclusions and future work with the rate controller.

2. Requirements and Goals

To fully understand the requirements for a sender rate controller for reliable multicast, we need to look at the operating environment of such a system. [YKT96] and [M97] indicate that most loss in the Internet Multicast Backbone (Mbone) is highly independent (vs. shared), i.e. spatially uncorrelated. The observation in [M97] that no relationship between sender rate and loss rate exists is difficult to except in light of two additional facts. The system under observation is highly chaotic and the consecutive loss observed was never really explained. In addition, the variable rate used was not reactive to network conditions. It was reactive to data source properties (such as lighting changes). This combined with no consideration of other traffic could easily lead to very complex relationships that are unobservable without extremely large data sets.

The rate controller should be able to adapt to loss levels that are high, but are the result of a large amount of independent loss. In this case, if the loss is only spurious, then the controller should not drastically change its rate in the long term (as TCP would). In this way, we can think of the rate controller being somewhat *loss tolerant*. The controller must also be able to deal with perhaps a large number of receivers, a large number of congested links within the distribution tree, and a large, varying amount of delay within the distribution tree. In addition, the controller must deal with the local recovery problem outlined above.

Our main goal with the controller is to develop a rate controller that works in the presence of some amount of background independent loss without reducing its rate drastically. In the presence of persistent loss (typically on one or more bottleneck links) the controller should set its rate to whatever is most responsive to other TCP and rate controlled flows. The controller assumption of the multicast infrastructure is that most receivers will experience some amount of loss "noise" that is mostly independent, but not persistent. For some, or perhaps all, receivers, some link, or set of links, will be the bottleneck were loss is going to be persistent. LTRC should determine the rate at which this loss versus the effective throughput is optimized and the impact to other competing flows is minimized. The controller must assume that other flows are responsive. If not, then the controller may enforce a minimum rate determined by the application characteristics, such as the rate of data generation, that the controller will send no slower than. These assumptions, with the exception of a minimum rate and multiple receivers, are essentially the same ones used by TCPs congestion control strategy.

The controller should reduce the risk of accidentally doing harm. Receiver controlled approaches have a problem as identified in [MJV96]. They have to assume that other receivers will follow the same actions to loss they experience. Shared links that become congested can only become uncongested if all the receivers using it remove themselves from the group. Failure to do so quickly is problematic in that congestion stays around. The use of IGMPv1, suspended processes (if control is in the application), and unreliable prunes makes this change unlikely to propagate quickly. In addition, receiver-based approaches allow denial of service attacks to take place because it only takes a single receiver to subscribe to all groups and ignore loss to have receivers sharing the same link(s) be effected. A sender rate controller must therefore protect itself from one or a few errant receivers. The easiest way to achieve this is to allow any receiver to sound a warning which the sender can adapt to. Thus only if all the nodes cooperate to cause congestion could the same situation arise.

In addition to the concepts described above, the controller should follow some generally desirable design goals. The controller should be as minimal as possible and not rely on any specific request/repair algorithm, such as SRM [FJLMZ95] or the use of specified DRs in RMTP [LP96]. This minimalist attitude should extend to both the controller, thus very little maintained state, as well as little interaction with the receivers. It should apply to a broad range of reliable multicast protocols, both connection-oriented and connection-less. A minimalist attitude helps to achieve this goal. Lastly, the controller should be configurable to be more or less responsive to loss. This helps to insure that even under circumstances with loss due to transmission error, such as wireless or satellites, and/or unresponsive competing traffic that the controller can be tuned to perform better. This also helps to match application desires and quality of service with congestion control.

3. The Rate Controller

The basic approaches taken by the Loss Tolerant Rate Controller (LTRC) to meet the requirements and goals outlined above are:

- Track loss at the receivers and feed it back to the sender as a measure of performance.
- Have the sender make rate change decisions based primarily on reported loss and not an aggregate of loss information from various receivers. Thus promote the model of making decisions based on a single routes behavior and not an aggregate of routes. In other words, don't allow one congested and two uncongested receivers to diminish the loss reported by the one congested. Make decisions on rate change based on a local observation.
- Allow the sender to track its last rate change decisions and use that history to determine how it reacts to newly reported loss. Stabilize the system in the event of a change and allow it to stabilize before making another change.
- Allow short term rate changes as well as long term rate changes. If loss is independent and spurious a drastic rate change should not be necessary. Only a short term, "equilibrium maintaining" change should be accomplished.

3.1 Controller Specification

LTRC is specified as a set of algorithms and a finite state machine. The state machine is very similar to [MJV96] in its operation. However, additions and modifications have been introduced as well as the machine operating at the sender instead of the receiver(s). For the basic approach to be useful in a variety of reliable multicast protocols, the state specification is limited to only a few different events, an internally maintained timer and incoming loss reports. These loss reports can be in the form of negative acknowledgments, reception reports, or some other direct message from a receiver to the sender. Along with this loss report is a *loss average* value. This value indicates the level of loss the receiver has observed over a "short" period of time expressed as a percentage between 0 and 100. The definition of short here is relative to how often loss is reported. The idea is to have the loss reports indicate a measure of the immediate loss being experienced. A discussion of how this can be performed with negative acknowledgments, or NACKs, is presented later.

The goal of the state machine is to control when the sender rate is increased and under what conditions a decrease is warranted. Changes to the sender rate are assumed to take effect immediately, or at the least, very shortly after the change is indicated. This is extremely important when the rate is decreased. Issues such as bucket inertia if a token bucket rate limiter are in use must be addressed.

The LTRC state specification is given in Figure 3 and a description of the used parameters is given in Table 1. The timer event, T, is used to indicate that a timer has expired. The event, L, indicates a loss report with an accompanying value is received. That received value is then compared to pre-determined thresholds, L_{min}, L_{imax}, L_{max}, or L_{wmax}, to determine if any actions or state transitions are to be performed. Actions are indicated in the figure by (i) and (d) on transitions. The (i) action indicates a rate increase and the (d) action indicates a rate decrease.

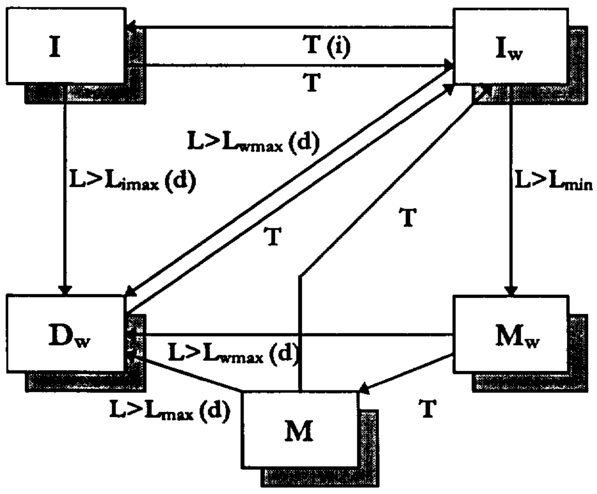


Figure 3: LTRC State Specification

The five states of LTRC are: the *increase* state (I), the *increase wait* state (I_w), the *measurement wait* state (M_w), the *measurement* state (M), and the *decrease wait* state (D_w). Each machine starts in the I_w state, and sets its rate to a *start rate*, R_0, and waits for the T timer to expire or loss to be reported above one of the thresholds.

Parameter	***Description***
T	Timer expiration
T_D	Value used for the timer
L_{imax}	Maximum loss threshold (increase)
L_{min}	Minimum loss threshold (wait transition)
L_{max}	Maximum loss threshold (post-wait)
L_{wmax}	Maximum loss threshold
R_{incr}	Rate increased by linear increase (bps)
R_{min}	Minimum allowed rate (bps)
R_{max}	Maximum allowed rate (bps)
R_0	Start rate of controller (bps)

Table 1: Description of Parameters

The timer, T, uses a value that is referred to as the *loss detection time*, T_D. Calculation of this value is discussed below. The T_D time is a measure of the time it takes for a rate change to "flush" through the system. Because LTRC is intended to work primarily with connection-less protocols, this value is just an estimate. For connection oriented protocols, such as RMP or RMTP, a more bounded time value can be used that is based on message stability information. Such protocols can increase faster and stabilize themselves much quicker due to the tight interaction between sender and receivers.

The change of rate in LTRC is done following the same philosophies outlined in [J88] for TCP. LTRC uses a multiplicative increase in rate until the first decrease occurs. Afterward a linear increase is used. In both increase methods (linear and multiplicative), the new rate, R_{x+1}, is a function of the current rate, R_x. The linear increase rule is given below.

$$R_{x+1} = R_x + R_{incr}$$

The multiplicative increase rule is given below.

$$R_{x+1} = I_g R_x$$

The I_g factor is the increase gain. Currently, a value of 2 is used just as is used in TCP for slow start. The decrease rule is given below.

$$R_{x+1} = D_g R_x$$

The D_g factor is the decrease gain. Currently, a value of 0.5 is used just as is used in TCP. A rate change has an additional set of restrictions involved. A set of minimum and maximum rates are enforced as well. Any rate change must adhere to those restrictions. The need for these restrictions might be less than obvious. The minimum rate allows LTRC to be used in applications where a *data generation rate* places a lower limit on how fast LTRC must transmit to ensure data is not buffered indefinitely. In this case, it is very important to also impose thresholds at the receiver so that if loss becomes too high for too long, then the application should leave the group. The maximum rate addresses another issue involved in connection-less protocols. In the event of partitions or congestion collapse on the channel used for loss reports, the rate should not continue to rise unchecked[2].

LTRC is purposely designed to work very closely to the way TCPs congestion control works as explained in [J88]. The main differences are:

- LTRC is clocked with respect to time and not ACKs. This is evident from the use of the timer to initiate rate increases.
- LTRC makes decisions on rate increase and decrease based on loss information being fed back to the sender as well as recent activities (indicated by the current state).

While in the increase state, LTRC uses the L_{imax} loss threshold to signal that the increase caused too much loss. The rate is decreased and LTRC waits in the drop wait state for the system to stabilize before making another decision. The increase wait state is entered when we assume the system is somewhat stable. While in this state, if loss occurs above our L_{min} threshold, we go into another wait state before we measure the loss again. If the loss was just temporarily above the threshold, we return to the increase wait state. If loss is above the L_{max} threshold, then we decrease the rate and wait for it to stabilize. In the increase wait, measurement wait, or measurement states, if loss is reported that is above the L_{wmax} threshold, then we decrease the rate immediately.

LTRC also uses an immediate action to address loss that is discussed below. This part of LTRC has an impact on rate, but can be considered to be separate from the major longer-term rate decisions outlined above. Many more issues are also involved in LTRC, including loss calculation at the receivers, timer calculation, etc.

3.2 The T_D Timer Value

The T_D timer value is composed of a large number of factors. In addition, the value will be dependent on the location of any bottleneck link. This widely varying value then becomes extremely difficult to measure or estimate when it can vary by several orders of magnitude. Compare a local network buffer with a link half way across the world. They each have different delays, propagation times, bandwidth, and distance from the sender.

Lets analyze the components of this timer. For this *loss detection time*, we want to know "how long it would be before a loss would be reported". Notice this is not, "loss occurs". The detection time needs to be based on when that loss is reported. If SRM is being used on each individual loss, then we can look at this time as something of this form[3].

$$T_D \approx Q_d + RTT + SRM_{RQD}$$

The SRM_{RQD} is the delay from the request scheduling. This has a relationship to the RTT from the sender to the receiver and can be approximated if we place bounds on the values that the receiver might be using for C_1 and C_2. An estimation of the highest RTT calculated for the group provides a nice cushion and value to use for RTT in this calculation. The Q_d element is the hardest to approximate. It represents the portion of time that corresponds to the bottleneck link building up its queue and causing the initial drop. An estimate of this delay could be done by sampling jitter in the RTT calculations. This is somewhat difficult and requires multiple RTT calculations to the same receiver, usually the one with the highest RTT. This queue delay is going to have a lot of very hard to calculate factors involved, including bandwidth, queue limit, queuing discipline, link delay, how much the bandwidth was overshot, etc. For this reason, the timer value is almost impossible to estimate with any real amount of certainty. Thus a static

[2] Loss of this "back channel" is very dangerous for LTRC. Maximum rate ensures that the rate goes no higher than a certain limit, but in addition, the protocol should use something like reception reports or periodic session messages containing loss information and current state to provide positive feedback. Such a scheme would cut the rate if the receivers suddenly all left the group.

[3] Determining this may be less than obvious. The time involves the delay in having the queue build up in the bottleneck link, Q_d, the sending of the data packet that is dropped due to the queue, ½ of an RTT, the delay in the request, SRM_{RQD}, and the delay in the request coming back to the sender, another ½ of an RTT. The first ½ RTT could be part of the queue buildup, but adding it in individually gives us a more conservative estimate.

value for T_D is the safest bet in most cases. However, this value should be configurable so that small, local groups are not needlessly penalized.

3.3 Loss Average Calculation

To report loss to the sender, each receiver must maintain a measure of loss. A possible mechanism to do this is to report a loss average measure in the negative acknowledgments (NACKs) that a receiver sends while using SRM. This piggy-backing of loss information in NACKs is a very useful tool. Here we describe how loss information can be tracked and reported.

On receiving each packet the average loss is updated. This loss sample, l_i, is calculated by dividing the number of lost packets by the number of expected packets. The average, avg_l, is calculated by an exponentially weighted moving average (EWMA) that is close in design to the one used for queue length in RED [FJ93].

$$avg_l \leftarrow (1 - w_l)avg_l + w_l l_i$$

In addition, avg_l is updated when a repair is received for a requested packet. This acts as a relaxation mechanism so that received repairs decrease the loss average.

$$avg_l \leftarrow w_r avg_l$$

The weights, w_l and w_r , are used to weight the moving average. You should notice a couple things from this immediately. The first is that if we update on reception of every packet, then a lot of those loss samples are going to be 0. This will cause the average to decrease very quickly under no loss. The second observation is that the relaxation mechanism will bring the average down rapidly if a lot of repairs start coming in quickly. This is intentional. Receiving repairs indicates that congestion is abating, so the loss reported should be less.

The loss average is reported in each NACK at the time the NACK is sent, not scheduled. In addition, the value is expressed as an integer from 0 to 100. Upon NACK retransmission, the loss value is updated. Thus retransmits will contain a value pertaining to the loss value at the time the NACK is sent. Thus the request delay of SRM acts to help smooth out any spurious loss. If, for instance, only a few lost packets are observed, then by the time the NACK is sent, the loss value would be very low (probably 0) if no more loss (or very little loss) was experienced or may increase if more loss was experienced. This mechanism also works to reduce ambiguity as to the independence of the loss. Even suppressed NACKs will cause the loss value to increase because it is updated before the NACK is scheduled. Thus the loss value is not dependent on the sending of a NACK, but on the loss experienced. In circumstances where loss is independent, but of low value, the loss value will be very small, but many NACKs will be sent. In circumstances where loss is shared, but of a higher level, the loss value will be higher no matter who actually sends the NACKs because every member of a congested subtree will experience the same loss. This does not eliminate ambiguities resulting from feedback suppression of NACKs, but it does help to allow LTRC to make more accurate decisions.

The same kind of loss information can be returned in positive acknowledgments (ACKs), such as is used by RMTP [LP96]. In this case, ACK bitmaps act to give potentially a better loss value and loss picture over varying time frames. This would not prevent a loss measure from being calculated and returned, but would allow a time frame to be attached to that loss measure.

3.4 Immediate Reaction to Loss

To be responsive, LTRC must address every loss that is reported and not just when the loss value is beyond a threshold. Essentially, LTRC must take some form of *immediate action*[4]. This action must be performed regardless of how rate will be changed based on loss information. Imagine that a sender is sending at a constant rate. Spurious congestion causes independent loss to be reported. The sender only wants to adjust his rate a small amount and not a large amount to account for the loss and the subsequent repair that it or another member may send. The larger rate adjustments should only be performed under more persistent loss (as defined by the loss thresholds). To maintain equilibrium, the sender would want to "consume" an amount of bandwidth equivalent to how much additional bandwidth the repair will take up (if the sender does not send the repair itself or repairs are not rate controlled). Suppose a repair is sent for each data packet, 100% independent loss from a very large group. We would like to have the original sender slow its rate of new data to 50% of its original rate if no loss were seen[5]. Thus the overall rate of the whole group (including any locally sent or globally sent repairs) stays constant. If individual (or shared individual) loss

[4] The term *immediate action* was chosen because it mimics the action taken when addressing misfires or jams of early firearms such as muzzleloaders.

[5] If every data packet is sent twice (once for the original and once for the repair), we get a true new data rate of 50% of the overall rate.

reaches certain levels, LTRC should slow its rate. This is what the state machine attempts to do.

LTRC must then "consume" a portion of its bandwidth when a NACK is received or a retransmission of previous data is eminent. Implementation of this is quite trivial with a token bucket rate limiter. In this case, the number of tokens in the bucket is decreased by the amount of the repair. For a send time interval mechanism, the send time would simply be extended or the next interval would be ignored. This overall technique is called, *bandwidth consumption*. The amount of bandwidth consumed should be a multiple of the repair size.

For bandwidth consumption to work, the sender must know when a data retransmission is going to occur, even if the sender is not the one to send the retransmission. This could be done using unicasts if need be. In the LTRC simulations with SRM, this consumption occurs on receiving a NACK for a packet which does not have a corresponding repair scheduled. Thus duplicate requests are not accounted for if they are received within the pending repair or ignore periods. If the sender later sends the repair, then the repair takes up additional bandwidth. In this case where the sender always sends a repair in addition to consuming bandwidth, the true new data rate drops to 33% of the original rate.

3.5 Maximum Rate

Because LTRC might have some delay in receiving loss reports from receivers as well as being connection-less, it is important that the available rate not be overshot too drastically. In addition, LTRC should attempt to insure fairness with competing flows by using the loss reports to determine how fair it should be. For both these reasons, LTRC should dynamically calculate the maximum rate it should use. In effect, this also acts to force LTRC to be responsive.

Dynamically calculating the maximum rate based on TCP responsiveness can be done by using the TCP responsiveness formula from [FF97]. This is convenient because it only has loss rate, RTT, and segment size as its parameters.

$$T \le \frac{1.5\sqrt{2/3} * B}{RTT\sqrt{p}}$$

In this equation, T is specified in bytes-per-second (Bps), B is the segment size in bytes, RTT is the total round-trip time (including queuing delay) in seconds, and p is the loss rate. The initial problem here is what to use for RTT and what to use for p. An as of now untested approach is to have the receivers send back two loss rates, one for short term and another for longer term. The long term loss rate would be used as p and the RTT to the receiver would be used as the RTT. Thus the controller would get a snapshot of how responsive it is on the route to the receiver. This is yet to be simulated or evaluated, but it does hold some promise.

3.6 Linear Rate Increment

To be as close to TCPs responsiveness as possible, LTRC should determine the increment it should use for linear increase, R_{incr}. The TCP rule is "at *most* one packet per round-trip time" [J88]. A conservative approach here would be to calculate an upper bound on R_{incr} as something similar to this.

$$R_{incr} \le \frac{2T_D B}{RTT_{\max}}$$

B is the same used in the maximum rate calculation. The RTT represents a conservative round-trip time estimate, and the rate increase is performed every other T_D time increment. This increment should be set to a maximum value equal to the present rate or a percentage of the increment calculated above. This way, the linear increase will never be more than what the multiplicative increase would produce.

3.7 Optimizing for Large Groups

Large receiver groups with high levels of independent loss can cause LTRC to scale poorly. The presence of duplicate repairs can cause more congestion and eventually cause LTRC to reduce its rate too far. To combat this, a possible solution is to use a mechanism to combat independent loss more fiercely. Forward Error Correction (FEC) is a very good mechanism to avoid the impact of independent loss. The simulations do not use this. However, future simulations will explore the issue.

Protocol modifications must be done to incorporate FEC. Mostly, this means modifying SRMs operation. The basic approach works like this. A set of data packets, d_1 ... d_k, is encoded into a set of encoded packets, e_1 ... e_k, and a set of "parity" packets, p_1 ... p_{n-k}, n-k is the amount of redundancy in the set. The sender sends the encoded packets, e_1 ... e_k, and then starts sending the next batch of encoded packets for the next set of data packets. The receivers can decode the original set of data packets if they receive the whole encoded packet set. If a receiver misses any of these packets, it then sends a NACK identifying the data set it needs and the number of encoded packets it did not receive. Receivers do suppression based on number of encoded packets requested. I.e., if one receiver loses 2 packets and another 3, then 2 would suppress sending a NACK if it saw 3 send one. The sender, upon receiving NACKs, sends the requested number of packets from the set of parity packets. If the parity set is exhausted, then it starts with the encoded set again. After sending a parity, the sender then ignores the number of requests or lower. If a new request comes in for more, it then sends the difference, i.e. if it has sent 3 and it gets a request for 5, it then sends 2 additional. This is an optimization of the integrated FEC approaches described in [NBT97]. As a last resort, the receiver can explicitly request certain packets in the encoded set using the normal SRM operation.

This integrated FEC technique is very effective in place of local recovery because both address the same problem of efficiency. The redundancy level should be set based on short term loss. I.e. 25% indicates a k of 10 would give a n-k of about 4. One advantage here is that the redundancy level is only going to effect the amount of encoding space consumed at the sender. More redundancy allows the parity set to be larger and thus helping to preventing the retransmission of the encoded set. From the receivers perspective, it only needs to know the size of the encoded set, k, and the location of each parity packet, i.e. a sequence number for the parity packet. The actual size of the parity set is not needed and could adaptively change. This approach ensures that the efficiency of request/response is preserved, while gaining the ability to handle independent loss efficiently. With independent loss being handled more efficiently, the rate controller can adapt to loss more efficiently.

4. Simulations

To investigate the performance of LTRC under various conditions, it was implemented in C++ along with a C++ implementation of SRM and integrated into the Network Simulator, ns [NS97]. Throughout the design process, the simulator was used to try out hypothesis and refine the controller. The controller is not perfect, it is not even done, it is merely in the process of maturing.

4.1 Simulation Topologies

The topologies used for the simulations are based on the type of arrangement shown in Figure 4. To perform simulations with fairly large group sizes, a simplified topology was needed.

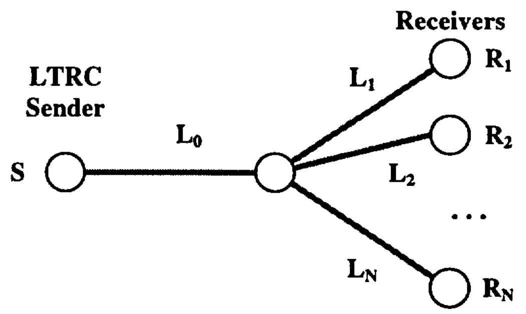


Figure 4: Base Topology for Simulations

The parameters used for most of the simulations are given in Table 2. Differences are mentioned in the results.

Parameter	*Value*	*Parameter*	*Value*
L_{imax}	2	w_l	0.25
L_{min}	2	w_r	0.25
L_{max}	5	R_{min}	32 Kbps
L_{wmax}	10	R_{max}	1.5 Mbps
R_{incr}	16 Kbps	R_0	128 Kbps
T_D	2 sec.		

Table 2: Base Simulation Parameters

The links are given varying parameters such as bandwidth, delay, queue limit, and queuing discipline based on the goals of the individual simulation. These parameters are mentioned with the results of each simulation.

4.2 Performance Measures

For the simulations, we define a few measures of importance. For transport protocols that must provide reliable delivery, an overwhelmingly important measure is perceived throughput at the receiver. Congestion control will undoubtedly effect this measure. In addition, throughput can be dually influenced by the rate increase and decrease measures mentioned above. Thus to get better throughput, the increase and decrease parameters can be played with at the expense of more loss. For this reason, the rate change parameters are static for all simulations. In addition, loss is used as a measure instead of throughput. With one exception and that being the measures of fairness. The goal of congestion control is to measure throughput against loss and to be responsive to other traffic.

A primary concern with LTRC is its scaling properties. To explore this, a set of simulations were set up and ran. These were:

- loss rate versus varying number of receivers with a static number of links congested

- loss rate versus varying number of congested links with a static number of receivers
- loss rate versus varying round-trip times with a static number of receivers and congested links

In these simulations, the loss rate is measured over three different time scales. The first, 1 second, represents spurious, short term loss. The second, 10 seconds, represents mid term loss and the last, 100 seconds, represents long term loss. Each simulation was run for 100 seconds or the equivalent of a normal kind of transfer. Additionally, each simulation used a link bandwidth of 1.5 Mbps and Drop-Tail queuing. The delay of all links was the same except in the last simulation where it was allowed to vary.

Links were congested by placing a Constant Bit Rate (CBR) source at the LTRC source end of the link and a sink at the other end. The CBR was set to consume half of the link bandwidth, or about 750 Kbps, with a uniform amount of random noise added to the timer interval. I.e. the interval between sends is allowed to vary by 0.5 times its true value. Thus each CBR will transmit a true instantaneous rate uniformly distributed over [497 Kbps, 1.5Mbps] but with an average transmission rate of 750 Kbps. Instantaneously, we should see that each link has some random amount of noise as 750 Kbps is approached. Thus, we should see some amount of independent loss as 750 Kbps is approached. This setup showed to be the most difficult for LTRC to deal with. For situations where a single link is congested, the loss rate is very low. This kind of situation is evident from the percent congested simulation results. When L_0 is congested as well as the other links, LTRC should experience more loss because the situation is analogous to having multiple congested gateways in addition to 100% shared loss.

The sender being controlled by LTRC uses a token bucket rate limiter that has a refresh rate of 50ms. The sender transmits at 1.5 Mbps (or about every 5.33ms) if not controlled. This interval also has the same amount of random noise added to it as mentioned above. This adds to the randomness of the rate that is controlled by the rate limiter proscribed by LTRC. Additionally, only the sender is allowed to send repairs. This helps to be more efficient as in this simulation of SRM, all repairs are multicast to the whole group. Simulations with active local recovery showed no substantial difference. The SRM simulation component uses adaptive constants for its request processing.

Simulations are also presented that demonstrate the fairness that LTRC can achieve among several instances of itself. For purposes of these simulations, we will define a *fairness index*, f_i, as:

$$f_i = \frac{\left(\sum_{x=0}^{N-1} T(x)\right)^2}{N\sum_{x=0}^{N-1} T(x)^2}$$

T(x) in the equation is a measure of the resource that is measured. In the case of the fairness simulations, we are measuring receiver perceived throughput which accounts for repairs received as well. Each receiver is treated as a single entity, so the value N will be equal to the number of groups times the number of members per group.

The fairness simulations contain no CBRs, and each link is 1.5 Mbps and 50ms, except for L_0 which is 2.5 Mbps. All of the LTRC senders are in the S node and each R node contains a single receiver for each group. Therefore, the groups are 100% overlapped. For the fairness simulations, L_{wmax} was lowered to 2. This was done to insure that "stable" senders would give up bandwidth easier. The loss rate simulations did not use L_{wmax} because they were never competing for traffic while they were stable. Each sender in the fairness simulation was started on the uniform distribution [1 second, 3 seconds].

4.3 Simulation Results

For each of the scaling simulation, the maximum loss rate observed for each run was recorded. Therefore, the value presented is the maximum loss rate observed over a number of simulation runs for that setup.

Loss rate versus number of receivers is presented in Figure 5. The delay used for all links was 50ms. All links were congested and the 1, 10, and 100 second loss time scales are shown. It should be fairly obvious that the loss rate does not substantially change after about 20 receivers. The throughput for the runs stayed right around 600 Kbps with hardly any variation. All of the runs showed that the 1 second loss occurred during the multiplicative increase. By contrast the linear increases showed very low short term loss.

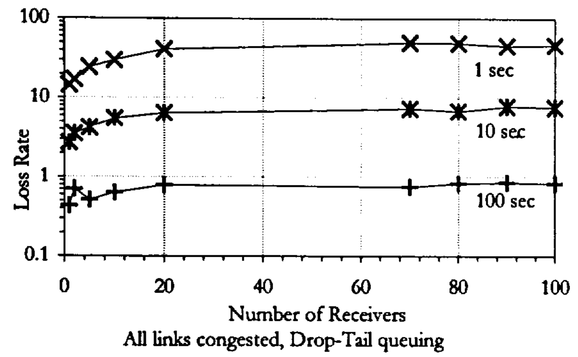


Figure 5: Loss Rate vs. Number of Receivers

When the L_0 link is *not* congested, we get a slightly different story as depicted in Figure 6. The graph is slightly more varying due to the fewer number of samples as well as the enhanced randomness of the CBRs, but the trend is never the less evident. The loss rate is substantially lower for medium and long term loss. This also represents a more accurate setup for most situations in which LTRC will have to operate. This is because rarely should LTRC see 100% shared loss and multiple congested gateways on the same route. The throughput observed stayed right around 610 Kbps with hardly any variation.

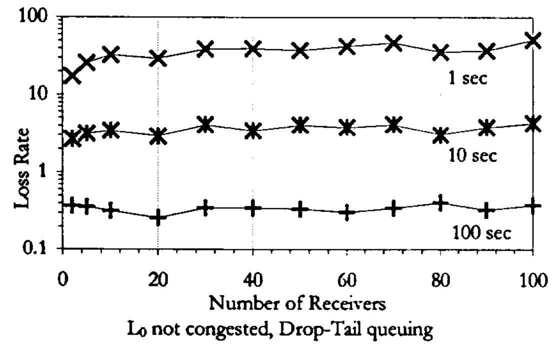


Figure 6: Loss Rate vs. Number of Receivers (Set 2)

Loss rate versus percentage of congested links is presented in Figure 7. The delay used for all links was 50ms and the number of receivers was 100. All links, except L_0, were congested and the 1, 10, and 100 second loss time scales are shown. It should be fairly obvious that the loss rate does not substantially change after about 10%. The throughput for the runs stayed right around 610 Kbps with hardly any variation. All of the runs showed that the 1 second loss occurred during the multiplicative increase just as in the previous simulation.

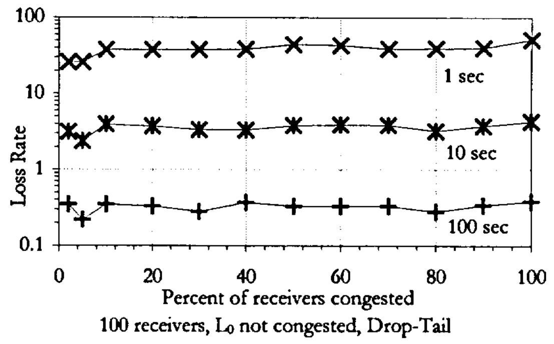


Figure 7: Loss Rate vs. Congested Percentage

Loss rate versus round-trip time is presented in Figure 8. The delay used for all links was RTT/4, the queue limit held constant at the default setting of 50, and the number of receivers was 100. All links, except L_0, were congested and the 1, 10, and 100 second loss time scales are shown. The throughput for the runs stayed right around 610 Kbps with only about 50 Kbps of variation. The outlying runs, RTTs of 0.001 and 1, showed a reduced throughput of roughly 390 Kbps. Mainly, this seems to be due to mismatches with T_D, a tendency to overshoot the target rate drastically and then cut the rate too much, and a mismatch with queue limits which were not changed. It should be noted that LTRC reduced its rate under these circumstances instead of allowing the loss rate to climb. All of the runs showed that the 1 second loss occurred during the multiplicative increase just as in the previous simulations.

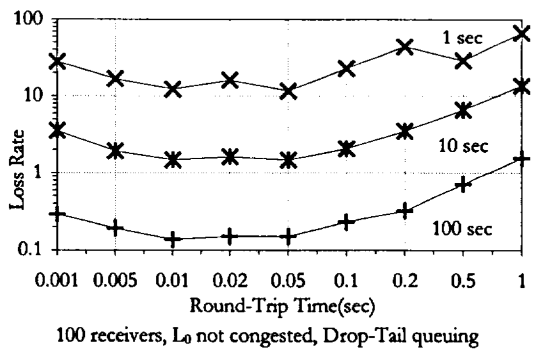


Figure 8: Loss Rate vs. Round-Trip Time

The previous simulations should show that LTRC scales nicely with regard to number of receivers, percentage of links congested, and round-trip time (or group locality). However, a congestion control scheme must be responsive and attempt to achieve some measure of fairness.

Using the fairness index defined above, similar scaling issues where explored. With respect to the number of receivers, the fairness index stayed very constant as long as the number of groups was kept constant. The only substantial variation was noticed when the number of groups was allowed to vary and the number of receivers was kept constant. A graph of this is shown in Figure 9. Each point shows the range of observed values as well as the average of those values represented by a dash. 10 receivers were used so that the larger number of group simulation runs would finish in a reasonable amount of time. As you should see, the variation was fairly broad. It should be noted that the largest variation in the results was the result of a set of groups getting ½ the bandwidth of another. At no time did a group or receiver only get the minimum rate.

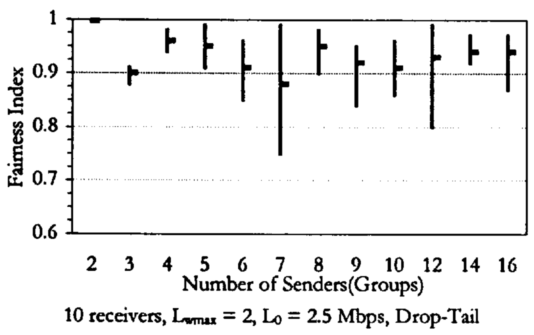


Figure 9: Fairness vs. Number of Groups

When RED queuing is used in place of Drop-Tail, we see a reduction in the amount of variation. This is shown in Figure 10.

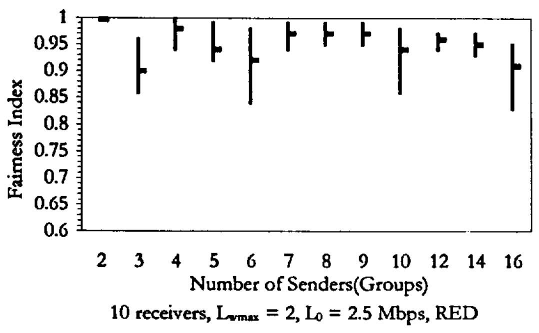


Figure 10: Fairness vs. Number of Groups (RED)

The simulation can be made to be fairer by using a maximum rate calculation as well as using a longer run length. A length of 100 seconds with T_D of 2 seconds and R_{incr} of 16 Kbps does not allow a starved sender much time to recover. Preliminary results indicate that longer run lengths as well as maximum rate calculations do allow for more constant fairness measures.

5. Conclusions and Future Work

The need and utility of a sender-based rate controller for reliable multicast protocols is quite evident. The Loss Tolerant Rate Controller (LTRC) presented goes a long way to achieving many of the properties such a rate controller should have. In many ways, LTRC poses more questions than it answers.

The basic approaches of LTRC have use in other areas. It is interesting to investigate if loss tolerant rate control could be applied to lossy links, such as wireless and satellites. In that environment FEC would be a necessity and tuning the loss thresholds would be necessary. That begs the question of whether adaptive loss thresholds would be of any use. It would also be interesting to investigate the use of loss report feedback in other contexts, such as request-response protocols and transaction protocols.

LTRC, as well as other congestion control schemes, must assume that traffic is responsive and must also be responsive to other traffic. This basic assumption is the only way that wide spread congestion control will work without imposed fairness in the network infrastructure, such as RED with Fair Queuing and resource reservation. A configurably responsive scheme, such as LTRC, might violate some of those assumptions. However, to suite some application demands, it might be necessary. Application-Transport characteristic mismatches hurt not only the network, but end-users. A good example is the use of TCP for HTTP. Clearly, the two are not designed to work well together and now we are only beginning to understand exactly how badly they interoperate. A possible way to alleviate these kinds of problems is to allow configuration of the transport congestion control under certain restrictions. This is a very logical next step because it incorporates many of the basic issues of ALF [CT90]. ALF should also encompass the congestion control scheme of the transport as well as the protocol itself.

Some of the questions LTRC raises that are to be explored are:

- Would a straight linear increase with a larger increment be more effective than a multiplicative increase?
- What is the effects of consecutive loss on loss averages at the receivers?
- What is the best policies for changing rates? Should increases and decreases take effect immediately or should ramp-ups and ramp-downs be used?
- What kinds of measures would produce better T_D values?

Beyond the logical next steps of adding FEC to the simulations and maximum rate determinations, numerous other issues are being explored. As has been mentioned, integrating a partial receiver-driven component could add several properties and allow LTRC to be quite powerful and adaptable for many uses beyond bulk-transfer. An investigation of the state machine stability as well as a full parameter study using the simulations is also well underway.

6. Acknowledgment

This work is funded by GlobalCast Communications, Inc. and by NASA Cooperative Agreement NCC2-979 at the NASA/Ames IV&V Facility in Fairmont, WV. In addition, extreme thanks go to the authors and maintainers of the network simulator, ns, without whose work on making ns available this work would not be possible.

7. References

[CT90] Clark, Tennenhouse, "Architectural Considerations for a New Generation of Protocols", Proceedings of SIGCOMM 1990, pp. 201-208, September 1990.

[FF97] S. Floyd, K. Fall, "Router Mechanisms to Support End-to-End Congestion Control". Technical report, February 1997, LBL.

[FJ93] S. Floyd, V. Jacobson, "Random Early Detection gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking, V.1 N.4, August 1993, p. 397-413.

[FJLMZ95] Floyd, Jacobson, Liu, McCanne, Zhang, "A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing", Proceedings of SIGCOMM 1995, pp. 342-356, August 1995.

[J88] V. Jacobson, "Congestion Avoidance and Control", Proceedings of SIGCOMM 1988, pp. 314-328.

[LP96] K.C. Lin, S. Paul, "RMTP: A Reliable Multicast Transport Protocol", IEEE INFOCOMM 1996, March 1996, pp. 1414-1424.

[M97] M. Handley, "An Examination of Mbone Performance", UCL and ISI, January 1997.

[MJV96] McCanne, Jacobson, Vetterli, "Receiver-based Layered Multicast", Proceedings of SIGCOMM 1996, pp. 1-14.

[NBT97] J. Nonnenmacher, E. Biersack, D. Towsley, "Parity-Based Loss Recovery for Reliable Multicast Transmission", Technical Report 97-17, Dept. Of Computer Science, U. Massachusetts, March 1997.

[NS97] UCB/LBNL Network Simulator (ns). Version 1. 1997.

[YKT96] M. Yajnik, J. Kurose, D. Towsley, "Packet Loss Correlation in the Mbone Multicast Network", Proceedings of IEEE Global Internet Conference, London, Nov. 1996.