LE-SBCC: Loss-Event Oriented Source-based Multicast Congestion Control

Puneet Thapliyal, Sidhartha, Jiang Li, Shivkumar Kalyanaraman[1][2]
Rensselaer Polytechnic Institute, Troy, NY 12180
Contact Email: shivkuma@ecse.rpi.edu

Abstract

Congestion control is an important building block of a Quality of Service (QoS) system for multicast-based multimedia services and applications on the World Wide Web. We propose an end-to-end single-rate multicast congestion control scheme (LE-SBCC) for reliable or unreliable multicast transport protocols. A distinguishing feature of the scheme is that it is *purely* source-based, i.e., it requires no support from multicast receivers or network elements, which makes it very easy to deploy for a range of protocols. It addresses all the pieces of the single-rate multicast congestion control problem including *drop-to-zero* issues, *TCP friendliness* and RTT estimation. The scheme design consists of a cascaded set of filters and a rate-based additive-increase multiplicative-decrease (AIMD) module. These filters together transform the multicast tree to appear like a unicast path for the purposes of congestion control. Unlike TCP, the scheme is not self-clocked but acts upon a stream of loss indications (LIs) from receivers. These LIs are filtered to get a stream of loss events (LEs) [8] (at most one per RTT per receiver). This LE stream is further filtered to extract the maximum LEs from any one receiver. Then the scheme effects at most one rate-reduction per round trip time (RTT). A range of results (numerical, simulation and experimental) is presented and compared against the mathematical model of the scheme components. Furthermore, we have successfully adapted TFRC [8] to our scheme, which is important to multimedia services desiring relatively flat data transfer rate. The scheme achieves all its goals in the common case of *partial LI aggregation*[3], though the rare worst case of *complete LI aggregation* could be aggressive towards TCP.

Keywords:

Multicast, congestion control, multimedia, drop-to-zero, TCP friendliness, loss event

1 Introduction

Multicast is the preferred transport mechanism for bulk data transfer to multiple receivers especially in multimedia applications and services on the internet. Applications like content distribution, streaming, multi-player games, multimedia multi-user chat/telephony, distance education etc could benefit from multicast and QoS. Multicast congestion control is the first step toward a multicast QoS architecture for the Internet. This paper presents an end-to-end single-rate congestion control scheme (LE-SBCC) for reliable or unreliable multicast transport protocols. The scheme is *purely source-based* because it operates on a stream of loss-indications (LIs) from the receivers and does not require any other support from network elements, receivers or in the packet format of underlying multicast transport protocols. This feature distinguishes it from recent single-rate schemes like PGMCC [20], Equation-based TFMCC [12, 24], MTCP [19], Kasera et al [14] and Golestani et al[9]. These latter schemes require some form of special support at receivers (eg: acker capability, RTT/loss estimation capability), network elements (eg: scheme-specific aggregation capability, active services) or protocol packet-formats (eg: new header fields). A *purely* source-based scheme can be implemented by a simple upgrade of the source of a multicast transport protocol without touching the receivers as long as they generate minimal congestion feedback. The source-based nature of the scheme does not incur extra congestion-related (S,G) state in network elements and receivers, which can be directly exploited to support multi-source multicast applications (like interactive multi-player games and online military training simulations). In this sense, our approach is very lightweight in terms of implementation, upgrade and management compared to the earlier schemes, while achieving comparable performance levels.

This scheme is modular and can allow different control policies like AIMD (for data) or TFRC (for multimedia). This holds tremendous potential in the implementation of multimedia services and applications such as live media streaming, video conferencing and video-on-demand.

The LE-SBCC scheme builds upon our earlier work GSC [16] and Bhattacharya et al's LPRF [2, 4]. It differs from these prior schemes in that it tackles the drop-to-zero problem under high degrees of multiplexing, addresses

[1]We acknowledge the help of Jye-Young Song to implement this scheme
[2]This work was funded in part by NSF ANIR grants: ANI9806660, ANI9819112
[3]Assuming that network elements aggregate the reverse LI flow to avoid implosion e.g. PGM model [22]

several implementation issues and has a much faster transient response. We believe that it is the first scheme to fully address all components of single-rate source-based multicast congestion control, i.e. *drop-to-zero* issues, *TCP friendliness*, RTT estimation, robustness and scalability up to 10,000 receivers.

LE-SBCC belongs to the class of single-rate schemes, i.e., which operate at the rate dictated by the worst path on the tree. Similar to PGMCC [20], it is aimed at small-medium scale receiver sets (up to 10,000 receivers). There exist another distinct class of congestion control schemes which are *multi-rate and receiver-based.* In such schemes, the sender organizes data as multiple layers, with each layer being sent to a separate multicast group address. Receivers measure congestion and determine the number of groups they join. These schemes could be further classified as end-to-end (eg: RLM, RLC, and FLID-DL [15, 23, 5, 7, 3]) or network-feedback driven (eg: [13, 21]).

In contrast to our work, the network-feedback driven schemes aim to achieve max-min fairness, but require complex support at all nodes in the multicast tree. The end-to-end receiver-based approach is attractive because of its scalability to very large group sizes. But recent analysis by Gopalakrishnan et al [10] suggests that the end-to-end receiver-based approaches are coarse-grained (in terms of join/leave latency and traffic on each group) and may have stability/fairness problems. Our scheme does not fall into this class, and is fine-grained in terms of control and traffic granularity. The combination of source-based and receiver-based approaches is largely an open problem.

2 LE-SBCC: Scheme Description

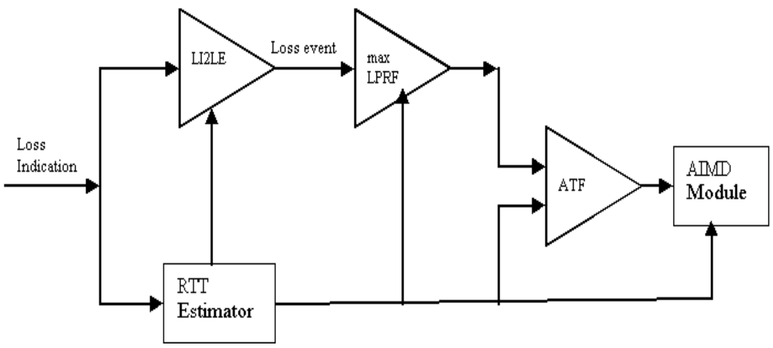


Figure 1: Cascaded Filter Model of LE-SBCC implemented at the multicast source

The LE-SBCC scheme consists of a *purely source-based* cascaded set of filters and RTT estimation modules feeding into a rate-based additive increase/multiplicative decrease (AIMD) module (Figure 1). These filters, elucidated in section 2.1, together transform the multicast tree to appear like a unicast path for the purposes of congestion control. Also, unlike TCP, the scheme is not self-clocked but acts upon a stream of loss indications (LIs) from receivers. The filters are then designed to address the following key problems:

Drop-to-Zero: is the problem of reacting to *more loss indications (LIs) than necessary* leading to a beat-down of the multicast flow's rate[24, 4]. This occurs because the multicast flow receives LIs from multiple paths and may not filter LIs sufficiently. In an earlier work [16], we showed that *without* flow multiplexing on paths, the problem of independent multi-path loss can be addressed through a simple adaptive time filter and a TCP-like RTT/RTO estimator. A single flow's congestion control actions directly impacts the loss rate and loss patterns in this case. However, this solution is insufficient if paths display a high degree of flow multiplexing, because then any *single* flow's congestion actions do not appreciably impact the path loss rates.

TCP-unfriendliness: is the problem of reacting to *less LIs than a hypothetical TCP flow would on the worst loss path* [6, 24, 12]. TCP-friendliness is a subject of current debate[8, 20, 12]. In particular, Padhye et al's TCP throughput formula [17] requires the definition and estimation of "loss rate" and "RTT" of the paths. The unicast scheme, TFRC introduced the concept of loss events (LEs) and uses the LE rate instead of the LI rate as the "loss rate." *An LE is a binary number defined every RTT per receiver which is 1 when one* *or more LIs are generated in that RTT by the receiver, and is 0 otherwise.* Like TFRC, this paper will also first derive LEs from LIs before responding to congestion.

On the subject of RTT estimation, PGMCC and TFMCC [20, 12] have receivers estimate RTT and drive the control based upon paths having the maximum value of $(RTT\sqrt{p_{le}})$, where p_{le} is the *per-packet LE probability*. Our proposed scheme LE-SBCC computes $\max RTT$ and $\max p_{le}$ *concurrently* and uses both these estimates to drive the AIMD module. This decoupled use of $\max RTT$ and $\max p_{le}$ represents a sub-optimal approach in general[20, 12, 24]. However, our RTT-estimation approach tends to measure the $\max RTT$ from the *currently congested sub-tree*, which reduces this suboptimality in practice.

2.1 Cascaded Filter Model

Figure 1 illustrates the outline of the building blocks of the scheme implemented at the multicast source. The cascade of filters converts a stream of loss indications (LIs) from receivers into per-receiver LEs, filter the LEs further, and finally outputs a stream of rate-reduction indications (RRs) to the rate-based AIMD module (at most one RR per RTT). The AIMD module performs multiplicative rate decrease upon receipt of a RR and performs rate increase each time a RR is not received within an increase interval. Observe that this closely models the behavior of AIMD control on a *single path* where RRs are triggered by LEs [8]. The modules work as follows:

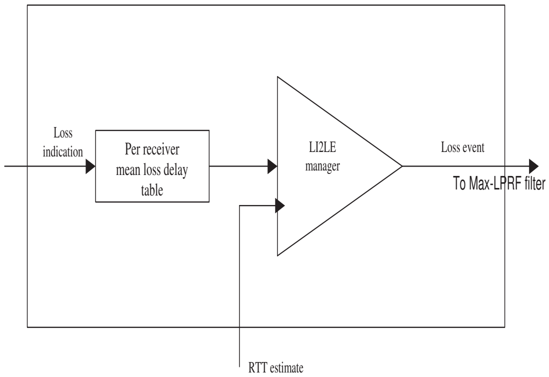


Figure 2: Loss Indication to Loss Event Filter (LI2LE)

LI2LE filter: This filter converts per-receiver loss indications (LIs) into per-receiver loss events (LEs). Recall that an LE is a per-receiver binary number which is 1 when one or more LIs are generated per RTT per receiver, and 0 otherwise. The source stores a *per-receiver timestamp* ($T_{LastPassed}$) which records the time when the last LI was converted and passed as an LE. If a new LI arrives from the receiver *after* a period $SRTT+2\sigma$ of[4] $T_{LastPassed}$, it is converted into an LE and passed, and the timestamp $T_{LastPassed}$ is updated to the current time. Else the LIs are filtered. This filter is critical because the rate-based AIMD module is not self-clocked (no acks like TCP, PGMCC [20] or Golestani [9]); which leads to large burst losses with drop-tail queues (see section 3.1.3).

Max-LPRF: The goal of this filter is to pass, on the average, the number of LEs corresponding to what the source would have received from the worst case receiver. This worst-case loss rate estimation problem has been studied earlier in the literature (see [2] and references therein). Our filter is an extension of Bhattacharya et al's Linear Proportional Response Filter[2] (LPRF). The *LPRF* is a filter which passes loss indications (LIs) with a probability $\frac{L_i}{\Sigma_i L_i}$ where L_i is the number of LIs from receiver i.

We found that we could not use the LPRF filter directly because of a number of issues. Firstly, it was originally presented under Markovian model assumptions [2] which do not hold under high multiplexing conditions (eg: see section 3). Therefore we found that it was susceptible to the drop-to-zero problem. Second, *LPRF* did not quickly adapt to sudden increases in worst-case loss rates. For example, consider a

[4] $SRTT =$ average of RTT samples similar to the variable used in TCP

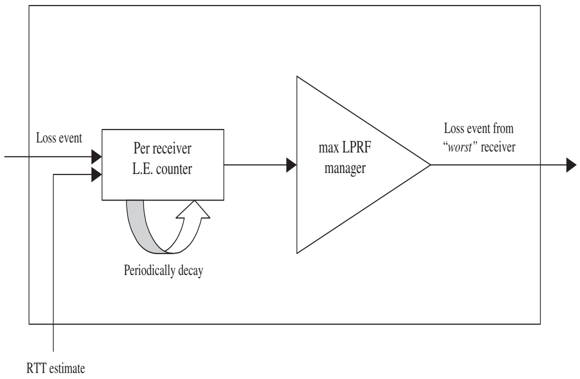


Figure 3: Max-LPRF: max-Linear Proportional Rate Filter

steady state case with a large number of receivers, where each receiver has a loss rate of 1%. If one of the receivers suddenly starts experiencing a loss rate of 10%, then *LPRF* converges to the "worst" receiver very slowly. Third, *LPRF* works with LIs instead of LEs. Our subsequent arguments show that using LEs is much superior to using LIs.

The *Max-LPRF* works as follows: Assuming X_i is the count of LEs from receiver i, this probabilistic filter takes as input all the LEs from receivers ($\Sigma_i X_i$) and on an average passes the maximum number of LEs from any one receiver (i.e. $\max_i X_i$). In particular, it *passes each LE with a probability* $\frac{\max_i X_i}{\Sigma_i X_i}$. The LE counts per receiver (X_i) are decayed periodically by 10% every 100 $SRTTs$. The O(N) state requirements of both *LI2LE* and *Max-LPRF* are not a big issue because single-rate schemes are typically targeted for a small-medium scale (< 10000 receivers). *Max-LPRF tracks the worst path better than LPRF and is the crucial building block for drop-to-zero avoidance.* It operates on per-receiver LE counts since they differ dramatically from LI counts in drop-tail queueing networks with no self-clocking.

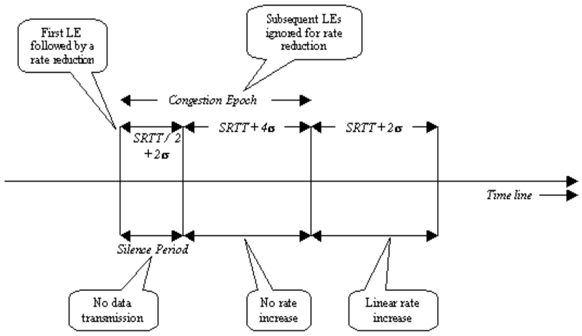


Figure 4: Adaptive Time Filter (ATF)

Adaptive Time Filter (ATF): This filter shown in Figure 4 is a modification of the time-filter introduced in our earlier work [16]. This filter simply drops excess LEs passed by Max-LPRF in any RTT to enforce *at most one rate reduction (RR) per* $SRTT + 4\sigma$. In addition, the filter also imposes a *"silence period"* of $\frac{1}{2}(SRTT + 4\sigma)$ when no packets are sent. As suggested in [16], the goal is to reduce the probability of losing any control traffic or retransmissions during this phase.

All filters and the AIMD module need RTT estimates which is fed by the RTT estimation module developed in our earlier work [16]. It works similar to the TCP timeout procedure i.e. it calculates a smoothed RTT ($SRTT$) and a mean deviation which approximates the standard deviation σ. However the set of samples is pruned to exclude a large fraction of samples which are smaller than $SRTT/2$ (i.e. smaller by an order of magnitude) to bias the average RTT higher. The rate increase uses intervals of length $SRTT + 2\sigma$ (see [16]) while other functions differ in their use of σ as described earlier. Observe that this LI-driven RTT estimation procedure will tend to opportunistically measure the worst RTT of paths which are generating more LIs. That is, the procedure tends to estimate the worst RTT from the *currently congested sub-tree*. Therefore, the de-coupling of RTT estimation from worst-case loss rate estimation does not lead to significant suboptimality in practice (unlike what a naive analysis of the TCP-throughput formula [17] would predict).

In summary, for ideal operation, the scheme expects: (a) at least one LI per receiver per RTT if packets are lost in that RTT (i.e. at least one LE per RTT) and (b) timely generation of feedback by receivers/network elements to allow reliable RTT sampling [5]. In general, if expectations (a) and (b) are not satisfied completely or satisfied in an unreliable manner, the scheme performance will degrade. In particular, the scheme is sensitive to arbitrary delays introduced by receivers/network-elements in generating an LI corresponding to a lost packet. Further *LI aggregation* like in PGM [22] itself is a form of filtering which suppresses receiver IDs, timing information and reduces LI or LE counts, thus conflicting with (a) and (b) above[6]. We examine such performance effects in the following sections. The pseudo code for the scheme is presented in Appendix A.

3 Performance Evaluation

We have evaluated the performance of LE-SBCC to test for Drop-to-Zero avoidance, TCP-friendliness and LI aggregation effects. We use the following methods to test our scheme:

1. *ns-2* simulations to observe the detailed scheme dynamics and background TCP dynamics for tens of receivers.
2. Simple Markov chain based modeling to obtain a better understanding the choice of LEs as opposed to LIs in our scheme.
3. Linux-based implementation/experimentation (high multiplexing degrees and up to tens of receivers) to understand implementation issues and test performance on a real network.
4. Simple C-program to numerically simulate the scheme using a model based on fixed loss probabilities of receivers, fixed RTT and packets assumed to be sent in rounds of length RTT. This simple analysis captures the first-order behavior of the scheme and allows us to explore its performance for large numbers of receivers (up to 100-10,000). We will refer to this method as *Numerical analysis* in the rest of the paper.

3.1 Drop-to-Zero Avoidance

Recall that Drop-to-Zero Avoidance is the problem of reacting to more loss indications (LIs) than necessary leading to a beat-down of the multicast flow's rate. Drop-to-Zero problem occurs in the following scenarios :

1. High multiplexing, where many flows share a common bottleneck; and the multicast flow receives independent feedback from several such paths. In such a case, the loss rates observed at the source are independent of sending rate, and the multicast flow receives feedback from several paths which requires filtering.
2. Large receiver sets result in a huge number of LIs. It becomes important that the scheme does not react to more LIs than that generated by the max loss rate receiver.
3. LI aggregation leads to degradation of loss and timing information as some LIs are suppressed by the intermediate aggregator. Aggregation effects are discussed in a later section.

We have analyzed the drop-to-zero issue using *ns-2*, numerical simulations and Markov Chain based theoretical analysis.

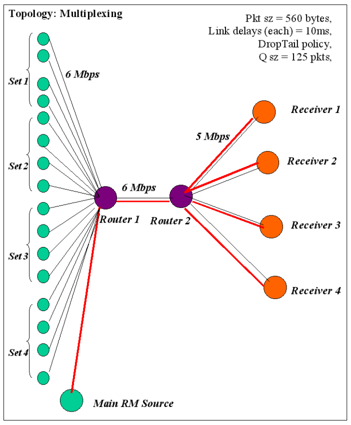


Figure 5: Topology to Test Drop-to-Zero Avoidance and TCP-friendliness

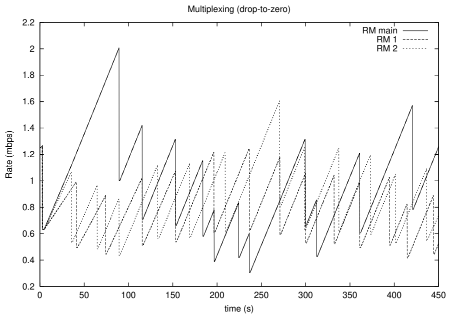


Figure 6: Drop-to-zero Avoidance Results

3.1.1 *ns*-2 Simulations Illustrating Drop-to-Zero Avoidance

Consider a set of reliable multicast (RM) flows that goes to receiver 1 through receiver 4 as shown in Figure 5. The capacity of the links connecting the sources (on the left) to “Router 1” are 6 Mbps each. The link from “Router 1” to “Router 2” has 6 Mbps for each flow. There is a *Main RM* source which sends data to receivers 1 through 4. The links from “Router 2” to each of the receivers are bottlenecks at 5 Mbps each. A set of 4 background unicast flows implementing the LE-SBCC algorithm compete on each of the 4 bottlenecks links. The *unicast* flows in “Set 1” through “Set 4” compete on the bottleneck links to “Router 2-Receiver 1” through “Router 2-Receiver 4” respectively. The fair rate for all flows is therefore 1 Mbps. The buffer size (125 pkts) is roughly twice the bandwidth times fixed delays. The packet size is 560 bytes. This topology tests the performance of the multicast congestion control scheme under independent loss rates on paths, and reasonable degrees of multiplexing on each path. In effect, the loss rate on such congested paths is less dependent on the rate change of any single flow on the path (especially the multicast flow).

Figure 6 (rate graph) shows that the multicast flow rate (solid line) competes fairly with a selected subset of unicast background flows (dotted lines) and the average rate is close to 1 Mbps, the fair share. More importantly, the multicast flow does not get beaten down because of the independent loss indications received from each path. This is a non-trivial illustration of drop-to-zero avoidance. Observe that the same topology also illustrates the basic applicability of LE-SBCC to multi-sender multicast, because the competing unicast flows use the same congestion control scheme. Each sender’s multicast traffic is controlled as if it were the only sender.

The analysis of the number of packets transmitted, the number of LIs and LEs and rate reductions (RRs) also illustrates the impact of the cascaded set of filters. In particular, the main RM flow transmitted a total of 167762 pkts in 450s and received 949 LIs (i.e. a low average LI rate). After LI2LE filtering, there were only 32 LEs, a

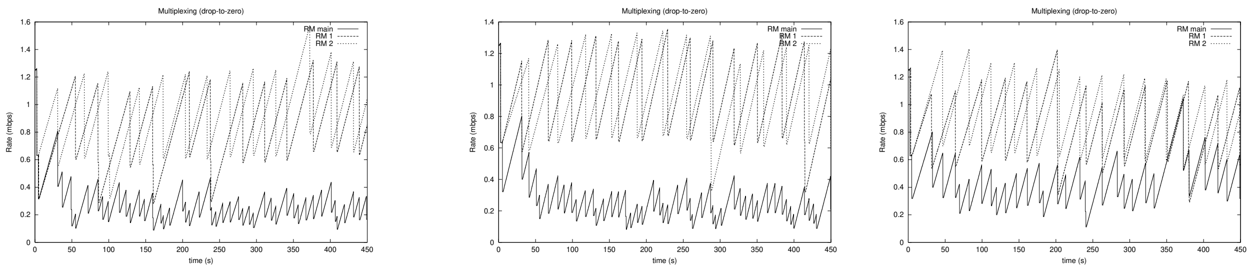
[5] Within these constraints the scheme may be successfully applied to unreliable multicast transport protocols

[6] LIs carry RTT information which affects the RTT estimation, and LEs carry receiver IDs which affects the max-LPRF filter.

dramatic reduction! The Max-LPRF and ATF further filter LEs to result in a total of 12 rate-reductions. When we go back to the graph in Figure 6 and count the number of rate-reductions in multicast as well as unicast flows, we observe that all flows have 12 rate-reductions. This is another measure of the fairness of the scheme to AIMD-based background traffic. The issue of TCP fairness is further explored in section 3.2.

As discussed in the next section, the dramatic difference between the number of LIs and LEs is because of bursty losses caused by the lack of self-clocking (unlike TCP), rate-based nature of control and the use of sizable drop-tail buffers.

3.1.2 Effects of Removing a Filter From the Cascade



(a) Removing both LI2LE and maxLPRF (b) Removing maxLPRF (c) Removing LI2LE

Figure 7: Effects of Removing a Subset of Filters from the LE-SBCC Cascade

A natural question is the relative importance of each filter in the LE-SBCC's cascade of filters. To further demonstrate the important role that each filter plays in our scheme, we repeat the simulations for the configuration shown in Figure 5 for three different cases:

(a) Removing both *LI2LE* and *maxLPRF* filters

(b) Removing only the *maxLPRF* filter, and

(c) Removing only the *LI2LE* filter.

In all these experiments all other parameters are kept constant as before. The results are illustrated in Figure 7. Clearly, we observe the drop-to-zero problem to some degree in all cases (solid curves always below dotted curves). The results are also affected by the fact that in some cases the max-LPRF operates on LIs. In these cases, the max-LPRF processes a larger absolute population of samples, and deals with a different relative distribution of counts from receivers (which affects the critical ratio: $\frac{\max_i X_i}{\Sigma_i X_i}$). Thus each filter in the cascade has a key role in filtering of LIs and avoiding the drop-to-zero problem.

3.1.3 Theoretical Analysis of LE vs LI Probability

Recall that our scheme uses LEs instead of LIs for the max-LPRF filter and ultimately in rate reduction decisions. Therefore, our scheme is critically dependent on the number of LEs passed by filters at the source. Thus it becomes important to study the relationship between LIs and LEs to get an understanding of when it is really important to use LEs instead of LIs.

This section presents a simple theoretical analysis to show that using LEs in general leads to a reduced probability of drop-to-zero because the number of LEs is always less than the number of LIs. Moreover, under bursty loss conditions, the number of LEs is significantly less than the number of LIs. This justifies the presence of LI to LE filter used in the cascade. Also, observe that the ratio $\frac{\max_i X_i}{\Sigma_i X_i}$ (used in MaxLPRF) could be very different if X_i represented the count of LEs vs if it represented the count of LIs. These two reasons justify the use of LEs instead of LIs.

Consider the following Markov Chain model for unicast transmission. Assume that the RTT is constant and that the system moves in cycles of RTT. In each RTT, the source transmits with a constant rate, and may experience one or more packet losses. At the end of each RTT, it changes its rate based upon the AIMD policy and depending upon the receipt of LEs. Therefore the system may be modeled as a Markov chain with N discrete states (from 1 to N), where the state x specifies the number of packets sent during that RTT. Therefore, $x = Rate(x) \times RTT$. The process starts in state 1, and may go up to the maximum allowable rate (which corresponds to state N). When a loss event (LE) is encountered within an RTT, the system transitions to state $\lfloor x/2 \rfloor$. When there is no loss event in an RTT, the system moves to state $x + 1$. This Markov chain model is illustrated in Figure 8. The goal of this Markov chain is to estimate the ratio of per-packet LI probability (p_{li}) to per-packet LE probability (p_{le}), i.e., $E(p_{li})/E(p_{le})$. We analyze this chain for two cases: the non-bursty and bursty packet loss case.

- **Non-bursty Loss Case**

  We assume that the packet loss probability is uniform. Define p: the probability that a packet is lost, and a: 1 - p. In this case $p_{li} = p$. In state x, the probability of having at least one loss is $1 - a^x$. The LE probability (p_{le}) is thus $\sum s(x)(1 - a^x)$ where $s(x)$ is the probability of being in state x.We have five cases to consider :

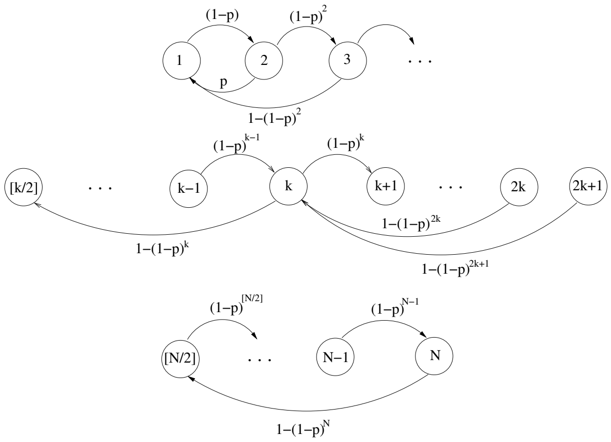


Figure 8: State transition of packets sent per RTT

For non-bursty loss, $p_{li} = p$; for bursty loss, p is the probability that a packet is lost and it is the first loss after the most recent RTT with a loss.

1. We are in state 1. We could increase the rate and move to 2. We could also get losses at states 2 or 3, and move to 1. We could also get a loss, and stay in 1. In steady state,

$$p(1) \times a = s(2) \times (1 - a^2) + s(3) \times (1 - a^3)$$

2. We are in any state x between 2 to $m - 1$. This is the general case mentioned above. In steady state,

$$s(k) = s(k-1) \times a^{k-1} + s(2k) \times (1 - a^{2k}) + s(2k+1)(1 - a^{2k+1})$$

3. We are in state m. We could increase the rate and move to $m + 1$. We could also get loss at state N only, and move to m. In steady state,

$$s(m) = s(m-1) \times (a^{m-1}) + s(N) \times (1 - a^N)$$

4. We are in any state x between $m + 1$ to $N - 1$. The only difference between this state and 3 is that we cannot enter this state by a rate reduction, as the $N = \max(rate \times RTT)$. In steady state,

$$s(k) = s(k-1) \times a^{k-1}$$

5. We are in state N. We cannot increase the rate further. The only way to leave this state is by cutting its rate, and moving to state m. In steady state,

$$s(N) \times (1 - a^N) = s(N-1) \times a^{N-1}$$

This Markov chain cannot be solved for an exact solution. However, we did a Matlab analysis to numerically solve the chain for different large N's. The solutions yield the steady state probabilities $s(x)$ and hence the probability p_{le}. We then plot the ratio of LI probability p_{li} and LE probability p_{le} ratio in Figure 10 (a). The plot shows that there is not much difference between LI and LE probability. In particular, the maximum ratio of LI to LE probability is less than 1.25. In other words, the number of LEs is at most 25% smaller than the number of LIs (which happens when the LI rate is 30-40%, a large absolute loss rate).

The uniform loss probability assumption made above is not valid in bursty packet loss scenarios (though it could apply to RED gateways). The combination of our scheme with a drop-tail buffer leads to a very bursty packet loss pattern, which yields a very different picture of LI vs LE probability.

- **Bursty Loss Case**

  Our scheme uses a rate-based AIMD policy, which is not self-clocked (unlike TCP). This lack of self-clocking is because the scheme depends only upon LIs and not a stream of acks for its basic operation. A simple control-theoretic analysis [18] shows that a rate-based scheme with linear rate-increase, and lack of self-clocking leads to a bottleneck queue which grows quadratically. Therefore, if the bottleneck has a larger buffer, the queue increases quadratically for a longer period of time. Moreover, when the buffer is full and packets are about to be dropped, the sending rate differs significantly from the optimal transmission rate. So, when the queue overflows, the number of packets dropped corresponds to (R - C)T where R is the aggregate transmission rate (which is much larger than C) and C is the bottleneck capacity, and T is the round trip time (RTT) including the queueing delay. This leads to a large burst of packets dropped during the RTT before the receivers send LIs and the sources detect congestion. Therefore it is important to understand how the number of LEs differ from the number of LIs under such bursty loss conditions. We show that under such conditions, the choice of LEs is even more justified.

  We use the Markov chain model as used earlier, albeit with some changes. To approximate the bursty loss behavior, we assume that if a packet is lost during any instant within the RTT, all remaining packets transmitted during that RTT are also lost. This assumption is also made by Padhye et al [17] for TCP throughput modeling. Under this assumption, the probability p assumes different semantics. p here is defined as the probability that a packet is lost given that either it is the first packet in the RTT or the preceding packet in the RTT is not lost. Assume like before that $a = 1 - p$.

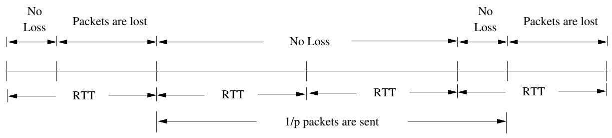


Figure 9: p's semantics in bursty loss case

In this case, $p \neq p_{li}$ (see Figure 9). However, the overall real LI probability p_{li} can be derived from p. First, recall that the state x specifies the number of packets transmitted during a certain RTT, $x - i$ is the number of lost packets in that RTT. Under our assumption, during a RTT,

$$P(\text{The first i packets are not lost and the subsequent packets in the round are lost}) = (1-p)^i p$$

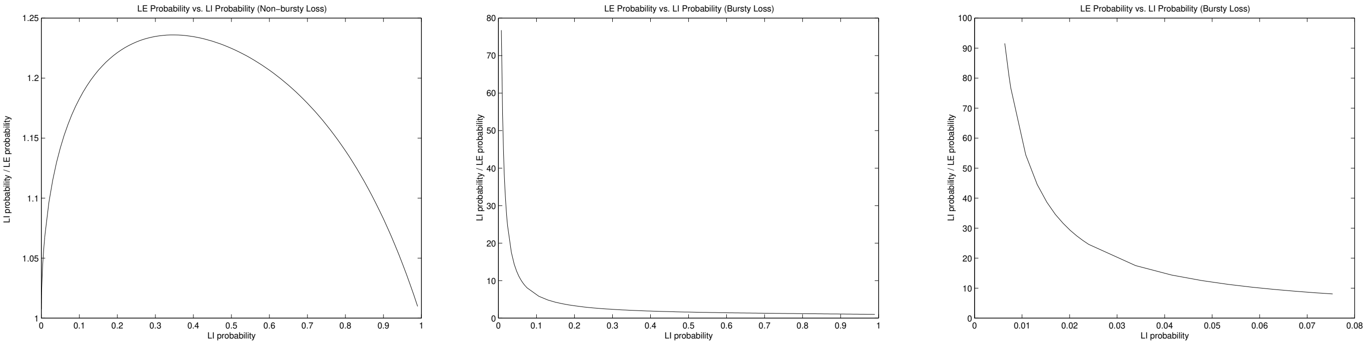
Therefore, at state x, the expected number of lost packets is $\sum_{i=0}^{x}(x-i)(1-p)^i p$. The LI probability p_{li} is the expected number of lost packets divided by the expected total number of packets transmitted:

$$p_{li} = \frac{\sum_{x=0}^{\infty} s(x) \sum_{i=0}^{x} (x-i)(1-p)^i p}{\sum_{x=0}^{\infty} x \cdot s(x)}$$

At state x, the expected number of loss events (LE) is the probability that there is at least one loss, that is $1-(1-p)^x$. Therefore,

$$p_{le} = \sum_{x=0}^{\infty} s(x) \cdot (1-(1-p)^x)$$

As a result, ratio of LI and LE probabilities ($\frac{p_{li}}{p_{le}}$) is substantially altered. This is illustrated in Figure 10 (b). The figure plots the ratio as p_{li} ranges from 0 to 1. The ratio $\frac{p_{li}}{p_{le}}$ is huge when p_{li} is small (which is the common case of network operations). This is because the small overall LI probability leads to concentrated burst loss in few RTTs leading to a small number of LEs in comparison to LIs.



(a) Non-bursty loss (b) Bursty loss (All loss probabilities) (c) Bursty loss (Low loss probabilities)

Figure 10: Ratio of LI and LE probabilities

For lower loss probabilities, the p_{li}/p_{le} values are higher. To see the relation between LI and LE more clearly in this region, we have plot another graph for the low LI probability region (Figure 10 (c)). We observe high p_{li}/p_{le} ratios (9-90). Comparing these numbers to the numbers of LIs and LEs observed in simulation (see section 3.1.1), we observe that the model gives the same order of magnitude of results as the simulation results. In that case $p_{li}/p_{le} = 949/32 = 29.7$. Therefore, we can conclude that the LI2LE filtering is critical in our cascade filter design.

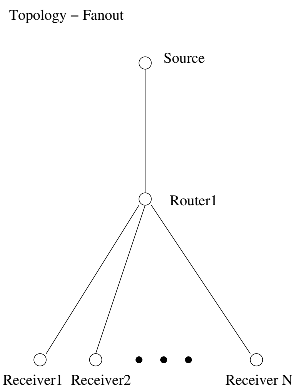
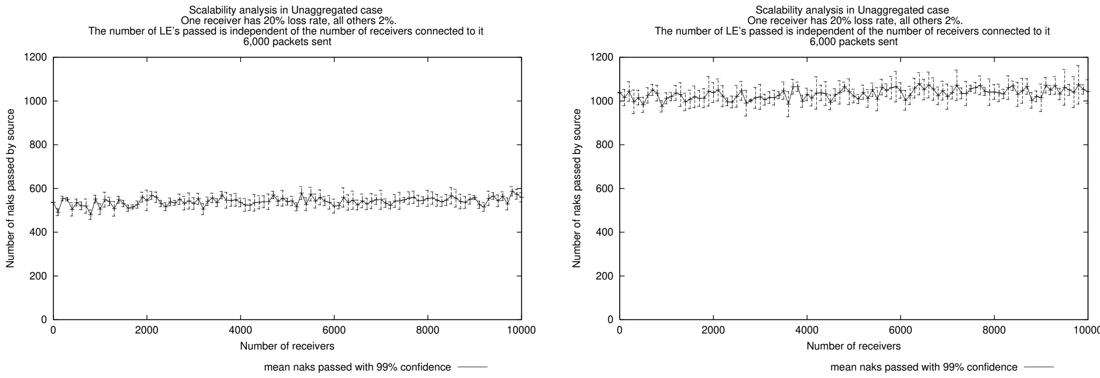


Figure 11: Topology : Fanout

3.1.4 Drop-to-Zero Avoidance in Medium-Large Scale Trees

Recall that large receiver sets result in increased number of LIs being generated. To test the performance of our scheme in such scenarios we gradually increase the number of receivers in the multicast session, and run the numerical simulation for each addition. The numerical simulator implements all the filters, but assumes constant RTTs and that each path experiences a uniform loss probability. We use a fanout(Figure 11) topology where all the receivers and the single source are connected to the same router. One of the receivers in the session has the worst loss rate, and the rest have lower loss probabilities. We expect that the number of LEs passed by source for rate reduction should remain fairly constant and equal to the number of LEs generated by the worst loss receiver. Since it was not possible to scale beyond 10's of nodes in a *ns* simultion, the scalability of drop-to-zero avoidance in non-aggregated scenarios is numerically analyzed.

The number of receivers in the multicast tree is increased from 1 to 10001 in a gradual manner. We assume one receiver has 10% loss, all others have 2% loss and study the effect of increasing the number of 2% receivers. The above numbers and topology were chosen as they represent a general case where one of the receivers (the 10% one) clearly has the worst loss rate. The expected number of LEs passed by the filters should be almost equal to the number of LEs generated by the worst loss receiver. The number of LEs passed for rate reduction by the source is plotted against the number of receivers(Figure 12(a)). The experiments were repeated for different random seeds and confidence intervals are plotted. Under no aggregation, the number of LEs passed by source remains fairly constant(500-520). The number of LEs passed are consistent with the non-bursty theoretical analysis performed above. The expected number of LIs by the 10% receiver would be $0.1 \times 6000 = 600$. By referring to the LI probability(p_{li}) vs LE probability(p_{le}) graph(Figure 10 (a)) for the non bursty model, we see that for 10% loss p_{li}/p_{le} ratio is around 1.2. Therefore the expected number of LEs would be $600/1.2 = 500$. This number agrees very closely with the numbers obtained by numerical simulation. We also repeated the same experiment with the max loss receiver having a loss rate of 20%. Again, the number of LEs passed (Figure 12(b)) remains fairly constant (around 1000). This indicates that under same RTT, fixed uniform loss probabilities, and no aggregation conditions our scheme passes a fairly constant number of LEs even when large number of receivers are present. This number is equal to the LEs generated by the worst loss receiver. Thus we achieve Drop-to-Zero avoidance in medium-large scale trees.



(a) 10% Max loss receiver

(b) 20% Max loss receiver

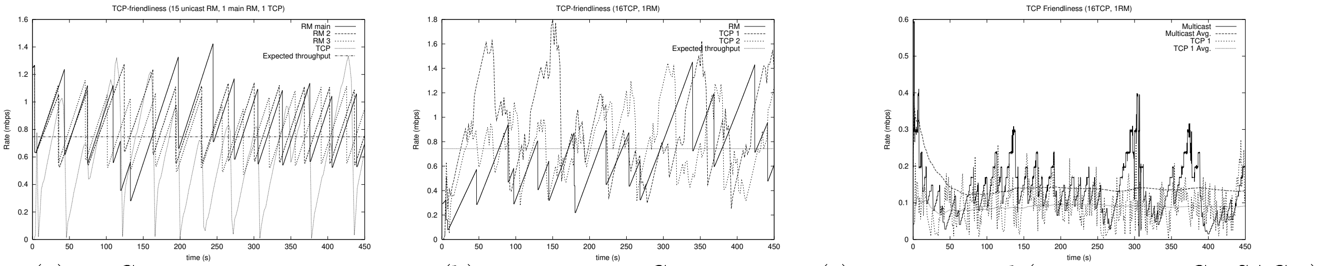
Figure 12: Numerical analysis: Scalability

3.2 TCP-friendliness

TCP-friendliness is a subject of current debate [8, 20, 12]. A good measure of TCP-friendliness would be the average throughputs of the competing TCP and RM flows. To illustrate TCP-friendliness, we use two topologies, a) where a common bottleneck is shared by many RM flows and a single TCP flow and, b) many TCP flows and a single RM flow. LE-SBCC would be TCP-unfriendly if either of the TCP or RM flow is beaten down. In Figure 5, consider two cases: case (a) when *one* of the flows of “Set 4” is replaced by a TCP Reno flow while the rest are LE-SBCC flows; and case (b) when *all* the flows in “Set 1” through “Set 4”) are replaced by TCP flows.

***ns* simulation results:** The simulation results for these two cases is shown in Figures 13(a) and 13(b) [7]. The dips by the TCP flow close to zero represent timeouts. Again observe that the main RM flow (solid line in both graphs) shares the bottleneck(s) fairly with the competing TCP flow(s). This becomes clearer by observing that the rate graphs for the RM and TCP flows oscillate about roughly the same mean and with the same variance. The horizontal dotted line is the expected TCP throughput plotted post-simulation using the simplified TCP equation ($1.22MSS/RTT\sqrt{q}$). Recall that q is the LE probability and its value is measured here by counting the number of LIs passed by the LI2LE filter. Also recall that MSS is 560 bytes and the RTT is the smoothed RTT measured by the *RTT estimator module* of our scheme.

Experimentation results: The scheme is implemented in a Linux 7.0 testbed. The topology used is identical to the one used for the *ns* simulations as described above. Again the RM flows are fair to the competing TCP flow (Figure 13(c)). More details on this are presented in the section on implementation issues.



(a) 1 TCP Reno vs 16 RM (b) 1 RM vs 16 TCP Reno (c) Experimental (1RM vs 16 TCP SACK)

Figure 13: TCP Friendliness (Simulation and Experiment)

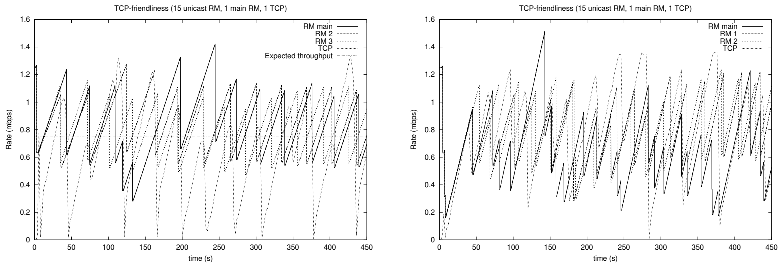
For the purpose of having a measure of TCP friendliness, we aim to reduce the multicast problem to a unicast one with the “worst” receiver being the only one with LE probability q and a worst RTT. The “worst” receiver is probabilistically determined. It is worst in terms of q, and may change over time in general. Then using the simple throughput formula for unicast proposed by Mathis et al ($1.22MSS/RTT\sqrt{q}$), we do a post-processing of the simulation data to compare the throughput of our scheme with the theoretical throughput. Clearly from the Figures 13(a) and 13(b), the rate values oscillate about their mean shown by horizontal dotted line. Their mean rate is close to the theoretical mean. The fact that the average throughput of the RM flows is equal to the unicast flows is a good measure of the TCP friendliness of our scheme. This further demonstrates the validity of choosing LE rather than LI probabilities in our scheme.

3.2.1 DropTail vs. RED Queues

We evaluated the TCP-friendliness performance of our scheme with the two queue management policies - *Tail Drop* and *Random Early Detection (RED)*. These are the two widely used queue management policies and it is important to test the performance of LE-SBCC with both the policies. The topology used for this purpose is the same as in the above experiment with changes in the following key parameters: TCP version: *Reno*, Queue size = *125 packets*, Packet size = *560 bytes*.

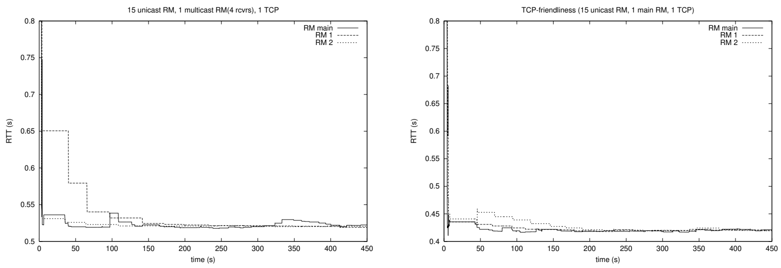
As we can see from Figure 14, the throughput is marginally lower with RED than with DropTail and so is the measured RTT (Figure 15). (The absolute RTT values are smaller in RED, because when losses occur, RED has lower average queue length.) Also observe that the RTT converges much faster with RED than DropTail. This is because the RED policy drops packets before the queue is full resulting in more frequent and timely loss indication from the receivers. The scheme remains fair towards the competing TCP flow in both cases and we do not observe any drop-to-zero with RM flows.

[7] To avoid cluttering, the TCP throughput is sampled every 2s



(a) DropTail (b) RED

Figure 14: Droptail vs. RED: Rate graphs

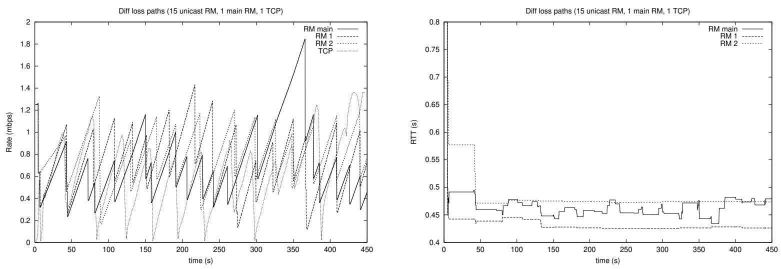


(a) DropTail (b) RED

Figure 15: RTT convergence - Droptail vs. RED

3.2.2 Different Loss Paths

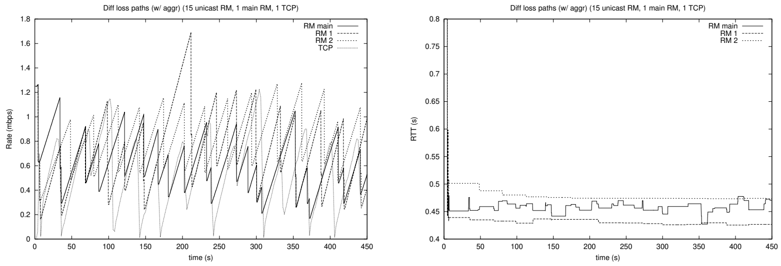
It was speculated that the scheme would not perform as well with the multicast tree having paths with varying degree of loss probabilities and RTTs. We perfomed the following simulation to test the robustness of our scheme under these heterogeneous conditions. Again the topology remains the same as the above experiment with modifications in following key parameter: The buffer is 125 packets on all links except that it increases on the four links from the router to receiver 1, receiver 2, receiver 3 and receiver 4 with the values 25, 50, 75 and 100 respectively.



(a) Rate (b) RTTs

Figure 16: Rate and RTT graphs for topology with different buffers (loss rate and RTTs)

Having different buffer sizes is an indirect way to affect RTT and loss probabilities on a link. The link from router to receiver 1 has the smallest buffer (25 packets). Therefore, it has the smallest RTT and the largest loss rate. On the other hand, the link from router to receiver 4 has the largest buffer (100 packets). So it has the largest RTT and smallest loss rate. For the *Main Source*, MaxLPRF will respond to LEs from receiver 1, whereas its RTT would be biased higher (average of the 4 different RTTs measures received from the 4 receivers). The simulation is performed for both non-aggregation (Figure 16) and aggregation cases (Figure 17). The graphs clearly demonstrate that the algorithm works well under heterogeneous loss probability and RTT conditions and with aggregation enabled at the router too.



(a) Rate (b) RTTs

Figure 17: Rate and RTT graphs for topology with different buffers (loss rate and RTTs) [With aggregation]

3.3 Loss Indication (LI) Aggregation Effects

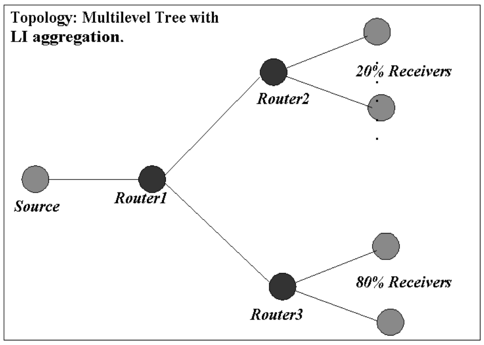


Figure 18: Topology to Illustrate LI Aggregation Effects

As explained earlier, LI aggregation has an effect on the number of LEs and timing information reaching the source. To evaluate these effects, consider the simple multi-level topology in Figure 18. Observe that the number of receivers and their loss rates can be varied. Consider three cases: (a) *no LI aggregation* performed at the routers (b) LI aggregation performed at router 2 and router 3 only (i.e. *partial LI aggregation*) (c) LI aggregation performed at all routers (i.e. *complete LI aggregation*).

Since *ns-2* simulation does not scale beyond tens of nodes, we simulate the scheme numerically. In Figure 18, we had 10% of receivers in the subtree headed by router 2, and 90% of receivers in the subtree headed by router 3. One receiver in the first subtree has a loss (LI) rate of 4% and all other receivers (in both subtrees) have a loss rate of 2%. A total of 6000 packets were transmitted which translates to a max-LI count of 240 and the observed max-LE count is roughly (210-225) (without aggregation).

The numerical simulation for cases (a), (b), (c) above are shown in Figure 19. The figure illustrates the LEs passed by the cascade of filters (one curve for each case), plotted against the number of receivers (up to 200 receivers). The experiment was repeated for different random seeds and confidence intervals are plotted. Except for the completely aggregated cases, each curve is roughly flat which means that the Max-LPRF passes roughly the same number of LEs irrespective the number of receivers or the total number of LEs generated. Two curves (no aggregation and partial aggregation) are nearly identical which shows that the scheme works well even with

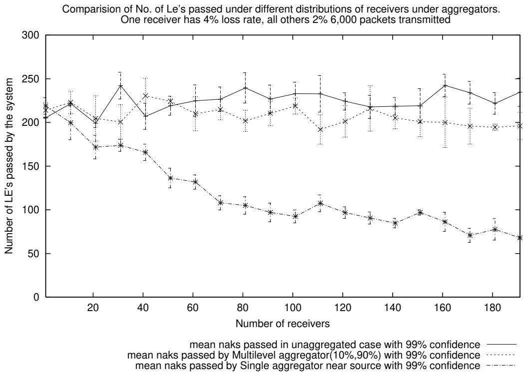


Figure 19: LI Aggregation Effects: LEs Passed by Cascade of Filters

non-trivial partial aggregation. This is because the max-LPRF scheme is *more sensitive to LE counts and receiver IDs rather than LI counts/IDs*. However, for the case of complete aggregation, the number of LEs passed is lower, which implies that the scheme will be more aggressive than an equivalent TCP. *In all cases, it does not incur the drop-to-zero problem*, i.e. passing more than 240 LEs.

3.3.1 Worst Case Aggregation Analysis

Recall that LI aggregation has an effect on the number of LEs and timing information reaching the source. Under certain aggregation conditions most of the LEs generated by the worst loss receivers could be suppressed. Since our scheme is sensitive to LEs, such suppression could lead to a degradation of performance. We analyze the worst case for aggregation numerically, in terms of LEs reaching the source. The fanout topology (Figure 11) is used for the analysis, where the source and all receivers are connected to the aggregator. The worst case of aggregation occurs when one of the receivers is clearly the worst loss receiver and the LIs generated by worst loss receiver always gets suppressed by the aggregators. This is worst case because the source would not get any congestion indication from the worst loss receiver, and just react to receivers with lower loss rates.

As an concrete example of worst case of aggregation : assume one receiver X has a fairly high loss rate, and all others have very low loss rates. Ideally, the number of LEs reaching through the source(after filtering) should be equal to the number of LEs generated by X. Further assume that X also has the longest RTT. This implies that X's LI would be suppressed, in presence of a LI from any other receiver. We call this situation "Biased Aggregation". As the number of receivers increase, we expect more overlaps between sequence numbers lost by X and other receivers. Fewer and fewer LIs generated by X might reach the source. In the worst cases no LEs generated by the worst loss receiver would reach the source. The maximum number of LEs generated by any single receiver would now become much lower, and thus affects the critical ratio $\frac{\max_i X_i}{\Sigma_i X_i}$ used by Max-LPRF. Hence MaxLPRF would pass lesser number of LEs and the source would be become more aggressive.

We assume one receiver has 20% loss rate, and all others 2%. We study the effect of increasing the number of 2% receivers under the aggregator. The number of LEs passed for rate reduction *should be* approximately equal to the number of LEs generated by the 20% receiver. Any significant change in the number of LEs passed due to the increase of 2% receivers would clearly affect performance. The number of LEs reaching the source vs the number of receivers is plotted(Figure 20). Under "biased" aggregation, the number of LEs passed by source drops sharply by increasing the number of 2% receivers. Under the worst case of aggregation, our scheme becomes aggressive towards TCP. However we do not incur drop-to-zero problems under these conditions.

3.3.2 Partial Aggregation Analysis

In the previous section it was shown that in the worst case of aggregation, the scheme becomes aggressive. However, in real networks many other intermediate cases are possible, which would not always affect performance significantly. We numerically analyze some of these cases. In the following graphs (Figures 21) we show a

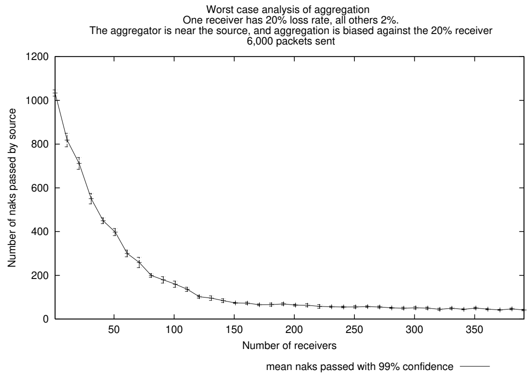
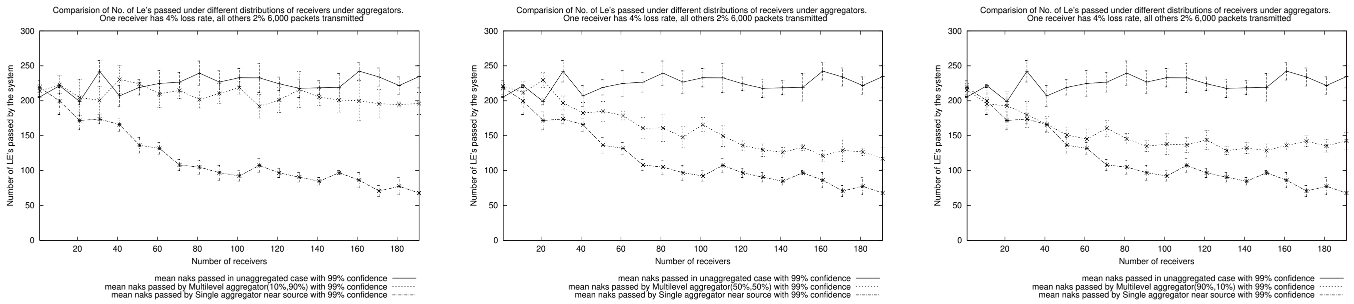


Figure 20: Numerical analysis: Worst case aggregation

complete range of results in case of partial aggregation. The topology is same as in Figure 18. One receiver has loss probability of 4% while all others have 2%. We vary the distribution of receivers under the two aggregators in the multicast tree. The convention followed is $x - y$, where x% receivers (including the receiver with 4% loss probability) are on Router2 and remaining y% are under Router3. The experiments were repeated for different random seeds and confidence intervals plotted. The following plots show the comparison between no aggregation, aggregation near source and varying degrees of partial aggregation. In many cases, even under partial aggregation the performance is close to no aggregation case. It can be seen that in (Figure 21 (a)) the max loss receiver is present in the branch which has lesser number of receivers. Under such cases, the number of LEs passed(210-225) is almost equal to the unaggregated scenario. However in (Figures 21 (b) & (c)) the max loss receiver is present in the branch with higher number of receivers. Under such cases our scheme becomes more aggressive as fewer LEs are passed. The number of LEs passed never drops below the case where a single aggregator(Router1) is present near the source. When only some of the routers have aggregation support, and certain paths remain unaggregated, our scheme performs much better in terms of LEs passed. Especially when the worst receiver does not experience too much aggregation (Figure 21 (a)) the number of LEs passed(around 210) is almost the same as the unaggregated case. Under no situation is the number of LEs passed lower than the fully aggregated situation. In summary, under partial aggregation circumstances, the performance of our scheme can still be relatively good.



(a) 10-90 (b) 50-50 (c) 90-10

Figure 21: Partial Aggregation Effects

4 TFRC module

TFRC [8] is a congestion control policy which provides smooth rate changes, which makes it suitable for audio/video applications. TFRC was originally designed for unicast. But our approach allows the use of the TFRC policy instead of the AIMD policy for rate adaptation. However, we need to modify TFRC to fit it into our framework.

4.1 TFRC Module Design

For AIMD, we used loss events (LEs) to trigger rate reductions. However, for TFRC we need to convert them into a LE rate to update the data transfer rate. TFRC [8] requires the *receiver* to calculate the LE rate and send it back to the sender. In contrast, our scheme is purely source based. There is no such kind of feedback from the receivers. We modify this mechanism so that the LE rate can be calculated at the source. Recall that the output of cascaded filters (Section 2.1) are loss events (LEs). The source can count the loss intervals (the number of packets between consecutive LEs) and calculate the LE rate and therefore the data transfer rate. The details follow.

With the following definitions,

- s_0: the interval since the most recent LE.
- s_i: the i-th most recent loss interval. ($i \geq 1$)
- N: the total number of loss intervals from the beginning of the data transfer.
- n: an integer greater than 0.
- α: $0 \leq \alpha \leq 1$.

the average loss interval $\hat{s}_i$ is calculated as following.

$$\hat{s}_i = \frac{\sum_{k=1}^{n} s_{i+k-1} w_k + \sum_{k=i+n}^{N} s_k \alpha^{k-i-n+1}}{\sum_{k=1}^{n} w_k + \sum_{k=i+n}^{N} \alpha^{k-i-n+1}}$$

In [8], $n = 8$, $\alpha = 0$, w_1 to w_8 have values of 1, 1, 1, 1, 0.8, 0.6, 0.4, 0.2 respectively. In our scheme, the randomness of the MaxLPRF filter can cause larger variance of the average loss interval and therefore more oscillation of the LE rate. To counter this effect, we chose different values for these parameters. In our experiments, $n = 8$, w_1 to w_8 are all 1, and $\alpha = 0.95$. (When there are not enough loss intervals, (1) $N < n$, set w_{N+1}, ..., w_n and α to 0, (2) $N = n$, set α to 0.) They yield satisfying results. Note that this method and the parameter values need not be the only viable set for calculating average loss interval. Other methods and parameter values could be explored.

Whenever a LE is detected, $\hat{s}_1$ is calculated, the reported LE rate p is $1/\hat{s}_1$. Then the data transfer rate T (bytes/sec) is calculated with the following formula [8] and enforced.

$$T = \frac{s}{R\sqrt{\frac{2p}{3}} + t_{RTO}(3\sqrt{\frac{3p}{8}})p(1 + 32p^2)}$$

where s is the packet size, R is round-trip time, p is LE rate, $t_{RTO} = SRTT + 4 * \sigma$.

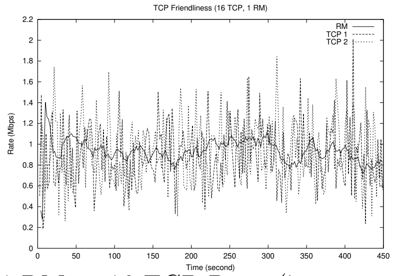
During absence of congestion, after each RTT, $\hat{s}_0$ is re-calculated. The final average loss interval $\hat{s}$ is $\max(\hat{s}_0, \hat{s}_1)$. If it is different from the previous value of $\hat{s}$, the rate T will be updated with the reported LE rate $p = 1/\hat{s}$. If congestion is absent for a long time, $\hat{s}_0$ will become larger and larger, so is $\hat{s}$, and the reported LE rate will drop. Consequently, the rate T will increase, given that the RTT does not vary much.

4.2 Simulation Results

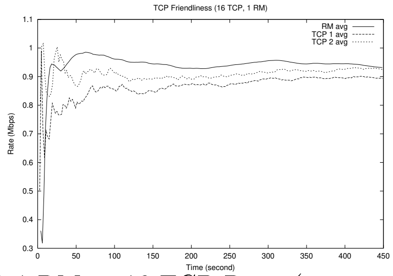
The results presented here show that the LE-SBCC scheme with the TFRC module works relatively well.

We tested the scheme with the TFRC module on the topology shown in Figure 5. The instant rates of multicast flow and two randomly selected TCP flows are shown in Figure 22 (a). Compared to Figure 13 (b), the rate changes are much smoother. Furthermore, the multicast flow is still fair to other TCP flows, as shown in Figure 22 (b) [8].

[8] avg. rate at time t = amount of data sent in period $[0, t]$ / t



(a) 1 RM vs 16 TCP Reno (instant rates)



(b) 1 RM vs 16 TCP Reno (avg. rates)

Figure 22: TFRC module simulation results

In a summary, TFRC as a module can be installed into our framework and work well, demonstrating the modularity of the scheme and its applicability to multimedia.

5 Linux Implementation and Experimentation

For any scheme to be deployable it needs to be implemented and tested in real-world networks. We implemented LE-SBCC on a real network to study implementation issues and performance under real-world conditions. Specifically, we tested TCP friendliness and drop-to-zero problems. Lot of real world issues and limitations not found in simulation had to be tackled. Such issues are discussed in detail in the following sections.

5.1 Implementation

The scheme is implemented on a simplified version of PGM. However, it is not limited to PGM or just reliable multicast.It has been implemented on top of UDP using RedHat Linux 7.0. The following aspects of PGM have been implemented:

- ODATA (original data packet), RDATA (retransmitted data packet), NAK (negative acknowledgement packet), NCF (NAK confirmation packet) packet formats specified in PGM.
- Transmit window at source.
- Receive window at receiver.

By implementing the features mentioned above, we achieved a reliable transport layer. On the other hand, for the sake of simplicity, we omitted certain aspects of PGM:

- SPM (source path message) packets.
- Network elements support.

SPM packets are sent by source periodically to receivers. They carry information about transmit window which affects the generation of NAK's. To compensate this, each packet from source carries the information about transmit window. Whenever a receiver gets a packet (ODATA, RDATA or NCF), it checks the sequence number continuity and sends NAK's if needed.

The program runs as a user process without requiring any root privileges. As a result, the granularity of the timer used is 10 ms which is quite coarse. Coarse timers can introduce unnecessary delays. Assume that an event is supposed to occur at time $t + \delta t$. The timer for it is started at t. If the δt is less than 10 ms, due to the coarseness of the timer, the event will actually occur at t + 10 ms. Then extra delay of $10ms - \delta t$ is introduced. Notice that the extra delay can accumulate. That is, the extra delay introduced by the first event can be added to that introduced by the second event. To reduce such negative effects, the following mechanisms are used:

- After processing any timeout event, the next event is checked against the current time obtained with the API *gettimeofday()*. If the timeout should actually happen, it will be processed. This process continues until no more timeout events are available. Since times returned by *gettimeofday()* have microsecond resolution, the interval between two successive events can be less than 10 *ms*, provided events are processed sufficiently fast.
- When a timeout event is scheduled, its timeout value is checked. If its timeout is within 5 *ms* (half of the timer granularity) from now, it is immediately triggered. This introduces a minor random factor ($< 5ms$) into event scheduling, but reduces the unnecessary accumulative delays mentioned above. In fact, together with the first mechanism, this results in possible bursty events. When the events are for packet transmission, bursty packet transmission occur. The optimal solution for this problem would be to get a system timer of finer granularity.
- We introduce a variant of TBF to control the packet transmission rate at source. Upon arrival of a token, if there are enough packets, the source will transmit as many packets as needed to maintain the required transmission rate. If there are not enough packets, all of them would be transmitted.

By inspecting the time out information, we have observed that the mechanisms described above worked as expected.

5.2 Experimentation

The experiments were conducted on our testbed consisting of RedHat Linux 7.0 systems. During the experiments, the diff-serv package *tc* in RedHat Linux 7.0 were used to restrict bandwidth and link latency. Also, a multicast router daemon *mrouted* [9] was used to route multicast packets from one LAN to others.

The LE-SBCC program runs in user mode subject to OS priority and timer constraints. It cannot send packets at high rates (> 1Mbps). Furthermore, in real experiments, it is very difficult to strictly control certain parameters such as link latency. Thus it is not possible to match the *ns* simulations exactly. However, we show that our scheme is TCP friendly and avoids drop-to-zero in real networks.

5.2.1 TCP Friendliness Experiment

During the TCP friendliness experiment, TCP SACK was enabled. We used TBF queue discipline of a software package *tc* to restrict the bottleneck bandwidth. Details of the configuration are listed as follows:

- Round trip link latency (set by *tc*): 500 *ms*
- Bottleneck bandwidth (set by *tc*): 1Mbps
- Source initial transmission rate: 200Kbps
- Source initial RTT estimate: 200 *ms*
- Receiver maximum random backoff before sending NAK's: 100 *ms*

The fixed part of RTT is the round trip link latency of 500 *ms*. Therefore, any RTT sample is at least of that value. Other parts of delay include hardware I/O delay, router buffer delay, OS processing time and so on. It is very hard to find out the exact delay.

Upon detecting a packet loss, a receiver waits for a certain amount of time before sending a NAK. This time is called backoff time in PGM. A backoff time is chosen randomly from the interval [0, max random backoff] according to uniform distribution. This backoff increases the RTT. The maximum random backoff time is a configurable parameter. When it is large in relation to RTT, it will affect RTT estimation. We chose this time to be $100ms$ so that the receivers have sufficient time to listen for NCF after a loss detection. This helps to avoid duplicate NAK's for the same sequence number from different receivers. At the same time, this does not affect RTT significantly.

[9] Source code and binaries of *mrouted* can be found online at *http://www.vcas.video.ja.net/mice*

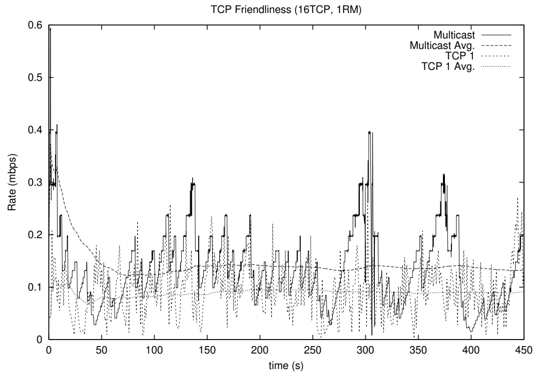


Figure 23: TCP friendliness experiment result (Linux implementation)

For this experiment, we used the topology in *ns* simulation (Figure 5) with the bandwidths and delays changed. All TCP sources and the multicast source ran on one machine. Behind the bottleneck, there were four receiver machines. To achieve load balancing, each of the receiver machine had four TCP receivers and one multicast receiver. The result, shown in Figure 23, roughly agrees with that of *ns* simulation.

In the graph, the average rate of multicast is higher than that of TCP. The reasons can be:

1. The implementation of our scheme is in user space, thus has lower priority and is less reactive to the network situation than TCP which runs in kernel and has higher priority.
2. The timer of our program is coarse, which could make our program less reactive.
3. TCP timeouts are triggered due to small window size, while the multicast scheme does not have such rate-cutting mechanisms.

The TCP flow in the graph was randomly chosen from 16 TCP flows. The results indicate TCP friendliness and drop-to-zero avoidance of our scheme.

6 Summary and Conclusions

We have presented an *LE-oriented, purely source-based multi-sender multicast congestion control scheme (LE-SBCC)* for reliable and unreliable multicast transport. Since the scheme is completely contained at the source, it is simple to implement and deploy.

The model is a simple cascade of filters followed by an AIMD module. The filters together transform the multicast tree to appear like a unicast path for the purposes of congestion control. In the section of theoretical analysis of loss events (LEs) probability versus loss indications (LIs) probability, we justified our choice of LEs for rate reduction consideration. By studying the effects of removing a filter from the cascade, we showed that all filters are important. Several *ns-2* simulation and Linux-based implementation experiments confirm the effectiveness of our scheme.

With partial or no LI aggregation, the scheme correctly filters the LEs (converted from LIs) for AIMD; avoids the drop-to-zero problem while being TCP-friendly. With substantial aggregation, the scheme is likely to be more aggressive towards TCP, but never incurs the drop-to-zero problem. Like all single-rate schemes, its scalability is limited by the heterogeneity of the tree which may depress the source rate.

In general, the AIMD module in our scheme may also be replaced by a source-based TFRC scheme [8] or rate-based binomial scheme [1]. With some modifications, we adapted the TFRC for unicast [8] to our framework. The performance shown by simulation experiments is satisfying. Since TFRC is suitable for audio/video transportation, our scheme can be applied to multimedia.

7 Acknowledgement

Thanks to Supratik Bhattacharyya, Sonia Fahmy, Luigi Rizzo, Mark Handley, Lorenzo Vicisano, Jon Crowcroft and Brian Whetten for extensive comments on various versions of this work.

References

[1] Bansal D. and Balakrishnan H., "Binomial Congestion Control Algorithms," *INFOCOM 2001*, Apr 2001.

[2] Bhattacharyya S. et al, "A Novel Loss Indication Filtering Approach for Multicast Congestion Control," *J. of Comp. Commns*, Feb '01 (to appear).

[3] Bhattacharyya S., Towsley D. and Kurose J., "Efficient Multicast Flow Control using Multiple Multicast Groups," *U.Mass, Amherst, CMPCSI Technical Report TR 97-15*, 1997.

[4] Bhattacharya S., Towsley D. and Kurose J., "The Loss Path Multiplicity Problem in Multicast Congestion Control," *INFOCOM '99*, March '99.

[5] Bolot J.C., Turletti T., Wakeman I., "Scalable Feedback Control for Multicast Video Distribution in the Internet," *SIGCOMM '94*, Aug '94.

[6] Bradner S. et al, "IETF criteria for evaluating reliable multicast transport and application protocols," *RFC 2357*, June '98.

[7] Byers J.W., et al, "FLID-DL Congestion Control for Layered Multicast," *NGC 2000*, Nov '00.

[8] Floyd S. et al, "Equation-Based Congestion Control for Unicast Applications," *SIGCOMM '00*, Aug '00.

[9] Golestani J., "Fundamental Observations on Multicast Congestion Control in the Internet," *INFOCOM 1999*, March '99.

[10] Gopalkrishnan R. et al, "Stability and Fairness Issues in Layered Multicast," *NOSSDAV 1999*, June '99.

[11] Handley M., et al, "The Reliable Multicast Design Space for Bulk Data Transfer," *RFC 2887*, Aug '00.

[12] Handley M., Floyd S., Whetten B., "TCP Friendly Multicast Congestion Control," *RMRG meeting*, June '99.

[13] Kar K., Sarkar S. and Tassiulas L., "Optimization Based Rate Control for Multirate Multicast Sessions," *INFOCOM '01*, Apr '01.

[14] S. Kasera et al, "Scalable Fair Reliable Multicast Using Active Services," *IEEE Network Magazine*, January/February 2000.

[15] McCanne S., Jacobson V. and Vetterli M., "Receiver-driven Layered Multicast," *SIGCOMM '96*, Aug '96, pp. 117-130.

[16] Natu N., Rajagopal P., Kalyanaraman S., "GSC: A Generic Source-based Congestion Control Algorithm for Reliable Multicast," *J. of Comp. Commns*, Feb '01 (to appear).

[17] Padhye J. et al, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," *SIGCOMM '98*, Aug '98.

[18] S. Ramakrishnan, S. Kalyanaraman, J. Wen, H. Ozbay, "Effect of Time Delay in Network Traffic Control," Short Paper, *Automatic Controls Conference (ACC)*, 2001.

[19] Rhee I. et al, "MTCP: Scalable TCP-like Congestion Control for Reliable Multicast," *INFOCOM '99*, March '99.

[20] Rizzo L., "PGMCC: A TCP-friendly Single-Rate Multicast Congestion Control Scheme," *SIGCOMM '00*, Aug '00.

[21] Rubenstein D., Kurose J., and Towsley D., "The Impact of Multicast Layering on Network Fairness," *SIGCOMM'99*, Sept 1999.

[22] Speakman T. et al, "PGM reliable transport protocol specification," *Internet Draft*, March '00.

[23] Vicisano L., Rizzo L. and Crowcroft J., "TCP-like congestion control for layered multicast data transfer," *INFOCOM '98*, Apr '98.

[24] Whetten B. and Conlan J., "A Rate Based Congestion Control scheme for Reliable Multicast," *RMRG meeting*, Jul '98.

[25] Whetten B. and Taskale G., "An Overview of Reliable Multicast Transport Protocol II," *IEEE Network Magazine*, Jan/Feb '00.

APPENDIX

A Pseudo Code

event: Send packet:
    Record $T_{send}[j]$ for seq j
event: LI[i][j] received for from receiver i for seq j:
    call **estimateRTT()**
    call **LI2LEfilter()**
    if passed
      call **maxLPRFilter()**
      if passed
        call **ATFilter()**
        if passed
          $rate = rate/2$
        endif
      endif
    endif

estimateRTT() {
  $RTT_{current} = T_{current} - T_{send}[j]$
  $\delta = SRTT - RTT_{current}$
  $SRTT = RTT_{current} + 0.125 * \delta$
  $\sigma = \sigma + (0.125 * (|\delta| - \sigma)))$
}

LI2LEfilter() {
  if $((t - T_{lastPass}[i]) > SRTT + 2\sigma)$
    $T_{lastPass}[i] = t$
    pass LI as LE
  else
    filter LI
  endif
}
maxLPRFilter() {
  update X_i, $\max X_i$, ΣX_i
  $P(accept) = \frac{\max X_i}{\Sigma X_i}$
  Accept LE w/ probability $P(accept)$
}

ATFilter() {
  if ($CongestionFlag == TRUE$)
    filter LE
  else {
    $SilenceFlag = TRUE$
    $SilencePeriodTimer = RTT/2 + 2\sigma$
    $CongestionFlag = TRUE$
    $CongestionEpochTimer = SilencePeriod + SRTT + 4\sigma$
    Accept LE
  }
}

event: $CongestionEpochTimer$ expires:
    $CongestionFlag = FALSE$

event: $SilencePeriodTimer$ expires:
    $SilenceFlag = FALSE$

event: $RateIncreaseTimer$ expires:
    if ($CongestionFlag == FALSE$)
      $rate += \frac{MSS}{SRTT+2\sigma}$
    else {
      if ($SilenceFlag == TRUE$)
        no data transfer
      else
        no increase in rate
    }
    $RateIncreaseTimer = RTT + 2\sigma$