SBMCC: A Generic Source-based Congestion Control Algorithm for Reliable Multicast [1]

Neelkanth Natu [2], Priya Rajagopal, Shivkumar Kalyanaraman
Department of ECSE, Rensselaer Polytechnic Institute
Troy, NY 12180-3590
Contact Email: {*natun, rajagp*}@networks.ecse.rpi.edu, *shivkuma*@ecse.rpi.edu

Abstract

This paper presents a simple, generic source-based end-to-end multicast congestion control (SBMCC) algorithm for reliable multicast transport (RMT) protocols. The algorithm is completely implemented at the source and leverages the reverse control information flow in RMT protocols like PGM or RMTP [13, 43]. Specifically, it does not introduce any new control traffic or new fields in RMT protocol headers. It addresses the drop-to-zero problem [42] by introducing a robust, adaptive time-filter based upon RTT estimates collected by observing NAK traffic. This solution allows it to scale for large multicast groups while being very adaptive to congestion situation changes in any part of the tree. The algorithm is provably friendly to TCP, and competes fairly with TCP for bandwidth. The scheme has minimal control traffic requirements and weak RTT estimation requirements which allows a large deployment space including multi-sender multicast.

1 Introduction

Multicast is slowly moving from experimental deployment on the MBONE to a service on the Internet [12, 2]. Several standards activities are underway in the IETF to support this trend. Activity in flow and congestion control [7] is happening in the IRTF reliable multicast research group (RMRG) [24] and within a subgroup of the Reliable Multicast Transport (RMT) working group [23]. Several congestion control schemes have been proposed in the the literature [5, 18, 35, 10, 26] which contribute substantially to the understanding of the problem and provide sample solutions. Since congestion control is an important standardization issue [7], an open question is whether generic schemes exist which cover the gamut of implementation requirements of RMT protocols.

In this paper, we argue the need for *two* generic congestion algorithms in the standards process: a generic source-based and a generic receiver-based algorithm (section 1.2). We propose a generic, source-based end-to-end algorithm (SBMCC) which leverages the underlying RMT control traffic to provide stable, scalable and TCP-friendly control. The scheme includes novel algorithms to address key issues such as RTT estimation, the drop-to-zero problem [42] and TCP-friendliness. The *distinguishing feature of our scheme is its high-level of deployability* which arises out of very low (near-zero) control traffic demands from RMT protocols, high level of adaptivity to dynamic congestion situation changes in the tree, weak requirements on the process of RTT estimation, and the need for only source-based support. It can therefore be mapped onto a large set of underlying RMT protocols including multi-sender RMT protocols.

[1] This work was supported in part by Reuters and by a Rensselaer Outstanding Young Faculty Award
[2] Neelkanth Natu is now with Entera Corp, CA

1.1 Reliable Multicast Transport Protocols

The general model of reliable multicast transport (RMT) protocols is to provide a combination of *"temporal redundancy"* and *"spatial redundancy"*. Temporal redundancy (a.k.a. "soft state") refers to providing redundancy over time by sending retransmission or repair traffic based upon status reports about the original traffic from receivers. Spatial redundancy refers to providing redundancy within the original transmission, typically in the form of forward error correction (FEC). Excellent surveys of RMT protocols may be found in [20, 27, 33]. In general, the two key problems which arise in RMT (and proposed solution approaches) are:

1. The *implosion* of control traffic on the reverse direction from multiple receivers to the sender (M-to-1). The proposed approaches to this problem include:
   - *Scoped multicasting of reverse control traffic combined with probabilistic suppression* techniques at the receivers [16]
   - Use of *negative acknowledgements (NAKs) or bitmaps* instead of simple acknowledgements (ACKs) [13, 38, 16, 43].
   - *Explicit aggregation* using aggregators or generic router-assist, and *reliable reverse NAK channels* [31, 43, 13, 9].
   - *Statistical or round-robin (scheduled)* selection of receivers which would send control traffic [19, 21].
   - *Scaling the feedback frequency based upon group size* [39].
2. *Optimization of the repair traffic* and the bandwidth resources consumed by such traffic in various parts of the tree. The proposed approaches to this problem include:
   - Use of *local retransmittors* (e.g., Designated Local Repairers (DLRs) in PGM and Designated Receivers (DRs) in RMTP [13, 31, 43]), or allowing receivers themselves to transmit repairs [16]. Such retransmission must necessarily be scoped to the subtree (with a combination of TTL and administrative scoping)
   - *Subcasting*: *Router-assist* to form a separate subtree per-packet to be retransmitted, and constraining retransmission to flow on this subtree [13, 9].
   - *Reactive FEC:* Sending FEC information instead of the packet itself *during retransmission/repair*. Rizzo [36] has shown that general erasure codes exist that allow the generation of n packets from k original data packets. Thus the sender can construct FEC knowing what *fraction* of the n packets reached the destination.
   - *Proactive FEC:* The use of carefully designed FEC is attractive for large, heterogeneous trees [32, 37] to reduce the cost of reverse and repair control traffic.

The mechanisms above deal only with the issue of reliability and not congestion control. Our purpose of presenting RMT mechanisms here is because congestion control schemes will have to build upon this infrastructure of available mechanisms. Reliable multicast congestion control (RMCC) in its own right has issues which differ significantly from unicast congestion control issues. These issues are the topic of the following section.

1.2 Reliable Multicast Congestion Control Issues

Reliable multicast congestion control (RMCC) issues can be understood by studying TCP congestion control and examining the effects of moving from a point-to-point model ("path") to a point-to-multipoint model ("tree") [17]. In comparison, TCP uses an adaptive window and sizes its window according to the bandwidth-delay product of the path [25, 1].

We assume a model where the sends data at a single rate which is adjusted based upon congestion information. Alternative models include layering (and grouping) application data, and giving the responsibility for receivers to join or leave groups based upon congestion estimation [41, 40, 29, 6]. We note, that our scheme could also be applied *within* each of the groups (layers) mentioned above or in conjunction with other group management strategies [17, 4]. The key RMCC problems in our model are discussed in the following subsections.

1.2.1 Window-based vs Rate-based Schemes

The first key problem in RMCC, noted by Golestani [17] is that source window-based TCP-like multicast congestion control schemes can run into the danger of allocating rates even below the fair share on the worst bottleneck. This is because the window is regenerated and paced based upon acknowledgements, which can be robustly collected only by waiting for RTT_{max} (the maximum round trip time on the tree), and the window must be set to the minimum bandwidth delay product (W_{min}). The resultant rate W_{min}/RTT_{max} is less than the fair share on the worst bottleneck. Rate-based schemes could avoid this pitfall. Hence most multicast congestion control work has focused on rate-based schemes, though some proposals combine rate-based and window-based control [10, 13].

Golestani proposes a receiver-driven window-based scheme which is equivalent to maintaining per-receiver state (multiple windows) at the sender. The scheme requires aggregation support for conveying the explicit congestion feedback to deal with the feedback implosion issue. Other rate-based schemes which require aggregation support for congestion control include MTCP [35], TFMCC (the RMRG baseline) [18], the representative-based scheme [11] and nominee-based scheme [26] (which use representatives or nominees as aggregators). In our scheme, we use rate-based control which is completely implemented at the source. We leverage underlying RMT NAK traffic and aggregation capabilities and do not insert new control traffic or require any support in terms of aggregation.

1.2.2 Parameter Estimation Ambiguity

The second key problem involves *parameter estimation ambiguity.* A path has a unique RTT (delay), and a bottleneck bandwidth. But a tree is likely to have many branches, each with its own RTT, loss rate and bottleneck bandwidth which leads to ambiguity in constructing a single bandwidth-delay product or defining what is the RTT of the tree.

One manifestion of such parameter estimation ambiguity is a problem called the *drop-to-zero problem* [42]. The drop-to-zero problem occurs when the sender reduces its rate in response to congestion indications from different parts of the tree to a level far below what is necessary to correct the congestion problem (i.e. the load-capacity imbalance). An example of a scheme with the drop-to-zero problem is the sender-based scheme proposed by Montgomery [30] which uses a time-based filter at the source, but does not include a mechanism to estimate the filter value correctly.

Bhattacharya et al showed that the drop-to-zero problem was a filtering problem and proposed source-based filter designs to counter it [3]. They argue that the window setting should be based upon the tracking of the *worst loss rate* on any of the paths [5, 26]. This work, however, does not examine the issue of measurement of RTTs at the source. If RTTs are measured at the receivers, there is need for new control traffic and aggregation of such traffic. Also since aggregation of regular loss indications loses information about receiver IDs (used in their filters), some fidelity in the filtering process may be lost.

Aggregation in general leads to interesting estimation-related issues. If aggregation is not perfect, multiple congestion indications (NAKs) would be received at the sender. If aggregation is present, the ID of the receiver is lost in the aggregation process and cannot be used in any parameter estimation or filtering algorithm. Moreover, control traffic which comes in late at the aggregator from long RTT branches is suppressed as part of the aggregation process. In such a case, the implicit information carried in control (eg: timing information useful for RTT estimation) will not reach the source. In other words, the requirements of finding the longest RTT at the source are at conflict with the assumptions of aggregation. Our solution to this issue is to be less sensitive to RTT estimates, while at the same time being adaptive to dynamic congestion situation changes in the tree.

1.2.3 TCP-friendly Behavior

The third key problem involves fairness to TCP, or *TCP-friendly transmission*. Since TCP traffic represents over 90% of Internet traffic today and a possibly dominant portion of future traffic, multicast transport protocols must be fair to TCP and not starve it of bandwidth. Golestani [17] discusses two types of fairness: rate-fairness and window-fairness and suggests that the goal of TCP-friendliness is to approximate window-fairness.

An alternate, stronger way of formulating the problem is for multicast transports to approximate the steady state TCP throughput as defined by the Padhye formula [34]. This formula depends upon loss probability (p), the RTT and the timeout period. With multiple branches, the goal is to follow the worst path in terms of the throughput achievable by an equivalent TCP connection on that path [42, 18]. Bhattacharya et al propose a slightly different formulation where the worst path is defined in terms of the observed loss rate alone, and not including the effect of RTT [5]. In either case, achieving *exact* fairness with respect to this TCP formula is a problem because the TCP timeout factor cannot be precisely estimated. The timeout length is roughly proportional to RTT rounded up to the nearest clock tick (depends upon timer granularity which ranges from 50ms-500ms). At the same time, ignoring the timeout factor is a debatable issue because over 50% of all TCP retransmissions today involve a timeout [34].

Our approach to TCP-friendliness is to construct a source-based scheme with TCP-equivalent response and filtering mechanisms (or approximations) which leads to the mimicking of TCP behavior.

1.2.4 Sender-based vs Receiver-based Schemes

The fourth key issue is that, unlike TCP, a one-size-fits-all transport solution is not available or possible for reliable multicast. The IETF RMT working group is looking to standardize a small number of transport protocols [20]. Though the requirements for best-effort congestion control (CC) is same in general for all these protocols, the mechanisms and underlying control traffic available to implement a CC protocol vary widely across the different RMT alternatives.

The receiver-based CC approach is immediately applicable (and may be the only alternative) in cases where RMT feedback traffic does not exist (eg: Digital Fountain [8]), where RMT feedback is sparse and not reliable as congestion feedback [19, 21], or where reverse RMT feedback terminates at intermediaries such as designated retransmittors [43] and may not reach the sender. Examples of the receiver-based approach include TFMCC [18, 42] and the nominee-based scheme [26].

For all other protocols which have NAK [13, 16], ACK, Hierarchical ACK (HACK or TRACK) or bitmap [31, 43, 38] traffic which can be leveraged, a source-based strategy would lead to reduced congestion control feedback traffic. The disadvantages of the source-based approach is the need for some reverse control traffic which they can reasonably leverage. If the underlying RMT schemes use FEC-based techniques for reliability, such reverse traffic may be sparse or provide unreliable congestion and timing information [19, 21].

The disadvantages of the receiver-based approach include the fact that point of rate computation (receivers) is different from the point of enforcement (sender). This results in the need for new congestion feedback traffic and associated aggregation or statistical selection/suppression mechanisms. The approach also needs a robust method to measure RTT at the receiver. The use of GPS-based synchronous timers has been proposed for this purpose in the TFMCC scheme [18]. From an complexity standpoint, support for congestion control is required in all receivers whereas source-based schemes need support only at the source. From a state requirements standpoint, since the receivers maintain (S,G) congestion state they might have scalability issues when extending to the case of many senders (though admittedly such applications are not in existence today).

We argue that given such conditions, the standardization process should consider two *generic* schemes which cover the gamut of implementation requirements: one generic source-based scheme and one generic receiver-based scheme. The RMT implementors can then choose one depending upon available underlying mechanisms. In this paper, we present a generic source-based scheme.

2 The Source-Based Congestion Control Scheme

For our baseline discussion, we assume a source-based congestion control scheme targeted at a NAK-driven RMT model like PGM [13]. Though the scheme as outlined is NAK-driven and our results presented here use a PGM model, the scheme can be adapted to other types of feedback including ACKs and HACKs. The scheme features are covered in the following sections.

2.1 Increase/Decrease Policy

The basic scheme is rate-based and uses the traditional additive increase/multiplicative decrease (AIMD) policy.

2.1.1 Rate Increase Policy

In order to emulate TCP behavior, the rate increases in increments of MaxPacketSize/RTT. This increment is inversely proportional to the RTT (round trip time) estimate. In our case, the RTT estimate comes from the congested sub-tree and not the entire tree. The congested sub-tree (see Figure 1) includes all paths from source to receivers which have at least one bottleneck, i.e. points where the demand outstrips capacity.

Rate increases are performed in the absence of new NAKs, i.e. when a timer expires and no "new" NAKs (see section 2.3) are received during that period. An interesting question is how long to set this "rate-increase trigger" timer. In the congestion avoidance phase (steady state) TCP increases its window by a constant (one MSS) approximately once per RTT. In the absence of a TCP-like ack-clock [25], we set this timer to a smoothed mean RTT estimate plus *twice* the smoothed mean deviation of samples collected from the congested sub-tree (see section 2.3.3). We choose the congested sub-tree RTT as the basis because it is that portion of the tree which needs to respond to the rate-increase i.e. signal if the rate increase has resulted in congestion.

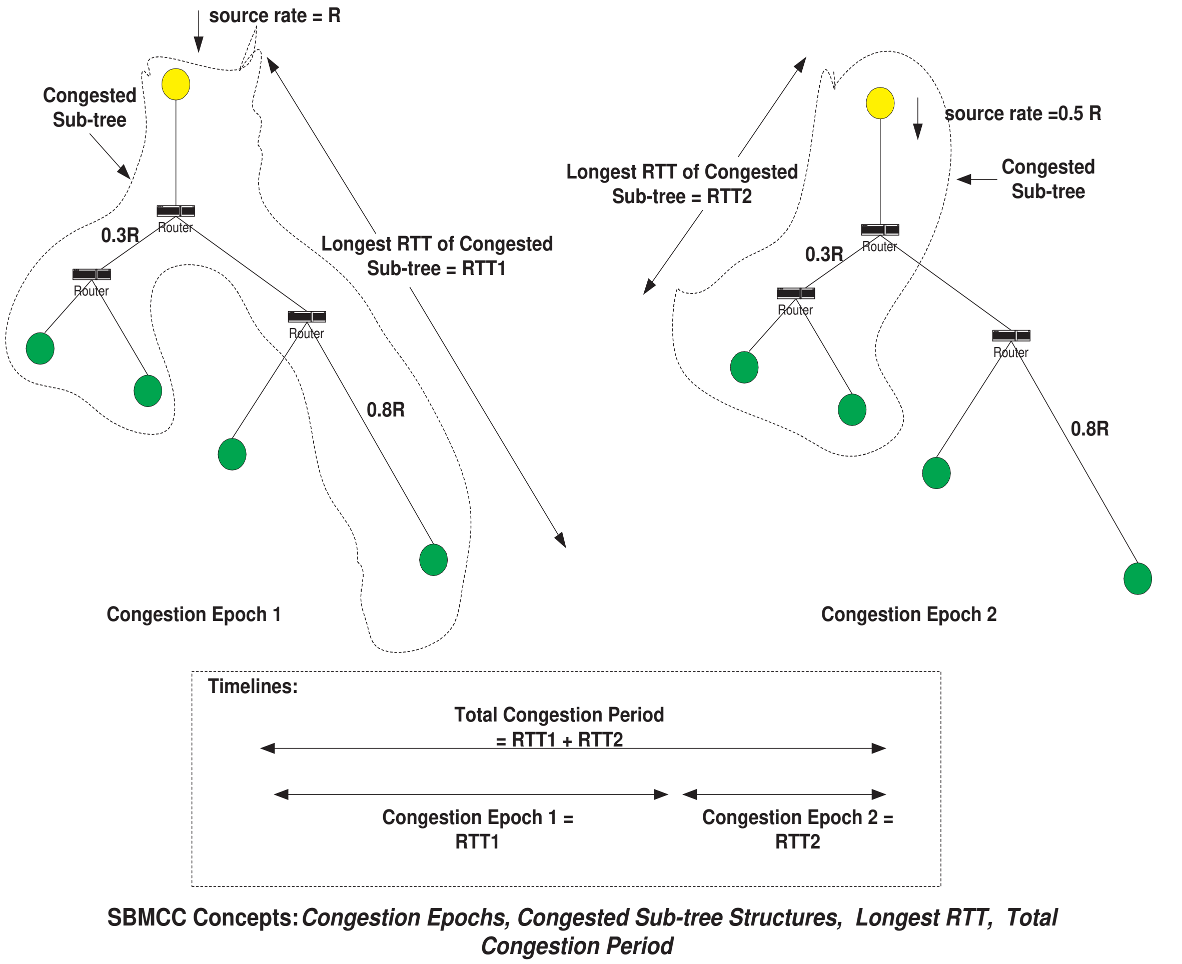


Figure 1: Illustration of SBMCC Scheme Concepts

The rate-increase interval chosen as discussed above entails a small risk of increasing the rate a little more than necessary when the structure of the congested sub-trees changes between periods of congestion. But this risk is not significant unless the RTTs of the newly congested part is several orders of magnitude larger than the maximum RTT of the currently known congested sub-tree. We discuss the estimation procedure in section 2.3.

2.1.2 Rate Decrease Policy

Rate decreases are made based upon an operational concept of a *"congestion epoch."* A congestion epoch is a sub-period of congestion (load exceeds capacity) which starts upon the first detection of congestion since the last epoch (or the beginning of transmission), and ends after allowing time for the following: **a)** time for the effect of any load change (rate-decrease) to propagate to the receivers (in the congested sub-tree) and, **b)** time for acknowledgements (possibly implicit) sent after the receivers experience the effect of load-change to reach the source. Note that the congestion epoch is a *sub-period* of congestion, i.e. it does not capture the entire duration of congestion but includes the occurrence of congestion, time for a subsequent response to congestion and time for the response to take effect in the congested sub-tree. The relationship between congestion epochs and the total period of congestion is illustrated in Figure 1.

The duration of the congestion epoch is *ideally* the maximum RTT of the congested sub-tree measured from the time-instant a rate-decrease is enforced. Therefore, we set the length of a congestion epoch using the TCP timeout estimation algorithm with some additional modifications [25, 1] The similarity between the congestion epoch and the TCP timeout is that both must *ideally* be larger than maximum expected RTT. The difference is that in the case of congestion epochs, we could afford to have an estimate smaller than the maximum RTT as long as we do not incur more congestion epochs than necessary (see section 2.1.3). In other words, the congestion epoch can afford to be less sensitive to the RTT estimation process. A TCP timeout however cannot have a high probability of being smaller than the RTT because a smaller timeout value would lead to spurious timeouts, retransmissions and window reductions.

The concept of congestion epoch is important from a rate-decrease perspective because *a new rate decrease is only performed at the beginning of every congestion epoch.* In our NAK-based model, the congestion epoch begins when a new NAK is received. The rate is immediately reduced by half (like TCP). Further, we enforce a couple of policies:

- First, the source remains silent for half an RTT immediately after rate-reduction, i.e., it does not send any packets during this phase (even retransmissions). This is done to allow the bottlenecks to clear, and to allow any control traffic which is not rate-controlled (like NCFs in PGM [13]) to be sent. The silence period is also part of the filtering period, i.e., NAKs received during this phase are ignored for rate-reduction purposes.

- Second, the rate is not changed for the duration of the congestion epoch. The hold-down of the rate to half the pre-congestion rate during the congestion epoch yields performance which is roughly equivalent to the effective rate achieved during the fast retransmit and recovery phase in TCP.

2.1.3 Congestion Epochs and the Drop-to-Zero Problem

The congestion epoch concept is important in addressing the "drop-to-zero" problem because the number of congestion epochs detected during congestion is equal to total number of rate reductions. The first "new" NAK (see section 2.3) received after the end of a congestion epoch is an indication that the source rate is still larger than the minimum bottleneck rate of the tree. It therefore triggers a new congestion epoch and corresponding rate-reduction.

Given this model, the drop-to-zero problem can be addressed if the the number of congestion epochs detected is *exactly equal to* $\lceil \log_2 \frac{\lambda}{\mu_{min}} \rceil$. Here λ is the source rate at the beginning of the set of congestion epochs and μ_{min} is the minimum bottleneck rate anywhere in the tree. The logarithm is set to the base 2 because in each epoch, the source rate is halved upon every rate-decrease. In other words, our goal is to have exactly the number of rate reductions (or congestion epochs) necessary to correct a load-capacity imbalance in any part of the tree during congestion. The cause of the load-capacity imbalance is immaterial. For example, capacities on links anywhere in the tree could suddenly vary, or there could be an increase in the background traffic (TCP or non-TCP flows). In Figure 1, the maximum imbalance occurs at the beginning of congestion epoch 1. The source rate is R, and the minimum bottleneck rate is 0.3R. According to our formula above, we should have no more than two congestion epochs (leading to a final source rate of 0.25R). Else we would be incurring excess reductions, i.e., suffering the drop-to-zero problem. As shown in the figure, our scheme would have two congestion epochs leading to a total congestion duration of RTT1 + RTT2.

Also observe that the condition above is only on the *number* of congestion epochs, and is *not* dependent upon the *length* of the epochs. Hence, we could tolerate some errors in the estimation of RTTs and the congestion epoch *lengths so long as it does not affect the number of epochs required to address a given congestion situation.* We will make use of this property to deal with the issue of timing information lost in certain cases such as NAK aggregation.

2.2 Filtering Issues

One implication of choosing not to reduce the rate multiple times during a congestion epoch is that NAKs received during the congestion epoch are "filtered" out for the purposes of rate-reduction. NAKs may be used for other purposes like retransmission and RTT estimation. The congestion epoch may also be considered as a filtering period in this context. The filtering of NAKs during the congestion epoch is similar to enhancements made in TCP NewReno and TCP SACK to avoid multiple window reductions for packet losses detected within a single window [15, 28]. The difference is that our filtering interval is a time interval, whereas the filtering interval in TCP is based upon a sequence number. In multicast, the filtering process addresses two problems: *spatial filtering* and *temporal filtering*.

Spatial filtering: This is the problem of choosing parameters only from the *congested sub-tree* of the multicast tree. Note that we do not have to pinpoint the structure of the congested sub-tree. We only need to *measure parameters* from that part of the tree. By definition, NAKs are generated by receivers which belong to the congested sub-tree. The identification of the congested sub-tree is difficult in a pure HACK- or ACK-based system (see section 2.3.5). But as we shall note in sections 2.1.3 and 2.3.5, the congestion epoch setting can be less sensitive to errors in RTT estimation which means that the congested sub-tree concept is important only in an optimization sense (and not for stability).

Temporal filtering: This is the problem of choosing the right length of the filtering interval to meet two constraints: **a)** not to allow congestion to persist longer than necessary, and **b)** to filter out multiple congestion indications which can be resolved by a single rate-reduction. We have discussed the latter constraint earlier in our description of the congestion epoch and its use in addressing the drop-to-zero problem. The former constraint is again an optimization constraint, not a stability constraint. Specifically, if the filtering interval is too large, severe congestion may persist for longer periods than necessary. We will encounter this problem in a pure HACK- or ACK-based system because we cannot identify the congested subtree RTT (see section 2.3.5). In our solution, if multiple rate-reductions are required to solve the congestion problem in any part of the tree, they are ultimately made. But we are constrained to effect these rate-reductions over multiple congestion epochs because we do not have information on the *extent* of overload during congestion and hence make only one rate-reduction per congestion epoch.

2.3 RTT Estimation and NAK issues

Our RTT estimation procedure works by collecting RTT samples, pruning the set of RTT samples, and finally calculating the smoothed average and mean-deviation of the remaining samples.

2.3.1 RTT sampling

Each RTT sample is collected as follows. When a packet is sent, a timestamp is locally recorded for that sequence number. When a NAK is received for that sequence number, the difference between the current time and the recorded timestamp gives an RTT sample. Observe that this RTT estimation technique is different from TCP because the latter times only selected segments and uses a single variable (in units of ticks) for the RTT measurement. Our method is geared for *"opportunistic"* RTT sampling required in the NAK-based model. Note that the NAK-based RTT samples are a better proxy for the maximum end-to-end delay than acks because the former samples RTT during congestion. At the same time, NAK-based samples may be "late" samples especially if the congested sub-tree structure changes to include a new, very long RTT path. This could lead to transient behavior involving possible extra rate-reductions because the congestion epoch was sized too small earlier. We note that the above problem is only transient and is usually corrected within a single congestion epoch.

Since multiple NAKs can be received for the same sequence number, and our RTT estimation technique could be inaccurate during transients, we have an additional mechanism to prevent spurious rate-reductions. Specifically, when a NAK is received, a bit is set if this the *first NAK seen for that sequence number*. Such a NAK is also called a *"new" NAK*. If the NAK is the first "new" NAK *after* the end of the last congestion epoch, then it signals the beginning of a new congestion epoch and a rate-reduction is effected. Rate reduction is not effected upon the receipt of subsequent new NAKs within a congestion epoch or old NAKs at any point of time. "Old" NAKs are NAKs for sequence numbers which have already seen an earlier NAK and the corresponding bit (see above) is set. Note that though we do not reduce the rate for old NAKs, we do measure RTT samples using them.

2.3.2 Ambiguity Issues and the Silence Period

To reduce the possibility of the retransmission ambiguity problem similar to the one seen in TCP, we do not record a timestamp when a retransmitted packet is sent. Note that our RTT sample measurement procedure runs a residual risk of including some RTT samples which are

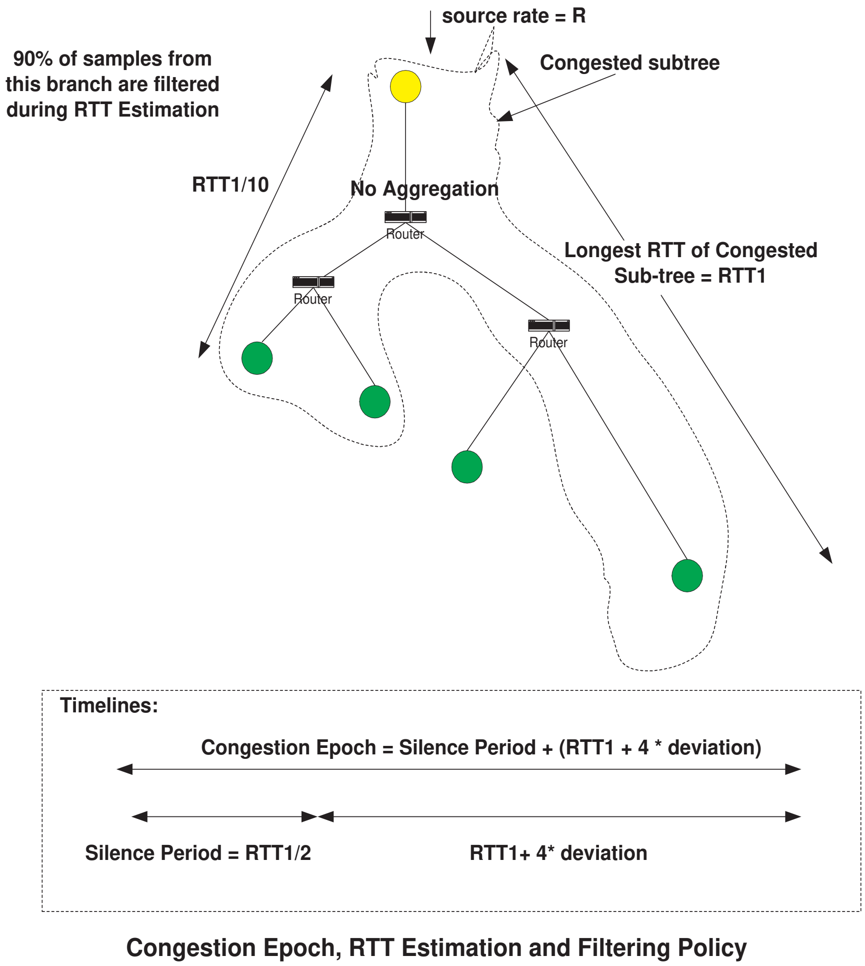


Figure 2: Illustration of Congestion Epoch, RTT Estimation and Filtering Issues

over-estimates of RTT *when retransmitted packets are lost* and the receivers resend their NAKs. To reduce this risk, we prescribe a silence period of RTT/2 just after the rate reduction has been effected in addition to the regular setting of the congestion epoch. This is illustrated in Figure 2. Any further support from underlying RMT protocols to minimize the probability of loss of retransmitted packets would help our RTT estimation algorithm. For example, if the receivers could optionally mark such NAKs (with a bit) as "resent" and the bit setting is preserved in any NAK aggregation, the congestion control algorithm could exclude such NAKs in the calculation of RTT samples.

The silence period mentioned above is also considered to be part of the congestion epoch. It helps in cases where there may be some miscellaneous control traffic (like NCFs in PGM [13]) which is not be controlled by the source rate. The silence period allows time for any queue accumulation caused by such traffic to be drained (though it cannot be guaranteed in general). The silence period also compensates for the fact that the NAK is a "late" measure of an RTT. That is, if the congested sub-tree structure changes abruptly, the RTT estimate will not reflect it when the first new NAK is received at the source and the congestion epoch is initialized. The silence period gives sufficient time for more RTT samples to be seen and a better setting of the congested epoch to be reached. As mentioned earlier, the silence period is also similar to the period of silence resulting from TCP's fast recovery procedure.

As noted above, the silence period combined with aggregation delays in NAKs or random backoff delays may lead to an overestimate of RTT in some cases. The cost of such overestimation is not significant. This is because, unlike the TCP timeout when no packets are sent, our scheme maintains the transmission rate at R/2 after the silence period which ensures progress. Only the subsequent linear increase or the beginning of the next congestion epoch may be delayed by a longer amount by such RTT over-estimates. If it is the subsequent congestion epoch detection which is delayed, the silence period ensures that the cost in terms of packets lost in the epoch is not significantly increased.

2.3.3 RTT Estimation and Timer Settings

The next phase in RTT estimation is to prune the set of RTT samples. Since we are interested in setting the congestion epoch close to the maximum RTT of the congested sub-tree, we reject 90% of the RTT samples smaller than half the current smoothed average RTT estimate as illustrated in Figure 2. The smoothed average and smoothed mean deviation is calculated only for the remaining samples. The objective is to reduce the deviation in the remaining samples and to bias the average RTT estimate higher. More importantly, if we get RTT samples spanning a number of orders of magnitude, our goal is to bias the estimate towards the highest order of magnitude while keeping the option of reducing the average if the large RTT samples stop coming. The cost of this procedure is slower adaptation to a change in the composition of the congested sub-tree (especially when the max-RTT of the sub-tree reduces due to longer RTT area being congested no longer). But this is again only a transient cost.

The canonical congestion epoch length is set to the smoothed average RTT plus *four times* the smoothed mean deviation of estimates (see Figure 2) similar to TCP timeout setting. According to the Chebyshev's theorem, this procedure would give us at least a 93.75% probability that the congestion epoch is larger than the maximum RTT of the congested sub-tree irrespective of the distribution of samples (i.e. even though the RTT samples may be received from different parts of the congested sub-tree). Note that the actual congestion epoch also includes the RTT/2 silence period imposed immediately after rate-reduction which would improve upon the probability calculated above.

The rate-increase timer is set to smoothed average RTT plus *twice* the smoothed mean deviation. The justification for this setting is based upon empirical evidence presented in section 3.3. The key idea is that we would like to have an estimate of RTT which includes only *average queuing delays* and *not maximum queuing delays*. This is because the equivalent TCP mechanism of window increase based upon ack clocking [1] measures average (and not maximum) queueing delays as part of the RTT. Since we do not have a good way to estimate average queuing delays, we choose the setting of the rate-increase timer when RMT is fair towards TCP. This setting turns out to be smoothed average RTT plus *twice* the smoothed mean deviation (see section 3.3).

As a miscellaneous issue, we note that TCP initializes its timeout to twelve ticks (or roughly 6 seconds at a 500 ms clock granularity). If we initialize our epoch to a small value, we would run the risk of incur a few excess rate-reductions initially. A reasonably safe initialization value is the maximum RTT of the multicast tree. The cost of this conservative initialization is a slower linear increase phase till the first congestion epoch is reached (and NAKs bring RTT samples).

2.3.4 Effect of NAK Aggregation

One important issue with NAK aggregation is that NAKs which come later at the aggregator are suppressed, and the source never gets the timing information implicit in them. In other words, the RTT estimated is in fact closer to the *smallest* RTT of the congested sub-tree and *not* anywhere close to the largest RTT of the sub-tree. However, as we noted earlier in section 2.1.3, this issue is moot if does not affect the *number* of congestion epochs required to correct a load-capacity imbalance. Assume that the load-capacity imbalance is on a common branch leading to multiple receivers and it can theoretically be corrected by one rate-reduction. Any NAKs which are still in the system after the end of the congestion epoch at the source will be absorbed by the aggregators. This argument can be extended for the case when multiple rate-reductions are necessary on the common link.

If the losses in the tree are randomized (because of congestion situations in different parts of the network), the aggregators will pass the NAKs through from different parts of the tree. Our sample rejection technique will reject samples that are very small, and average out the samples within the highest order of magnitude. The use of the mean deviation and the silence period then ensures a low probability of spurious congestion epochs (and associated rate-reductions). The same argument holds in the case of absence of aggregators and in the case of imperfect or inconsistent aggregation. In other words, we can rely on the combination of underlying RMT aggregation mechanisms, our RTT estimators and silence periods in congestion epochs to avoid unnecessary rate-reductions in spite of erraneous RTT estimates and thus avoid the drop-to-zero problem.

2.3.5 Extending NAK-based Semantics to Diverse RMT Protocols

Our first note on the generic nature of this scheme is that the NAK-based semantics are readily applicable to NAK-based protocols including PGM [13], RMTP-II (with NAK-enabled) [43], and SRM [16]. The semantics can be easily extended to bitmap-based RMTs (where the negative feedback is captured in a bitmap) like MFTP [38] or RMTP-II (some versions) [43].

These semantics are hard to preserve in a *pure* ACK or HACK/TRACK based system (like the original RMTP-I). This is because a HACK is delivered to the source only when all receivers have implicitly or explicitly acknowledged this packet. Unlike TCP, these systems do not allow "duplicate acks" which enables the early detection of congestion. However, out-of-order HACKs [43] can be used to detect packet losses and these can be viewed as proxies for NAKs in our scheme. The RTT estimates should therefore be taken *only* upon receipt of out-of-order HACKs. But the RTT sample measured by these HACKs reflect the maximum RTT of the *entire* tree and not a RTT sample from the congested sub-tree. The use of these estimates would mean a compromise - rate increase intervals and congestion epochs would have to be based upon maximum RTT of the entire tree and not just the sub-tree.

This compromise implies that all congestion epochs (and consequently periods of severe overload) would last longer, delaying rate-decreases. The longer periods of overload would in general lead to more packet losses and larger queue lengths, though it can be ameliorated by using large buffers and random early drop (RED-like) policies. The impact on rate-increase is insignificant on the long-term, because the average rate achieved is independent of the period of rate oscillations which is the key parameter affected by RTT. In other words, assuming that the loss rates are very small in percentage terms, the average rate is expected to be similar to a NAK-based

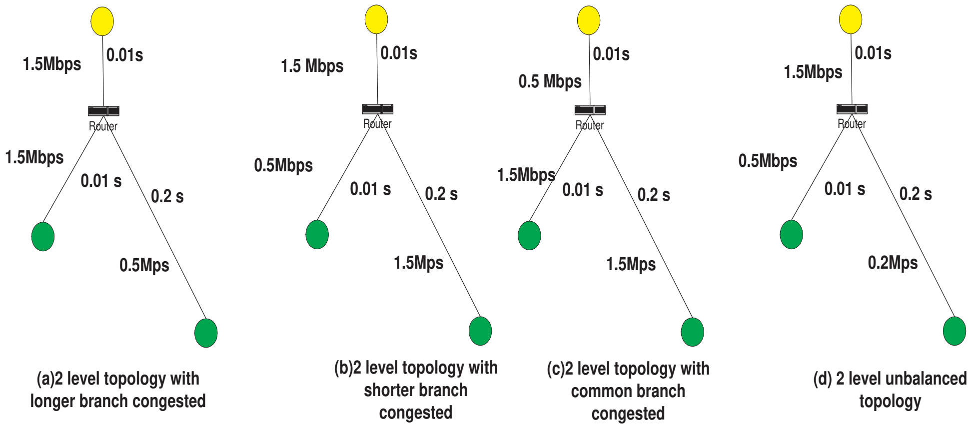


Figure 3: Configurations Used to Illustrate RTT Estimation Performance Issues

system (see section 3.1 for a related performance result).

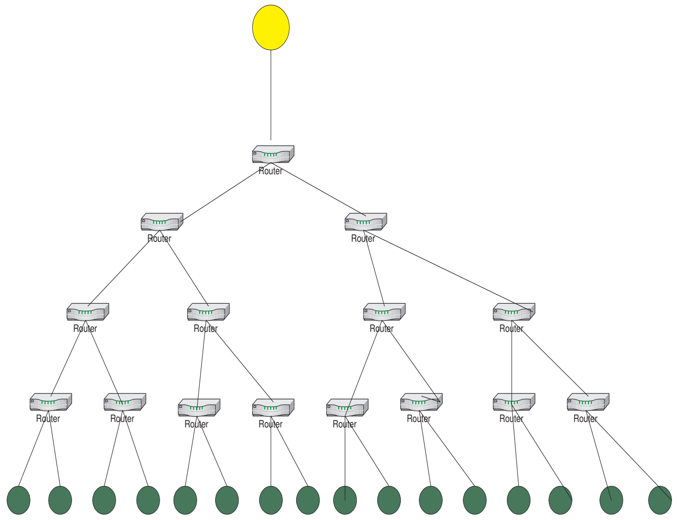
For schemes with designated retransmissors (DRs), even if congestion feedback can be forwarded to the source from DRs *periodically* (not for every NAK) during periods of consistent congestion, these indications allow rough estimation of congested sub-tree RTTs useful for setting up congestion epochs. Observe that the emphasis is on the source having reliable access to the information that congestion in fact occured in different parts of the tree. The scheme is not sensitive otherwise to the frequency of feedback during congestion unlike Bhattacharya et al's filters [3, 5].

The SBMCC scheme therefore is fairly generic. Provided any form of RMT control feedback which can convey minimal, but reliable congestion information during congestion periods to the source, it can provide generic source-based congestion control. The demands on new control traffic is indeed very low in general (and zero in several cases) which allows a large deployment space. The fidelity of timing information available from such RMT feedback only affects the optimization of congestion response, but not long-term stability or the average rate achieved. The scheme can also be easily extended to multi-sender multicast without increasing the state requirements in receivers over-and-above their base RMT requirements. This is because the scheme is source-based. Code modifications and the (*,G) state requirement is only at the source. Each source's traffic will compete with the other sources' traffic as they were independent flows.

3 Performance Analysis

In this section, we illustrate the performance of the SBMCC scheme using simple simulation experiments. Figures 3, 4, and 5 show the configurations we use in our various experiments. The buffer sizes in all experiments are 125 pkts each, and the packet sizes are 560 bytes. The initial RTT estimate is set conservatively in each of these simulations to be larger than the maximum congested-subtree RTTs (including maximum queueing delays). We assume for simplicity that the receiver does not have any random timer backoff before sending NAKs.

We use the simple two-level tree configurations shown in Figure 3 to illustrate issues with RTT estimation. The position of the bottleneck is varied in the first three configurations, and the last



(a) A 5 level balanced tree topology.All links have a propagation delay of 0.01 sec and a bandwidth of 0.5 Mbps

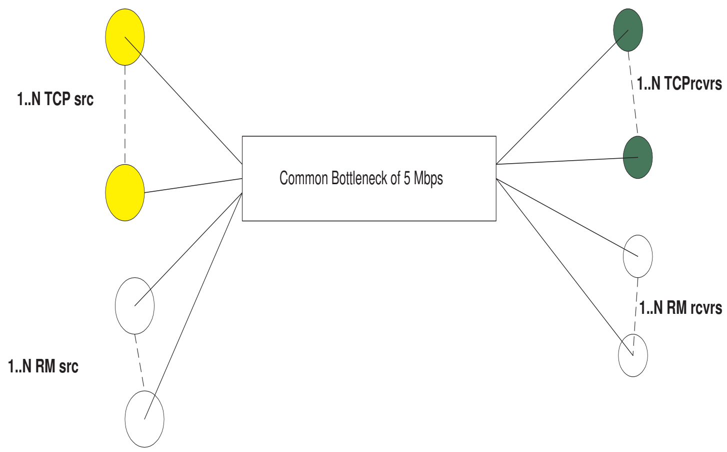
Figure 4: Configuration Used to Illustrate Performance with Multiple Tree Levels

configuration has two bottlenecked links (at different rates). In all cases the RTTs of the paths to the destinations are different. One of our simulations also uses the RED drop policy [14] instead of the default drop-tail policy at the bottlenecks. Figure 4 shows a multi-level balanced tree configuration which illustrates that our source-based approach scales with the number of receivers without suffering from the drop-to-zero problem. Figure 5 shows a configuration used to test TCP-friendliness, where N TCP flows and N RMT flows compete for bandwidth on a single bottleneck.

3.1 RTT Estimation

Figure 6(a) and (b) show the results of a simulation using a two-level tree with an initial topology as indicated in Figure 3(a). the topology changes after 500 sec of the beginning of the simulation to that of Figure 3(b), i.e., the bottleneck shifts. Figure 6(b) shows that the RTT estimate during the first 500 sec of the simulation is biased towards the RTT of the congested sub-tree (which is the longer branch). Figure 6(a) (the corresponding source rate graph) indicates that the rate is reduced to half the original rate and no more, thus avoiding the drop-to-zero problem. Note that after the first 500 sec, the slope of the linear rate increase phases are steeper. This is because the corresponding RTT estimates based upon the new congested sub-tree are smaller.

A notable feature is that even though the period of oscillation is reduced, the average rate during each cycle is the same before and after the topology change. This is because the source cannot send at an average rate faster than the bottleneck rate. In other words, the long term average rate is same irrespective of the RTTs measured and the corresponding rate-oscillation periods. The effect of measuring smaller RTTs is seen as smaller queue lengths at the bottleneck (and lower packet drop rates on the long term). As explained in section 2.3.5 we use this property to relax the congested sub-tree semantics and apply the scheme to pure ACK- or HACK-based RMTs.



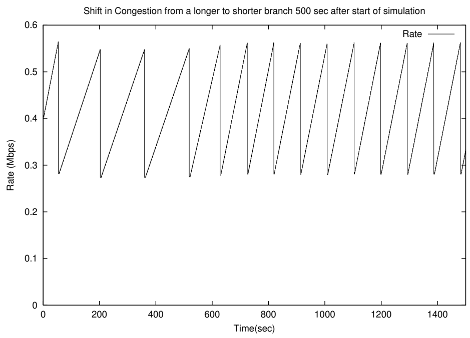
(a) N TCP flows and N Multicast flows competing over a single bottleneck link of 5 Mbps.N ranges from 1 to 80

Figure 5: Configuration Used to Illustrate TCP-friendliness of the SBMCC Scheme

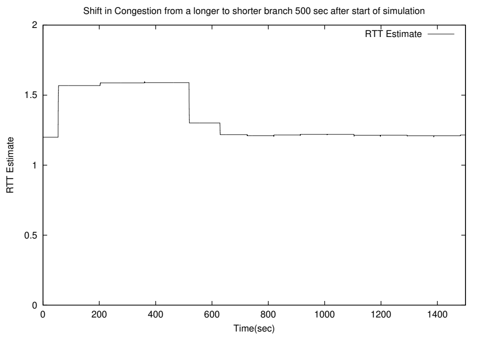
Figure 7(a) and (b) show the results based upon the configuration shown in Figure 3(d) (called the "unbalanced configuration") where the different branches have different bottleneck rates (0.5 Mbps and 0.2 Mbps) and the initial source rate is 0.6 Mbps. This situation requires two rate-reductions to solve the congestion problem. Since there is aggregation at the router, several NAKs from the longer RTT branch (for packets lost on both links) are suppressed. As a result, there is a transient period which includes the first linear increase phase where the RTT computed is erraneous i.e. smaller than the max RTT of the congested sub-tree (see Figure 7(b)). The price of this error is an extra transient rate reduction initially, and a steeper rate-increase slope for the first rate increase phase (see Figure 7(a)). However, this is only a transient problem. There is no drop-to-zero problem in steady state after the first cycle of decrease/increase because the congestion on one of the branches is solved and the RTT estimation process converges.

Figure 8 shows the effect of NAK aggregation on the RTT estimator. The configuration used for this simulation is the one shown in Figure 3(c) where the first link which is common to both the receivers is congested at 0.5 Mbps. We plot two curves, one which corresponds to the use of aggregation at the router, and one which corresponds to no aggregation at the router. Since the same packet losses are detected at both the receivers, both send NAKs towards the source. If aggregation is used, the NAKs from the longer RTT receiver are always absorbed at the router and not sent to the source. Therefore, as seen in Figure 8(b), the source will measure the congested sub-tree RTT as the *smallest* RTT of the sub-tree. However, as seen in the corresponding source rate graph Figure 8(a), there is no drop-to-zero effects because of this erraneous RTT estimation. The reason of this behavior is two-fold. First, there are no packets dropped after the effect of the rate decrease and silence period reaches the bottleneck. Second, the NAKs of earlier packets lost which come from the long RTT branch are absorbed at the aggregator (also see section 2.3.4).

Figure 8 also plots graphs when aggregation is not used at the router. Without aggregation, the NAKs from both branches reach the source. As a result the RTT estimate seen in Figure 8(b) is higher in this case and assumes a value to intermediate to the small and large RTT values.

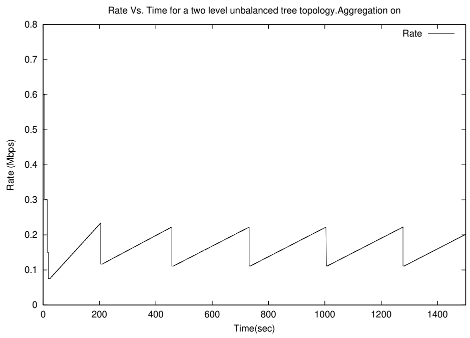


(a) Source Rate

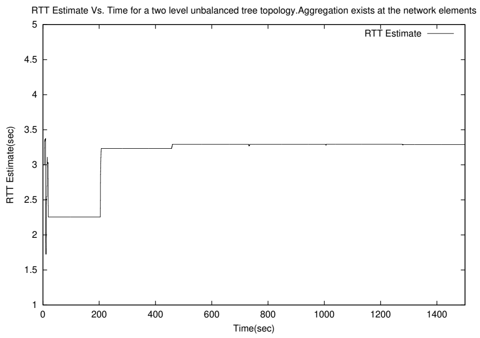


(b) RTT Estimate

Figure 6: Simulation results with a Two-level Tree with NAK Aggregation. The bottleneck shifts from the larger to the smaller RTT at 500s.

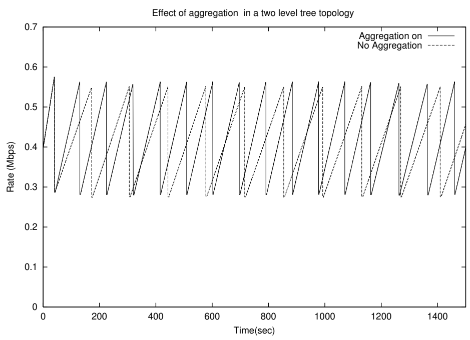


(a) Source Rate

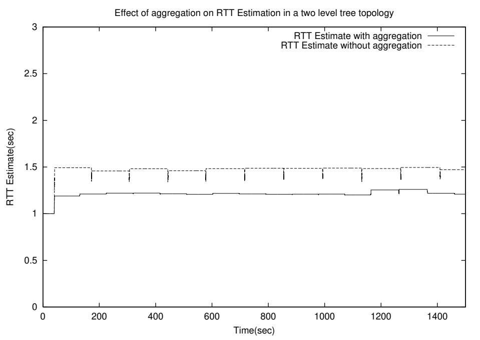


(b) RTT Estimate

Figure 7: Simulation results with a Two-level Unbalanced Tree with NAK Aggregation

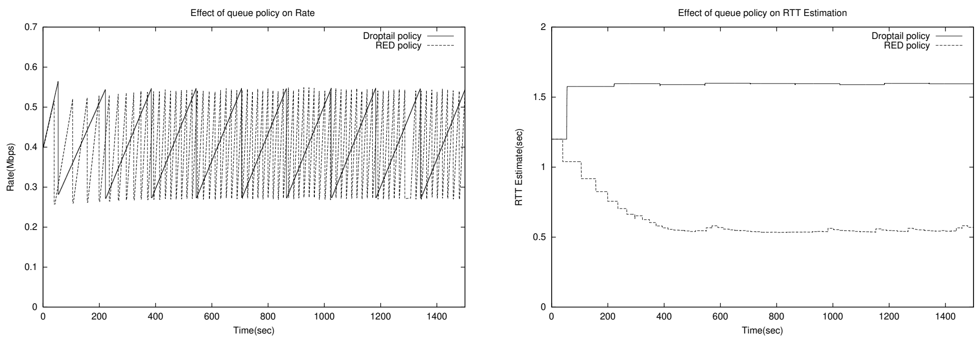


(a) Source Rate



(b) RTT Estimate

Figure 8: Aggregation vs Absence of Aggregation



(a) Source Rate

(b) RTT Estimate

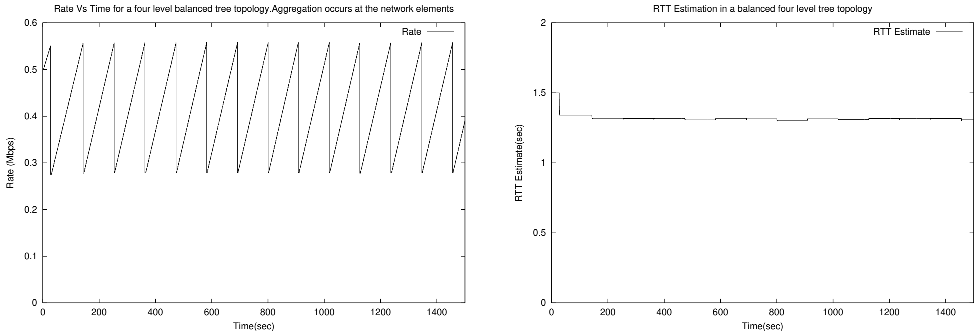
Figure 9: Effect of RED Drop Policy on RTT Estimation

However, the addition of the mean deviation and the use of the silence period (RTT/2) in the congestion epoch ensures that there is no drop-to-zero problem i.e, no excess rate decreases (Figure 8(a)). Since RTT estimates are larger, the linear increase phases in the graph have a smaller slope. As mentioned earlier, the smaller rate increase slope does not affect the average rate over long-terms, but only the queue lengths. As minor note, in PGM we have assumed that NCFs are not rate-controlled which leads to NCF-floods near the source if we do not aggregate NAKs. This assumption is not necessary in general. But an important lesson we learnt is that as long as there exists control traffic which is not source rate-controlled, there is a risk of the drop-to-zero problem if there is no aggregation especially for large trees. But aggregation is necessary in PGM to deal with the implosion problem in any case.

Finally, we examine the case of a bottleneck with the RED packet drop policy instead of the regular drop-tail policy. The configuration used is Figure 3(a) i.e. the longer RTT branch is congested, and the comparative simulation results (with and without RED) are shown in Figure 9. RED drops packets early when the average queue crosses a small threshold whereas drop-tail drops packets only when the queue is full. As a result, the RTTs measured when NAKs are seen reflect a much smaller queueing delay. The graph Figure 9(b) shows that with RED, the RTT measured drops dramatically and comes close to the sum of propagation and transmission delays (plus a queueing delay one order of magnitude smaller than that seen with the drop-tail policy). As a result, the period of additive increase/multiplicative decrease (AIMD) oscillations is dramatically reduced (Figure 9(a)). There is no drop-to-zero problem inspite of the randomized nature of packet drops because of the addition of the mean deviation factor and the silence period. As mentioned earlier, rapid oscillations seen in Figure 9(a) does not increase the average rate of transmission of the source; it only affects the average queue length.

3.2 Performance Scaling with Multiple Receivers

We tested the performance of the scheme with upto 32 receivers connected in a multi-level balanced tree as shown in Figure 4. In this balanced tree configuration, all links are of 0.5 Mbps and the link propagation delays are 0.01 s each. In other words, the RTT is not affected by long links (like the 0.2 s link used in the earlier configurations. The performance graphs are shown in Figure 10. Observe that the RTT computation stabilizes quickly from a higher



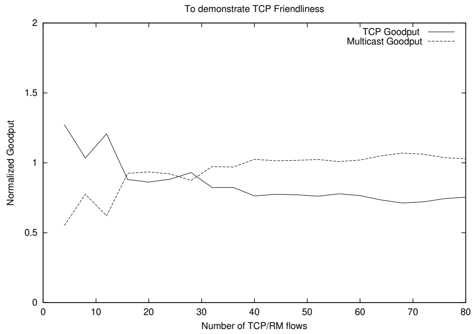
(a) Source Rate

(b) RTT Estimate

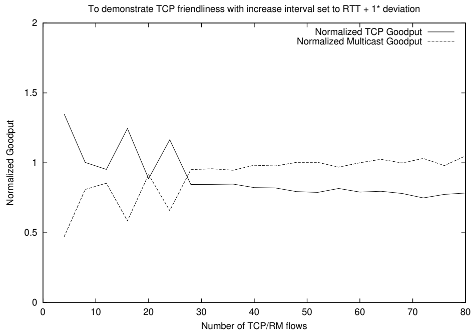
Figure 10: Simulation results with a Five-level Tree

initialization value to the max RTT of the tree, and there is no drop-to-zero effects - only regular AIMD oscillations with single rate-decrease in each oscillation. This shows that the scheme has a reasonable potential to scale. We are currently making simulator changes in ns to allow large scale simulations and will verify this claim for larger trees.

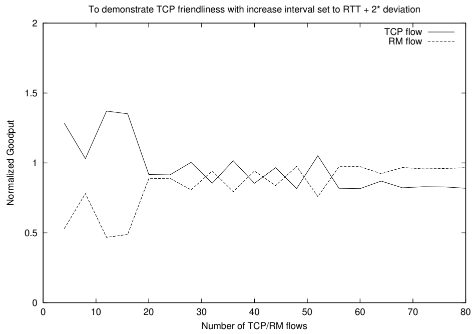
3.3 TCP-friendliness



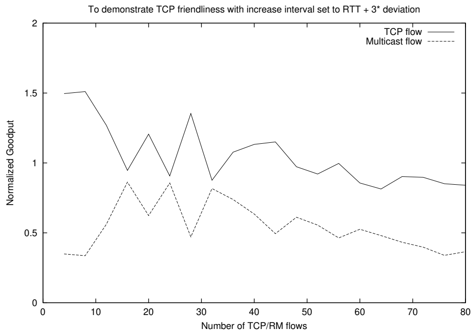
(a) Increase Interval = MeanRTT



(b) Increase Interval = MeanRTT + 1*Deviation



(c) Increase Interval = MeanRTT + 2*Deviation



(d) Increase Interval = MeanRTT + 3*Deviation

Figure 11: TCP Friendliness with Different Values of Increase Interval

We have designed the scheme carefully to be friendly to TCP. Specifically, several of the scheme features mimick TCP behavior:

- The scheme uses a loss-driven additive increase/multiplicative decrease (AIMD) policy with similar parameters like TCP. Specifically, the additive rate increase used is MSS/RTTEstimate which is equivalent to the TCP window increase during congestion avoidance. We do not attempt to emulate slow start.

- Upon detection of congestion, we effect a multiplicative decrease of half the current source rate, similar to TCP. Further, this decrease is performed on the first new NAK seen which is similar to TCP's window reduction upon seeing three duplicate acks.

- The additive increase is done once per RTT + 2 × D (see section 2.3.3) which is a reasonable estimate of the RTTs including *average* queueing delays and *not the maximum queueing delays.*

- The silence period of RTT/2 is similar to the silence period in TCP fast recovery.

- The average rate after the silence period during the remaining part of the congestion epoch is R/2 (R is the rate before congestion was detected) which is similar to the average rate maintained during fast recovery in TCP.

- The congestion epoch is set based upon the smoothed average and mean deviation of RTT samples similar to the algorithm used to set TCP timeout.

- The filtering of NAKs during the congestion epoch is similar to the filtering of loss indications in NewReno and SACK (no new window reductions for losses within the same window).

- Our concept of "New" NAKs (see section 2.3), and the policy of not recording local timestamps for retransmitted packets addresses an ambiguity problem similar to the retransmission ambiguity problem. The TCP problem is addressed by Karn's algorithm which is different in nature from our mechanisms used in this context.

We also used the configuration shown in Figure 5 to evaluate the TCP friendliness of the algorithm. In this configuration we have N TCP connections competing with N RMT flows on a single bottleneck. We varied N from 1 to 80, and plotted the normalized average goodput of a TCP flow vs an RMT flow (Figure 11). At any value of N, the average of the number assigned to RMT and the number assigned to TCP is equal to the total goodput divided by the bottleneck bandwidth. In other words, if TCP shows a normalized value V_{tcp} larger than the value for RMT V_{rmt}, it is grabbing $(\frac{V_{tcp}-V_{rmt}}{V_{rmt}}) \times 100$ % more bandwidth than an average RMT flow for that value of N. The goal of TCP friendliness is to have these curves be close to each other, not diverge, and be close to unity. We found that the performance of the scheme in this regard was sensitive to the value of the rate increase interval. We experimented with four different values of the rate increase interval (SRTT + $i \times D$, where $i = 0..3$). We find that the best performance is in Figure 11(c) where $i = 2$, i.e., we add two mean deviations to the average to determine the rate increase interval. In the other graphs, for $i < 2$, the average RMT flow gets a larger share compared to TCP, and for $i > 2$, the RMT flow is rapidly beaten down by TCP which grabs a dominant share of the bandwidth. These results show that a reasonable selection of parameters exists such that the scheme is TCP friendly.

4 Summary

In summary, the SBMCC scheme has simplified the requirements for source-based multicast congestion control by the use of the concepts of congestion epoch and RTT estimation of the congested sub-tree. This has allowed us to develop a simple generic core algorithm (the pseudo code presented in appendix A is less than a page) which has very low control traffic requirements in general and zero traffic requirements from NAK-based protocols like PGM and RMTP. The scheme's weak requirements on RTT estimation allows it to leverage any available congestion related timing information from underlying RMT reverse control traffic. This feature allows it to be adapted to a number of RMT protocols, and hence highly deployable. The concepts and techniques developed to allow a high degree of deployability of source-based multicast congestion control are therefore the key contributions of this paper.

5 Acknowledgements

We would like to acknowledge all members of the RMRG group who gave us detailed feedback on a predecessor of this work, and in particular Supratik Bhattacharya, Dah-Ming Chiu, Jon Crowcroft, Mark Handley, Sally Floyd, Ken Miller, Luigi Rizzo, Tony Speakman and Brian Whetton. Thanks are due to Paul Stirpe (Reuters) and Gursel Taskale for sponsoring this work, detailed discussions and teaching us the user perspective of RMT. We would like to thank Hari Balakrishnan (MIT) and Sonia Fahmy (Purdue) for their unique perspectives and detailed critiques. Finally we would like to thank our colleagues Markus Kuhn for developing a PGM implementation on Linux, Jiang Li and Hitesh Raigandhi for engaging group discussions.

References

[1] M. Allman, V. Paxson and W. Stevens, "TCP Congestion Control," *IETF Internet RFC 2581, Proposed Standard*, April 1999.

[2] K. Almeroth, "The Evolution of Multicast: From the MBone to Inter-Domain Multicast to Internet2 Deployment," *IEEE Network Special Issue on Multicasting*, January/February 2000 (to appear)

[3] S. Bhattacharya et al, "Design and Analysis of Loss Indication Filters for Multicast Congestion Control," *submitted to INFOCOM 2000*, 1999.

[4] S. Bhattacharya, J. Kurose, D. Towsley, "Efficient Multicast Flow Control using Multiple Multicast Groups," *University of Massachusetts, Amherst, CMPSCI Technical Report TR 97-15*, 1997.

[5] S. Bhattacharyya, D. Towsley, J. Kurose, "The Loss Path Multiplicity Problem in Multicast Congestion Control," *University of Massachusetts, Amherst, CMPSCI Technical Report TR 98-76*, 1997.

[6] J.-C. Bolot, T. Turletti, and I. Wakeman, "Scalable feedback control for multicast video distribution in the internet," *ACM SIGCOMM*, (London, England), pp. 58–67, ACM, Aug. 1994.

[7] S. Bradner, A. Mankin, V. Paxson, and A. Romanow, "IETF criteria for evaluating reliable multicast transport and application protocols," *IETF Internet RFC 2357*, June 1998.

[8] J. Byers, M. Luby, M. Mitzenmacher, A. Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data," *Proceedings of ACM SIGCOMM'98*, Vancouver, August 1998.

[9] B. Cain, T. Speakman, D. Towsley, "Generic Router Assist (GRA) Building Block Motivation and Architecture," *IETF Internet Draft, draft-ietf-rmt-gra-arch-00.txt*, October 1999.

[10] D. Chiu, "Using Dynamic window and rate control for RM," *Proceedings of the 5th RMRG meeting*, London, England. Available from: http://www.east.isi.edu/RMRG/

[11] D. DeLucia, Hughes, and K. Obraczka, "Multicast Feedback Suppression Using Representatives," *IEEE INFOCOM'97*, April 1997.

[12] C. Diot et al, "Deployment issues for the IP Multicast Service and Architecture,"

[13] D. Farinacci, A. Lin, T. Speakman, and A. Tweedly, "PGM reliable transport protocol specification," *Internet Draft, Internet Engineering Task Force*, Aug. 1998. Work in progress.

[14] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, Vol. 1, No. 4, August 1993, pp. 397–413.

[15] S. Floyd, and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm," *IETF Experimental RFC 2582,* April 1999.

[16] S. Floyd, V. Jacobson, S. McCanne, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing", *Proc. of ACM SIGCOMM 95*, Aug 1995 pp. 342-356.

[17] J. Golestani, "Fundamental observations on Multicast Congestion Control in the Internet", *IEEE INFOCOM'99*, April 1999. Available from: http://research.ivv.nasa.gov/RMP/links.html

[18] M. Handley, S. Floyd, B. Whetton, "TCP Friendly Multicast Congestion Control," *Proceedings of the 7th RMRG meeting*, Pisa, Italy, June 1999.

[19] M. Handley, "Multicast address allocation protocol (AAP)," *Internet Draft, Internet Engineering Task Force*, Jun 1999. Work in progress.

[20] M. Handley et al, "The Reliable Multicast Design Space for Bulk Data Transfer," *IETF Internet Draft, draft-ietf-rmt-design-space-00.txt*, October 1999.

[21] M. Handley and J. Crowcroft, "Network text editor (NTE) a scalable shared text editor for MBone," *ACM Computer Communication Review*, vol. 27, pp. 197-208, Oct. 1997.

[22] M. Hofmann, J. Nonnenmacher, J. Rosenberg and H. Schulzrinne, "A Taxonomy of Feedback for Multicast," *Internet Engineering Task Force, draft-hnrs-rmt-avt-feedback-00.txt*, Jun. 1999.

[23] IETF, "Reliable Multicast Transport (rmt)," *IETF Working Group*, 1999. http://www.ietf.org/html.charters/rmt-charter.html

[24] IRTF, "The Reliable Multicast Research Group (rmrg)," *IRTF research group*, 1999. http://www.east.isi.edu/RMRG/

[25] V. Jacobson, "Congestion Avoidance and Control," *Proceedings of the SIGCOMM'88 Symposium*, pp. 314-32, August 1988.

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

[27] B.N. Levine and J.J. Garcia-Luna-Aceves, "A Comparison of Reliable Multicast Protocols," *ACM Multimedia Systems Journal*, August 1998.

[28] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgement Options," *IETF RFC 2018*, April 1996.

[29] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven Layered Multicast," Proceedings of ACM SIGCOMM, August 1996, pp. 117-130.

[30] T. Montgomery, "A Loss Tolerant Rate Controller," *NASA IV&V Technical Report, NASA-IVV-97-011*, August 1997.

[31] T. Montgomery, B. Whetten, M. Basavaiah, S. Paul, N. Rastogi, J. Conlan, and T. Yeh, "THE RMTP-II PROTOCOL," *Internet Draft, Internet Engineering Task Force*, April 1998. Work in progress

[32] J. Nonnenmacher and E. W. Biersack, "Optimal multicast feedback," in IEEE Infocom , (San Francisco, California), p. 964, March/April 1998.

[33] K. Obraczka, "Multicast Transport Protocols: A Survey and Taxonomy," *IEEE Communications Magazine*, 1997.

[34] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," *Proceedings of SIGCOMM'98*, Vancouver, August 1998.

[35] I. Rhee, N. Ballaguru, G. N. Rouskas, "MTCP: Scalable TCP-like Congestion Control for Reliable Multicast," *TR-98-01, Department of Computer Science, NCSU*, January 1998,

[36] L. Rizzo, "Effective erasure codes for reliable computer commu- nication protocols," *ACM Computer Communication Review*, vol. 27, pp. 24-36, Apr. 1997.

[37] L. Rizzo, L. Vicisano, "A Reliable Multicast data Distribution Protocol based on software FEC techniques", *Proc. of The Fourth IEEE Workshop on the Architecture and Implementation of High Performance Communication Systems (HPCS'97)*, Sani Beach, Chalkidiki, Greece June 23-25, 1997.

[38] K. Robertson, K. Miller, M. White, and A. Tweedly, "StarBurst multicast file transfer protocol (MFTP) specification," *Internet Draft, Internet Engineering Task Force*, April 1998. Work in progress.

[39] H. Schulzrinne, et al, "RTP: A Transport Protocol for Real-Time Applications," *Internet RFC 1889*, 1997.

[40] D. Sisalem, H. Schulzrinne, "End-to-end quality of service control using adaptive applications," *Proceedings 5th International Workshop on QoS (IWQPS'97)*, May 1997.

[41] L. Vicisano, L. Rizzo, J. Crowcroft, "TCP-like congestion control for layered multicast data transfer," *UCL Research Note RN/97/75*, 1997.

[42] B. Whetton, J. Conlan, "A Rate Based Congestion Control scheme for Reliable Multicast," *Proceedings of the 5th RMRG meeting*, , London, England. Available from: http://www.east.isi.edu/RMRG/

[43] B. Whetton, G. Taskale, "An Overview of Reliable Multicast Transport Protocol II," *IEEE Network Magazine*, January/February 2000.

A Pseudo Code

Algorithm 1 Pseudo Code

average_rtt: TCP-like average round trip time
current_rtt: The most recent round trip time sample
mean_deviation: TCP-like mean deviation of the round trip time
congestion_epoch_start_time: The time when the rate was last reduced
NAK_Filter_Timer: A timer which is used for the NAK filtering interval
Silence_Timer: A timer which is used for the silence period.
congestion_state: Either: CONGESTED and NOT_CONGESTED
RTT_Estimator() estimates the average_rtt and mean_deviation of RTT using the same algorithm as used in TCP [25]. In addition, it pre-filters a sample with a 90% probability if it is smaller than $0.5 \times$ average_rtt.

```
EVENT: A NAK is received for a packet sent at time T1
if (congestion_state == NOT_CONGESTED) then
  if (This is a New NAK) then
    Cut rate by half. Set a "old NAK" bit for this sequence number.
    Set silence_period to average_rtt/2. Indicate beginning of silence period
    congestion_state = CONGESTED
    congestion_epoch_start_time=current_time
    Call RTT_Estimator()
    Set the NAK_filter_timer to silence_period + average_rtt + 4 * mean_deviation
  else
    Call RTT_Estimator()
  end if
else
  Call RTT_Estimator()
end if

EVENT: NAK_Filter_Timer expires: congestion_state = NOT_CONGESTED

EVENT: Silence_Timer expires: Indicate end of silence period

EVENT: Rate_Increase_Timer expires
if (congestion_state == NOT_CONGESTED) then
  Increment rate by MSS/(rtt_average + 2 * mean_deviation)
end if
Set Rate_Increase_Timer to rtt_average + 2 * mean_deviation
```