Technical Report: *LE-SBCC: Loss-Event oriented Source-based Multicast Congestion Control*

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

Abstract

We propose a *loss-event oriented, purely source-based single-rate* multicast congestion control scheme (LE-SBCC) for reliable or unreliable multicast transport protocols. The scheme is easy to deploy and addresses the *drop-to-zero* and *TCP friendliness* problems [15, 18]. It consists of a simple cascaded set of filters and a rate-based AIMD module. Loss indications (LIs) from receivers are converted into loss events (LEs) [6]; the maximum LEs from any path are chosen, and at most one rate-reduction per round trip time (RTT) is effected. The scheme performs well in the common case of *partial LI aggregation*[2], though the worst case in the rare instance of *complete LI aggregation* could be aggressive towards TCP. A range of results (numerical, simulation and experimental) is presented including the mathematical modelling of the LE vs. LI probability. The scheme has been implemented using PGM as the underlying RMT protocol over a Redhat 7.0 Linux testbed.

1 Introduction

This paper presents a single-rate loss-event oriented *purely source-based* congestion control scheme (LE-SBCC) for reliable or unreliable multicast transport protocols. The scheme can function without any *new* support from receivers, network elements or packet-header support and can leverage the underlying loss indications (LIs) provided by protocols like PGM [16] or RMTP [19]. LE-SBCC is a single-rate scheme like PGMCC [15], LPRF [2, 3], Equation-based TFMCC [9, 18], MTCP [14] and Golestani [7]. In contrast to LE-SBCC, these schemes require *new* support at the source, receivers and transport packet-format level which hinders their easy deployment. Examples of multi-rate schemes (which are not source-based) include RLM, RLC, FLID-DL etc [11, 17, 5, 10].

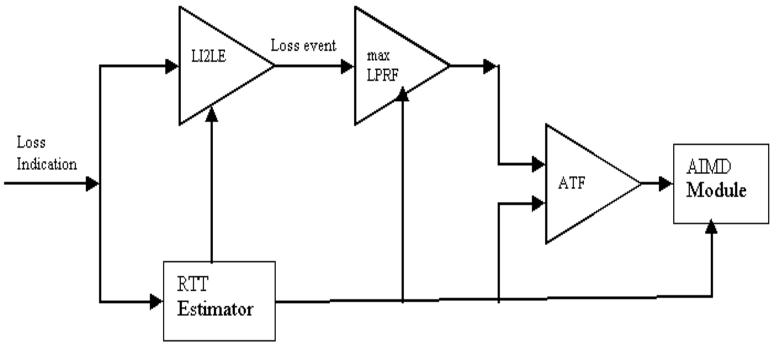


Figure 1: Cascaded Filter Model of LE-SBCC

Our 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). The filters are 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[18, 3]. This occurs because the multicast flow receives LIs from multiple paths and may not filter LIs sufficiently. In an earlier work [12], 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

[1]We acknowledge the help of Jye-Young Song to implement this scheme
[2]Network elements aggregate the reverse data flow to avoid traffic implosion

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* [4, 18, 9]. TCP-friendliness is a subject of current debate[6, 15, 9]. In particular, Padhye et al's TCP throughput formula [13] 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 [15, 9] have receivers estimate RTT and drive the control based upon paths having the maximum value of $(RTT\sqrt{q})$, where q is the *per-packet LE probability.* Our proposed scheme LE-SBCC computes $\max RTT$ and $\max q$ concurently and uses both these estimates to drive the AIMD module. This decoupled use of $\max RTT$ and $\max q$ represents a sub-optimal approach in general[15, 9, 18].

2 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 [6]. The modules work as follows:

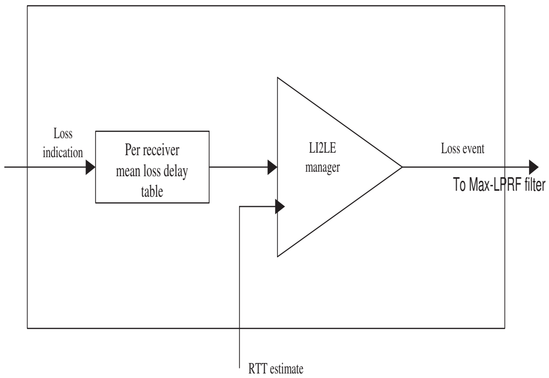


Figure 2: Loss Indication to Loss Event Filter (LI2LE)

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. It 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[3] $T_{LastPassed}$, 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 [15] or Golestani [7]); which leads to large burst losses with drop-tail queues[4]

Max-LPRF: This filter is a extension of Bhattacharya et al's Linear Proportional Response Filter[2]. Assuming X_i is the count of LEs from receiver i, this probabilistic filter takes as input all the LEs from receivers $(\Sigma_i X_i)$

[3] $SRTT$ = average of RTT samples similar to the variable used in TCP
[4] During overload the queue increases quadratically with time in rate-based AIMD.

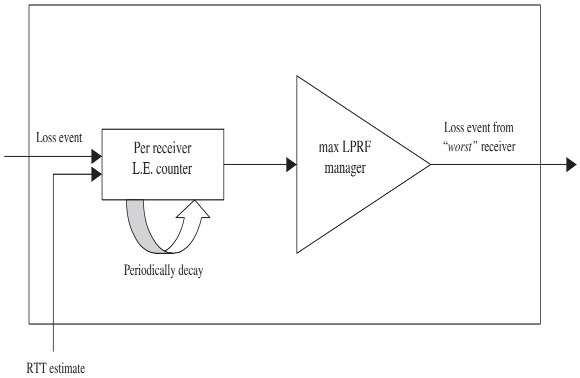


Figure 3: Max-LPRF: max-Linear Proportional Rate Filter

and on the 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 networks with no self-clocking.

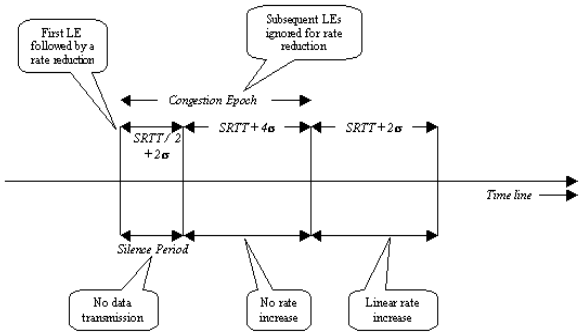


Figure 4: Adaptive Time Filter (ATF)

Adaptive Time Filter (ATF): This filter shown in Figure 4 is a modification of the time-filter introduced in our earlier work [12]. 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 an optional *"silence period"* of $\frac{1}{2}(SRTT+4\sigma)$ when no packets are sent. As suggested in [12], 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 [12]. 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 [12]) while other functions differ in their use of σ as described earlier.

In summary, the scheme expects: (a) at least one LI per receiver per RTT if packets are lost in that RTT and (b) timely generation of feedback by receivers/network elements to allow reliable RTT sampling[5]. The scheme is sensitive to arbitrary delays introduced by receivers/network-elements in generating an LI corresponding to a lost

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

packet. Further *LI aggregation* like in PGM [16] 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 the performance effects in the following sections.

3 Performance Evaluation

We have evaluated the performance of LE-SBCC to test for Drop-to-Zero avoidance, TCP-friendliness and LI aggregation effects. Note that these are the main factors affecting the performance of any multicast congestion control scheme. We use the following methods to test our scheme in very general scenarios :

1. *ns-2* simulations to generate representative test cases. These test cases have been chosen carefully to isolate key metrics for performance evaluation without loss of generality.

2. Simple Markov chain based modeling to obtain a better understanding for the choice of LE's as opposed to LI's for rate reduction.

3. Linux-based implementation/experimentation (high multiplexing degrees and upto 10s of receivers) to understand implementation issues and test performance on a real network.

4. Since *ns-2* simulations do not scale beyond tens of nodes, we wrote a 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. The LIs for a round are computed and informed to the source at the end of the round, and the control decision is enforced in the next round. (upto hundreds of receivers). We will refer to this method as *Numerical analysis* in the rest of the paper unless specified otherwise.

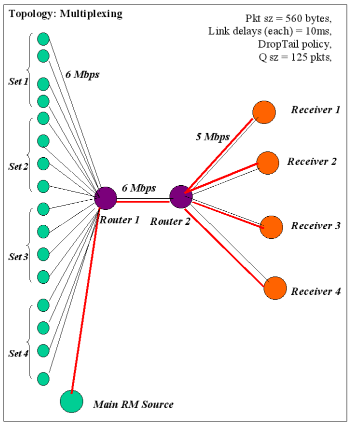


Figure 5: Topology to Test Drop-to-Zero Avoidance and TCP Friendliness

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. Heavy multiplexing, where many flows share a common bottleneck. In such a case, the loss rates observed at the source are independent of sending rate. A flow with a low rate will receive LIs with the same probability as other flows and thus goes through multiple rate reductions leading to a drop-to-zero problem.

2. Large receiver sets result in increased number of LIs being generated. It becomes important that the scheme does not react to more LIs than that generated by the max loss rate receiver.

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

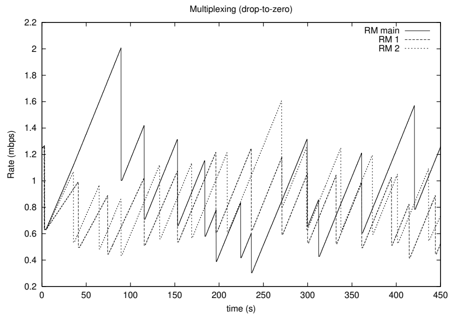


Figure 6: Drop-to-zero Avoidance Results

3. LI aggregation leads to loss 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 drop-to-zero problem using *ns-2*, numerical simulations and Markov Chain based theoretical analysis.

3.1.1 Theoretical Analysis of LE vs LI Probability

Recall that our scheme uses LEs instead of LIs for rate reduction decisions. Our scheme is critically dependent on the number of LE's passed by filters at the source. Thus it becomes crucial to study the relationship between LI's and LE's. We do a theoretical analysis to show that this choice has less probability of drop-to-zero as it results in fewer rate reductions. We also show that under bursty loss cases the number of LE's generated is significantly less than the number of LI's. This justifies the presence of LI to LE filter which plays an imortant role in our scheme. We propose a Markov Chain model for this process. RTTs are assumed constant for this analysis.

Our model assumes that at any point in time, the source may be in any discrete state from 1 to N. The state x specifies the number of packets sent during that RTT, $x = Rate(x) \times RTT$. We start in state 1, and may go upto the maximum allowable rate, which translates to N. In general, whenever a loss event is encountered, we do a transition to state $\lfloor x/2 \rfloor$. In case of no loss we move to state $x+1$. (See Figure 7) Let p be the LI probability. Let $a = 1 - p$. The probability of having at least one loss in state x (i.e probability of having an LE) is $1 - a^x$. For simplicity, assume $N = 2 \times m$. Let $s(x)$ be the probability of being in state x.We have five cases to consider :

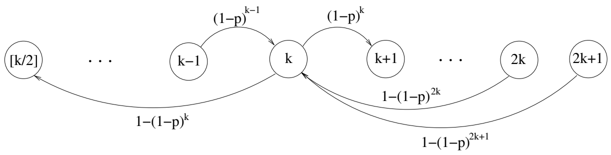


Figure 7: State transition of packets sent per RTT

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 stae is by cutting its rate, and moving to state m. In steady state,

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

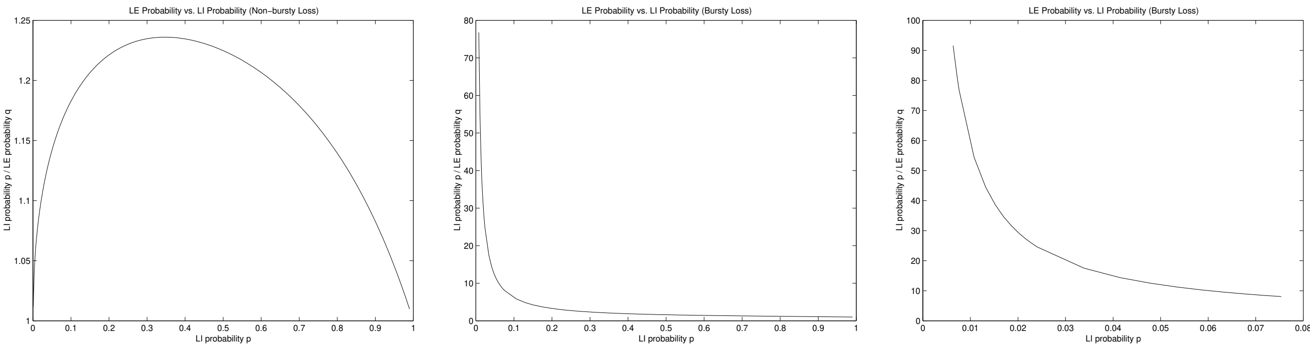
We did a Matlab analysis to solve the above set of equations for different large N's, final plots of which are similar. One plot of LI probability p and LE probability q ratio is shown below (Figure 8 (a)).

Note that we did not consider bursty loss in the situation above. But in the real world, packet losses are mostly bursty because of buffer queues and tail drop policy. To approximate the bursty behavior we modified our model as follows: we assume that if a packet is lost during any instant within the RTT, all remaining packets transmitted during that RTT are also lost. Under this assumption, p above is not the LI probability (packet loss rate) any more. Instead, it should be seen as the probability that a packet is lost and it is the first loss during a certain RTT. LI probability is calculated as follows:

During a RTT, the probability of no loss for the first i packets is $(1-p)^i p$. Recall that the state x specifies the number of packets transmitted during a certain RTT. At state x, the expected number of lost packet is $\sum_{i=0}^{x}(x-i)(1-p)^i p$. The LI probability is thus the expected number of lost packets over the expected number of total packets transmitted,

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

As a result, ratio of LI and LE probabilities is changed (Figure 8 (b)). In all cases we see that LE probability is lower than LI probability. This suggests fewer rate cuts and thus less likelihood of drop-to-zero, especially when the packet loss is bursty.



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

Figure 8: Ratio of LI and LE probabilities

For lower loss probabilities, the $LI(p)/LE(q)$ values are higher. To see the relation between LI and LE more clearly in this region, we have another graph for low LI probability (Figure 8 (c)), which fits well with our simulation results. We observe high p/q ratios (10-90). This indicates our choice of LEs as being more effective for drop-to-zero avoidance as fewer rate reductions would be performed.

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

We carefully choose a network topology that could focus on the drop-to-zero avoidance under *high multiplexing* scenario. Consider a main reliable multicast (RM) flow which goes to receivers 1 to 4 as shown in Figure 5. 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. In Figure 5, 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 to maintain a constant loss rate and avoid loss burstiness. The packet size is 560 bytes. Figure 6 shows that the multicast flow (solid line) competes fairly with a selected subset of unicast background flows (dotted lines) and the average rate is close to 1 Mbps, which is a non-trivial illustration of drop-to-zero avoidance.

The main RM flow transmitted a total of 167762 pkts in 450s and received 949 LIs (i.e. a low loss rate). After LI2LE filtering, there were only 32 LEs, a dramatic reduction! As discussed earlier, the bursty losses are caused by the lack of AIMD self-clocking in combination with drop-tail buffers. If one packet is lost, there is a high probability that the rest of the packets of the flow are dropped too due to the drop-tail policy. The Max-LPRF and ATF further filter LEs to result in a total of 12 rate-reductions. This is the same number of rate reductions experienced by a unicast flow as shown in Figure 6. This illustrates the important role of the LI2LE and Max-LPRF filters in the drop-to-zero avoidance process.

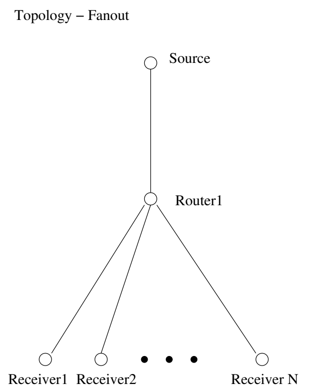
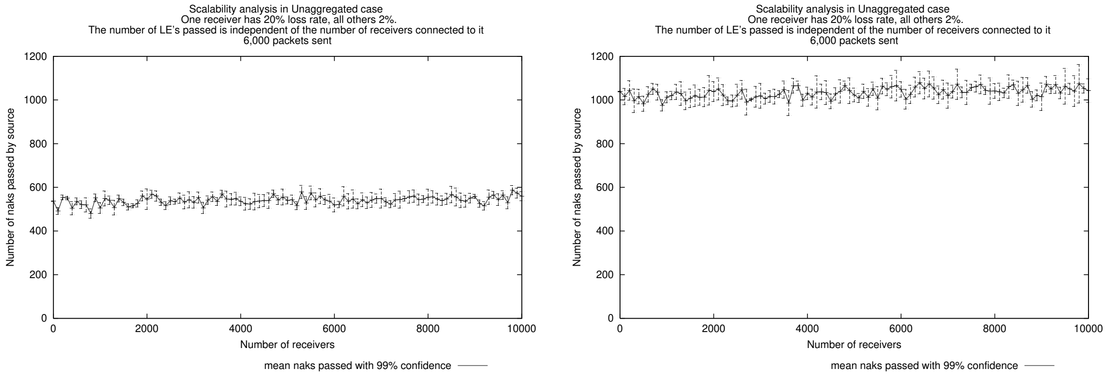


Figure 9: Topology : Fanout

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

Recall that large receiver sets result in increased number of LI’s being generated. To test the performance of our scheme in such scenarios we gradually increase the number of receivers in the multicast session, and rerun the simulation for each addition. One of the receivers in the session is clearly the worst loss one, and the rest have lower loss probabilities. We expect that the number of LE’s passed by source for rate reduction should remain fairly constant and equal to the number of LE’s 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. We use a fanout(Figure 9) topology where all the receivers and the single source are connected to the same router. 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 LE’s passed by the filters should be almost equal to the number of LE’s generated by the worst loss receiver. The number of LE’s passed for rate reduction by the source is plotted against the number of receivers(Figure 10(a)). The experiments were repeated for different random seeds and confidence intervals are plotted. Under no aggregation, the number of LE’s passed by source remains fairly constant(500-520). The number of LE’s passed are consistent with the non-bursty theorteical analysis performed above. The expected number of LI’s by the 10% reciver would be $0.1 \times 6000 = 600$. By refering to the LI probability(p) vs LE probability(q) graph(Figure 8 (a)) for the non bursty model, we see that for 10% loss p/q ratio is around 1.2. Therfore the expected number of LE's 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 LE's passed (Figure 10(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 LE's even when large number of receivers are present. This number is equal to the LE's 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 10: Numerical analysis: Scalability

3.2 TCP-friendliness

To illustrate TCP-friendliness, we generate topologies where a common bottleneck is shared by many RM and a single TCP flow and in the other case many TCP 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.

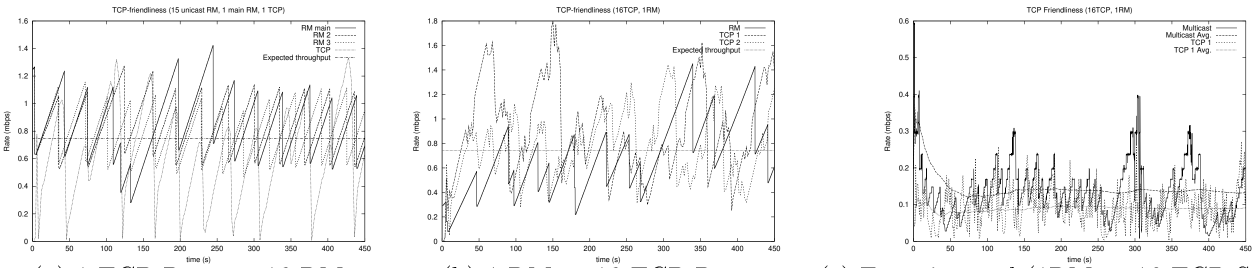
***ns* simulation results:** The simulation results for these two cases is shown in Figures 11(a) and 11(b) [7]. The dips by TCP close to zero represent timeouts. Again, the main RM flow (solid line in both graphs) shares the bottleneck(s) fairly with the competing TCP flow(s). The horizontal dotted line is the expected TCP throughput plotted post-simulation using the simplified TCP equation ($1.22MSS/RTT\sqrt{q}$).

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 11(c)). More details on this are presented in the section on implementation issues.

3.2.1 Expected RM Throughput – A Measure of TCP-friendliness

TCP-friendliness is a subject of current debate [6, 15, 9]. A good measure of TCP-friendliess would be the average throughputs of the competing TCP and RM flows. For that purpose, 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 11(a) and 11(b), the rate values oscillate about their mean shown by horizontal dotted line. Their mean

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



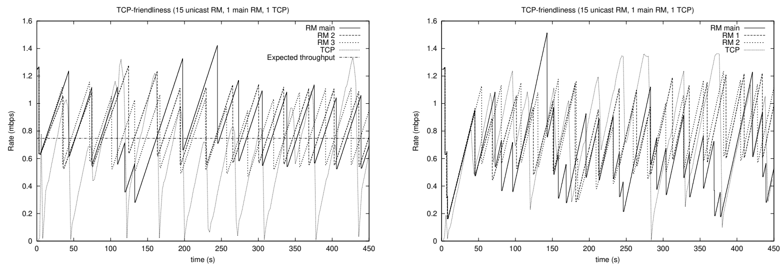
(a) 1 TCP Reno vs 16 RM (b) 1 RM vs 16 TCP Reno (c) Experimental (1RM vs 16 TCP SACK)

Figure 11: TCP Friendliness (Simulation and Experiment)

rate is almost equal 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.

3.2.2 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 toplogy 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*.



(a) DropTail (b) RED

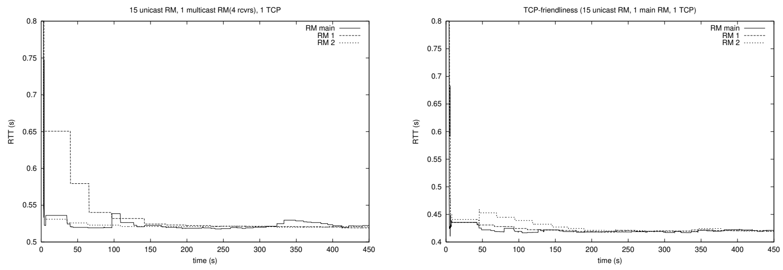
Figure 12: Droptail Rate vs. RED Rate

As we can see from Figure 12, the throughput is marginally lower with RED than with DropTail and so is the measured RTT (Figure 13). Also observe that the RTT converges much faster with RED than DropTail. The scheme remains fair towards the competing TCP flow in both cases and we do not observe any drop-to-zero with RM flows.

3.2.3 Different Loss Paths

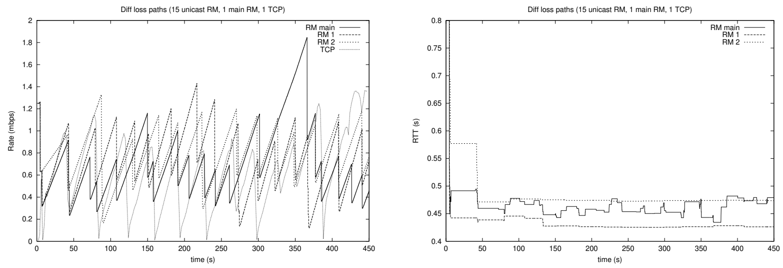
It was speculated that the scheme would not perform as well with the multicast tree having paths with varying degree of loss probabilities and RTTs. We perfomed the following simulation to test the robustness of our scheme under these hetrogeneous conditions. Again the toplogy remains the same as the above experiment with modifications in follwing 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.

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



(a) DropTail (b) RED

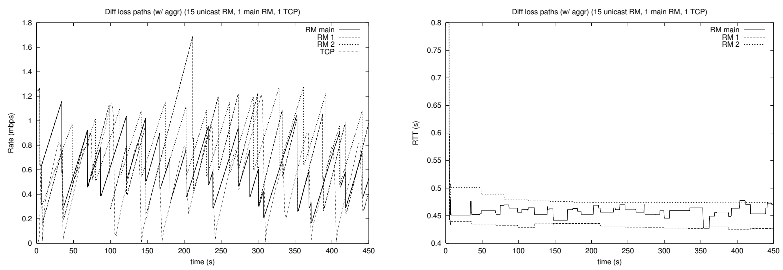
Figure 13: Droptail vs. RED RTT convergence



(a) Rate (b) RTTs

Figure 14: Rate and RTT graphs for topology with different buffers (loss rate and RTTs)

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 recieved from the 4 receivers). The simulation is performed for both non-aggregation (Figure 14) and aggregation cases (Figure 15). The graphs clearly demonstrate that the algorithm works well with aggregation enabled at the router too.



(a) Rate (b) RTTs

Figure 15: (With aggr) Rate and RTT graphs for topology with different buffers (loss rate and RTTs)

3.3 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 16. Observe that the number of receivers and their loss rates can be varied. Consider three cases: (a) *no LI aggregation* performed at the

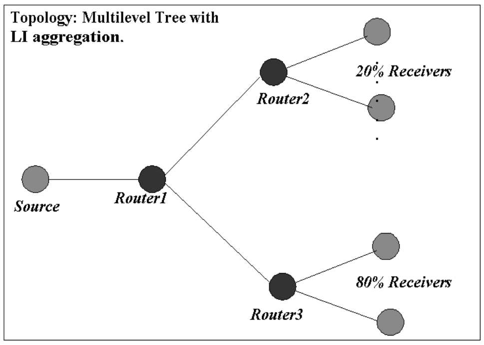


Figure 16: Topology to Illustrate LI Aggregation Effects

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 16, 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).

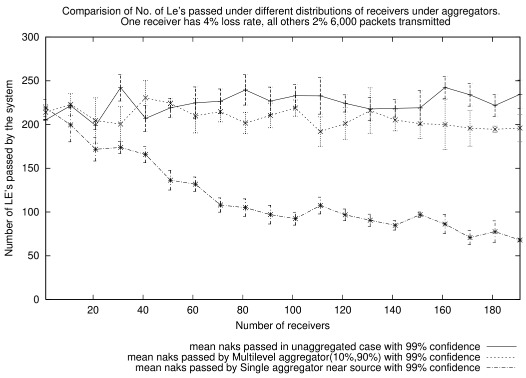


Figure 17: LI Aggregation Effects: LEs Passed by Cascade of Filters

The numerical simulation for cases (a), (b), (c) above are shown in Figure 17. The figure illustrates the LEs passed by the cascade of filters (one curve for each case), plotted against the number of receivers (upto 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 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 LE's generated by the worst loss receivers could be supressed. Since our scheme is very sensitive to LE's, such supression could lead to a degradation of performance. We analyze the worst case for aggregation numerically, in terms of LEs reaching the source. The topology used is a fanout(Figure 9) one, where the source and all receivers are connected to the aggregator. Intitutively, this topology provides for total aggregation, as only one LI among the posible N will reach the source for a particular sequence number. As an 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 LE's reaching through the source(after filtering) should be equal to the number of LE's 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 reciver. 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. Lesser and lesser LI's generated by X might reach the source. In the worst case this would reach zero. The source might become more aggresive, as it will get lesser number of LIs after filtering. We use the topology mentioned above for this analysis. We assume one receiver has 20% loss rate, and all others 2%. We study the effect of increasing the number of 2% recivers under the aggregator. These numbers were chosen because the 20% loss rate receiver is clearly the worst loss receiver. The number of LE's passed for rate reduction should be approximately equal to the number of LE's generated by the 20% receiver. Any significant change in the number of LE's passed due to the inrease of 2% receivers would clearly affect performance. The number of LEs reaching the source vs the number of receivers is plotted(Figure 18). Under "biased" aggregation, the number of LEs passed by source drops sharply by increasing the number of 2% recevers. Finally, the source only reacts to 2% loss, instead of 20%. Our scheme becomes overly aggresive in the worst case scenario (Figure 18).

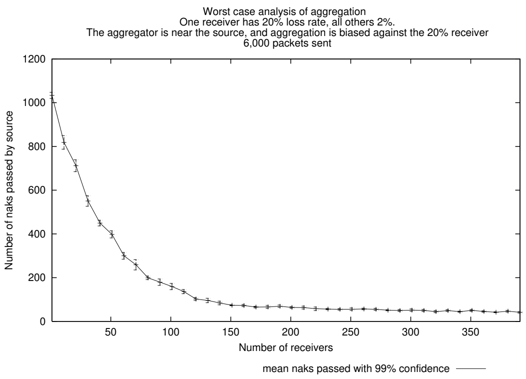
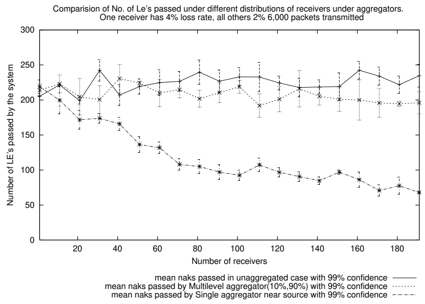


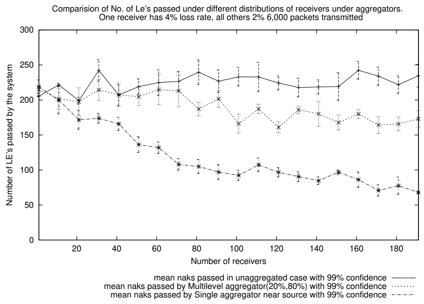
Figure 18: Numerical analysis: Worst case aggregation

3.3.2 Partial Aggregation Analysis

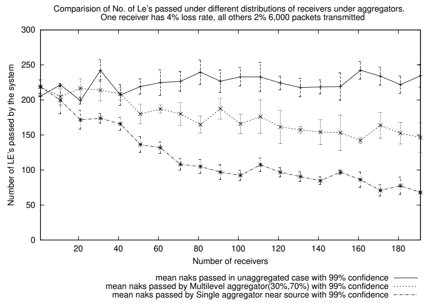
In the previous section it was shown that in the worst case of aggregation, the scheme becomes overly aggresive. However, in real networks many other indermediate cases are possible, which would not always affect performance significantly. We numerically analyze some of these cases. In the following graphs (Figures 19, 20 and 21) we show a complete range of results in case of partial aggregation. The topology is same as in Figure 16. 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 4% receiever) are on Router2 and remaining y% are under Router3. The experiments were repeated for different random seeds and confidence intervals are 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 19) the max loss receiver is present in the branch which has lesser number of receivers. Under such cases, the number of LE's passed(210-225) is almost equal to the unaggregated scenario. However in (Figures 20 and 21) the max loss reciver is present in the branch with higher number of receivers. Under such cases our scheme becomes more aggresive as fewer LE's are passed. The number of LEs passed never drops below the case where a single aggregator(Router1) is present near the source.



(a) 10-90

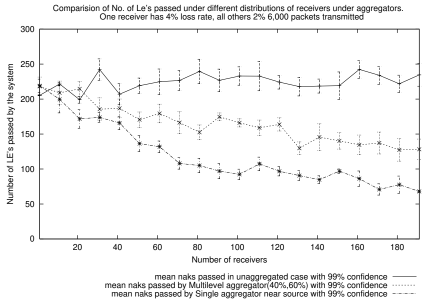


(b) 20-80

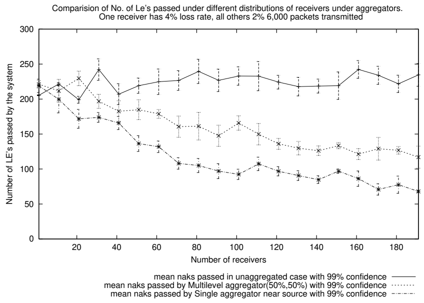


(c) 30-70

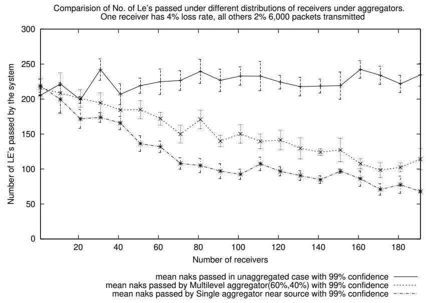
Figure 19: Partial Aggregation Effects



(a) 40-60

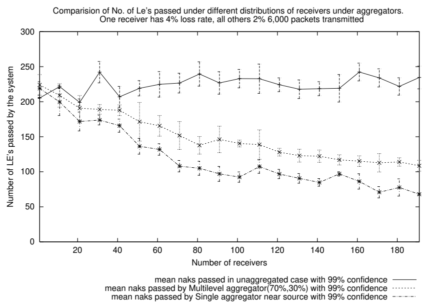


(b) 50-50

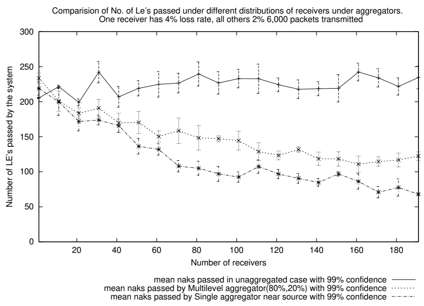


(c) 60-40

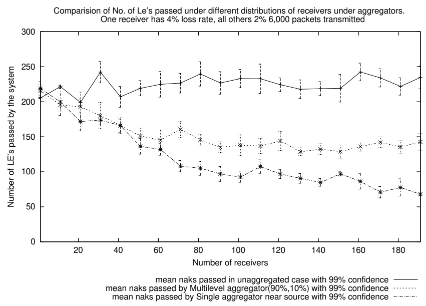
Figure 20: Partial Aggregation Effects (contd.)



(a) 70-30



(b) 80-20



(c) 90-10

Figure 21: Partial Aggregation Effects (contd.)

4 Linux Implementation and Experiments

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 remaining sections.

4.1 Implemenation

The scheme is based on a simplified version of PGM. It has been implemented on top of UDP using RedHat Linux 7.0. The following aspects of PGM have been realized:

- 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 suppposed 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 accumulative. 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 μs 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 triggered. This introduces a minor random factor ($< 5ms$) into event scheduling, but reduces the unnecessary delays mentioned above.
- We introduces 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 observered that the mechanisms described above worked as expected.

4.2 Experiments

The experiments were conducted on our testbed consisting of RedHat Linux 7.0 i586/i686 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* [8] was used to route multicast packets from one LAN to others.

The LE-SBCC program runs in user mode subject to 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.

4.2.1 Basic Experiments

Some basic experiments, like one source to two receivers transmission, have been done to ensure correctness of the rate control module. The results agree with *ns* simulation. (Graphs are omitted).

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

4.2.2 TCP Friendliness Experiment

During the TCP friendliness expriment, TCP SACK was enabled. We used TBF queue discipline of *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 unifrom distribution. This backoff increases the RTT. The maximum random backoff time is a configurable parameter. 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.

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 22, 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.

5 Summary and Conclusions

We have presented an *LE-oriented, purely source-based multicast congestion control scheme (LE-SBCC)* for reliable and unreliable multicast transport . The model is a simple cascade of filters followed by an AIMD module. In general the AIMD module may also be replaced by a source-based TFRC scheme [6] or rate-based binomial scheme [1]. With partial or no LI aggregation, the scheme correctly filters the 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. The scheme is simple to implement and deploy since it is completely contained at the source[9].

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

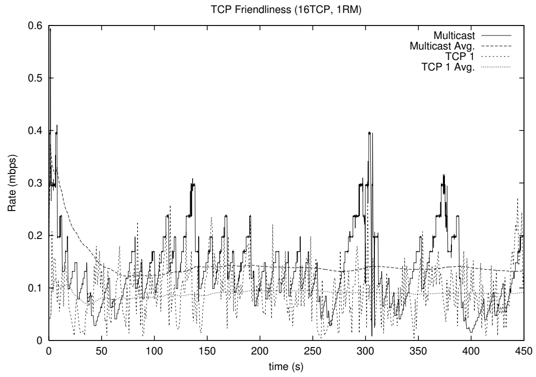


Figure 22: TCP friendliness expriment result (Linux implementation)

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

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

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

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

[7] Golestani J., "Fundamental observations on Multicast Congestion Control in the Internet," *INFOCOM 1999*,

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

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

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

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

[12] 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).

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

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

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

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

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

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

[19] 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 + 2σ
    CongestionFlag = TRUE
    CongestionEpochTimer = SilencePeriod + SRTT + 4σ
    Accept LE
  }
}

event: CongestionEpochTimer expires:
        CongestionFlag = FALSE

event: SilencePeriodTimer expires:
        SilenceFlag = FALSE

event: RateIncreaseTimer expires:
        if (CongestionFlag == FALSE)
          rate+ = MSS/(SRTT+2σ)
        else {
          if (SilenceFlag == TRUE)
            no data transfer
          else
            no increase in rate
        }
        RateIncreaseTimer = RTT + 2σ
```