REIHE INFORMATIK
13/2001

Extending Equation-based Congestion Control to Multicast Applications

Jörg Widmer and Mark Handley
Universität Mannheim
Praktische Informatik IV
L15, 16
D-68131 Mannheim

Extending Equation-based Congestion Control to Multicast Applications

Jörg Widmer * Mark Handley †

Abstract

In this paper we introduce TFMCC, an equation-based multicast congestion control mechanism that extends the TCP-friendly TFRC protocol from the unicast to the multicast domain. The key challenges in the design of TFMCC lie in scalable round-trip time measurements, appropriate feedback suppression, and in ensuring that feedback delays in the control loop do not adversely affect fairness towards competing flows. A major contribution is the feedback mechanism, the key component of end-to-end multicast congestion control schemes. We improve upon the well-known approach of using exponentially weighted random timers by biasing feedback in favor of low-rate receivers while still preventing a response implosion. We evaluate the design using simulation, and demonstrate that TFMCC is both TCP-friendly and scales well to multicast groups with thousands of receivers. We also investigate TFMCC's weaknesses and scaling limits to provide guidance as to application domains for which it is well suited.

Keywords: *congestion control, multicast, single-rate, TCP-friendliness, feedback suppression*

1 Introduction

It is widely accepted that one of several factors inhibiting the usage of IP multicast is the lack of good, deployable, well-tested multicast congestion control mechanisms. To quote [10]:

> *The success of the Internet relies on the fact that best-effort traffic responds to congestion on a link by reducing the load presented to the network. Congestion collapse in today's Internet is prevented only by the congestion control mechanisms in TCP.*

We believe that for multicast to be successful, it is crucial that multicast congestion control mechanisms be deployed that can co-exist with TCP in the FIFO queues of the current Internet.

* University of Mannheim, Germany
† AT&T Center for Internet Research at ICSI (ACIRI)

The precise requirements for multicast congestion control are perhaps open to discussion given the efficiency savings of multicast, but we take the conservative position that a multicast flow is acceptable if it achieves no greater medium-term throughput to any receiver in the multicast group than would be achieved by a TCP flow between the multicast sender and that receiver.

Such a requirement can be satisfied either by a single multicast group if the sender transmits at a rate dictated by the slowest receiver in the group, or by a layered multicast scheme that allows different receivers to receive different numbers of layers at different rates. Much work has been done on the latter class [12, 18, 4], but the jury is still out on whether any of these mechanisms can be made safe to deploy.

This paper describes TCP-Friendly Multicast Congestion Control (TFMCC), which belongs to the class of single rate congestion control schemes. Such schemes inevitably do not scale as well as layered schemes. However, they are much simpler, match the requirements of some applications well, and we will demonstrate that they can scale to applications with many thousands of receivers. These schemes also suffer from degradation in the face of badly broken links to a few receivers – how to deal with such situations is a policy decision, but we expect that most applications using a single-rate scheme will have application-specific thresholds below which a receiver is compelled to leave the multicast group.

TFMCC is not the only single-rate multicast congestion control scheme available. In particular, Pragmatic General Multicast Congestion Control (PGMCC) [17] is also a viable solution with some nice properties and a certain elegant simplicity. However, TFMCC and PGMCC differ considerably in the smoothness and predictability of their transmission. We will argue that both are appropriate solutions, and that some applications are better suited to one than the other.

1.1 TFMCC and TFRC

The TCP-friendly Rate Control protocol (TFRC) [5] is a unicast congestion control mechanism intended for applications that require a smoother, more predictable transmission rate than TCP can achieve. TFMCC extends the basic mechanisms of TFRC into the multicast domain.

TFRC is an *equation-based* congestion control scheme. It uses a control equation derived from a model of TCP's long-term throughput to directly control the sender's transmission rate. Basically TFRC functions as follows:

1. The receiver measures the packet loss rate and feeds this information back to the sender.
2. The sender uses the feedback messages to measure the round-trip time to the receiver.
3. The sender uses the control equation to derive an acceptable transmission rate from the measured loss rate and round-trip time (RTT).
4. The sender's transmission rate is then adjusted directly to match the calculated transmission rate.

For full details of TFRC, we refer the reader to [5].

TFMCC follows a very similar design for multicast congestion control. The primary differences are that it is the receivers that measure their RTT to the sender and perform the calculation of the acceptable rate. This rate is then fed back to the sender, the challenge being to do this in a manner which ensures that feedback from the receiver with the lowest calculated rate reaches the sender whilst avoiding feedback implosions. Moreover, we need to make sure than any additional delay imposed to avoid feedback implosion does not adversely affect the fairness towards competing protocols.

2 The TFMCC Protocol

Building an equation-based multicast congestion control mechanism requires that the following problems be solved:

- A control equation must be chosen that defines the target throughput in terms of measurable parameters, in this case loss event rate and RTT.
- Each receiver must measure the loss event rate. Thus a filter for the packet loss history needs to be chosen that is a good stable measure of the current network conditions, but is sufficiently responsive when those conditions change.
- Each receiver must measure or estimate the RTT to the sender. Devising a way to do this without causing excessive network traffic is a key challenge.
- Each receiver uses the control equation to calculate an acceptable sending rate from the sender to itself.
- A feedback scheme must be so devised that feedback from the receiver calculating the slowest transmission rate always reaches the sender, but feedback implosions do not occur when network conditions change.
- A filtering algorithm needs to be devised for the sender to determine which feedback it should take into account as it adjusts the transmission rate.

Clearly, all these parts are closely coupled. For example, altering the feedback suppression mechanisms will impact how the sender deals with this feedback. Many of our design choices are heavily influenced by TFRC, as these mechanisms are fairly well understood and tested. In this paper we will expend most of our efforts focusing on those parts of TFMCC that differ from TFRC.

2.1 Determining an Acceptable Sending Rate

The control equation used by TFRC and TFMCC is derived from a model for long-term TCP throughput in bytes/sec [15]:

$$T_{TCP} = \frac{s}{t_{RTT}\left(\sqrt{\frac{2p}{3}} + \left(12\sqrt{\frac{3p}{8}}\right)p\left(1+32p^2\right)\right)} \quad (1)$$

The expected throughput T_{TCP} of a TCP flow is calculated as a function of the steady-state loss event rate p, the round-trip time t_{RTT}, and the packet size s. Each TFMCC receiver measures its own loss event rate and estimates its RTT to the sender. It then uses Equation (1) to calculate T_{TCP}, which is an estimate of the throughput a TCP flow would achieve on the network path to that receiver under the same network conditions. If the sender does not exceed this rate *for any receiver* then it should be TCP-friendly, in that it does not affect a TCP flow through the same bottlenecks more than another TCP flow would do.

In the following section we will elaborate on how the necessary parameters for the model are computed and how to deal with potentially large receiver sets.

2.2 Adjusting the Sending Rate

The sender will continuously receive feedback from the receivers. If a receiver sends feedback that indicates a rate that is lower than the sender's current rate, the sender will immediately reduce its rate to that in the feedback message.

In order to eliminate a large number of unnecessary messages, receivers will not send feedback unless their calculated rate is less than the current sending rate. However, this leaves us with a problem – how do we increase the transmission rate? We cannot afford to increase the transmission rate in the absence of feedback, as the feedback path from the slowest receiver may be congested or lossy. As a solution we introduce the concept of the *current limiting receiver* (CLR). The CLR is the receiver that the sender believes currently has the lowest expected throughput of the group.[1] The CLR is permitted to

[1] In this respect, the CLR is comparable to the *representative* used in congestion control schemes such as PGMCC.

send immediate feedback without any form of suppression, so the sender can use the CLR's feedback to increase the transmission rate.

The CLR will change if another receiver sends feedback indicating that a lower transmission rate is required. It will also change if the CLR leaves the multicast group – this is normally signaled by the CLR, but an additional timeout mechanism serves as a backup in case the CLR crashes or becomes unreachable.

Normally the way loss measurement is performed limits the possible rate increase to roughly 0.3 packets per RTT, as shown in [5]. However, if the CLR leaves the group, the new CLR may have a significantly higher calculated rate. We cannot afford to increase directly to this rate, as the loss rate currently measured may not be a predictor of the loss rate at the new transmission rate. Instead we then impose a rate increase limit of one packet per RTT, which is the same as TCP's additive increase constant, so that the rate gradually increases to the new CLR's rate.

2.3 Measuring the Loss Event Rate

The loss event rate can only be scalably measured at the receivers. The measurement mechanism closely matches that used for TFRC. A receiver aggregates the packet losses into *loss events*, defined as one or more packets lost during a round-trip time. The number of packets between consecutive loss events is called a *loss interval*. The average loss interval size can be computed as the weighted average of the m most recent loss intervals $l_k, \ldots, l_{k-m+1}$:

$$l_{avg}(k) = \frac{\sum_{i=0}^{m-1} w_i l_{k-i}}{\sum_{i=0}^{m-1} w_i}$$

The weights w_i are chosen so that very recent loss intervals receive the same high weights, while the weights gradually decrease to 0 for older loss intervals. For example, with eight weights we might use $\{5, 5, 5, 5, 4, 3, 2, 1\}$. This allows for smooth changes in l_{avg} as loss events age. While large values for m improve the smoothness of the estimate, a very long loss history also reduces the responsiveness and thus the fairness of the protocol. Values around 8 to 32 appear to be a good compromise.

The loss event rate p used as an input for the TCP model is defined as the inverse of l_{avg}. The interval since the most recent loss event does not end with a loss event and thus may not reflect the loss event rate. This interval is included in the calculation of the loss event rate if doing so reduces p:

$$p = \frac{1}{\max\left(l_{avg}(k), l_{avg}(k-1)\right)}$$

For a more thorough discussion of this loss measurement mechanism see [5].

2.4 Round-trip Time Measurements

A key challenge of TFMCC is for each receiver to be able to measure its RTT to the sender without causing excessive traffic at the sender. In practice the problem is primarily one of getting an initial RTT measurement as, with the use of timestamps in the data packets, a receiver can see changes in the delay of the forward path simply from the packet's arrival time. We will discuss this further in Section 2.4.3.

2.4.1 RTT Estimate Initialization

Ideally we would like a receiver to be able to initialize its RTT measurement without having to exchange any feedback packets with the sender. This is possible if the sender and receiver have synchronized clocks, which might be achieved using GPS receivers. Less accurately, it can also be done using clocks synchronized with NTP [13].

In either case, the data packets are timestamped by the sender, and the receiver can then compute the one-way delay. The RTT is estimated to be twice the one-way delay $d_{S\to R}$. In the case of NTP, the errors that accumulate between the stratum-1 server and the local host must be taken into account. An NTP server knows the RTT and dispersion to the stratum-1 server to which it is synchronized. The sum of these gives the worst-case error ϵ in synchronization. To be conservative:

$$t_{RTT} = 2(d_{S\to R} + \epsilon_{sender} + \epsilon_{receiver})$$

In practice NTP provides an average timer accuracy of 20-30 ms [13], and in most cases this gives us an estimate of RTT that is accurate at least to the nearest 100 ms. Although not perfect, this is still useful as a first estimate.

In many cases though, no reliable form of clock synchronization is available. Each receiver must then initialize its RTT estimate to a value that should be larger than the highest RTT of any of the receivers. We assume that for most networks a value of 500 ms is appropriate [1]. This initial value is used until a real measurement can be made. In Appendix A we reason why it is safe to also use this value to aggregate losses to loss events, where a low RTT value would be the conservative option.

2.4.2 RTT Measurement

A receiver gets to measure the instantaneous RTT t_{RTT}^{inst} by sending timestamped feedback to the sender, which then echoes the timestamp and receiver ID in the header of a data packet. If more feedback messages arrive than data packets are sent, we prioritize the sender's report echoes in the following order:

1. a receiver whose report causes it to be selected as the new CLR

2. receivers that have not yet measured their RTT
3. non-CLR receivers with previous RTT measurements
4. the existing CLR.

Ties are broken in favor of the receiver with the lowest reported rate. Normally the number of data packets is larger than the number of feedback packets, so the CLR's last report is echoed in any remaining data packets.[2]

To prevent a single spurious RTT value from having an excessive effect on the sending rate we smooth the values using an exponentially weighted moving average (EWMA)

$$t_{RTT} = \beta \cdot t_{RTT}^{inst} + (1 - \beta) \cdot t_{RTT}$$

For the CLR we set $\beta_{CLR} = 0.05$. Given that other receivers will not get very frequent RTT measurements and thus old measurements are likely to be outdated, a higher value of $\beta_{non-CLR} = 0.5$ is used for them.

2.4.3 One-way Delay RTT Adjustments

Due to the infrequent RTT measurements, it would also be possible for large increases in RTT to go unnoticed if the receiver is not the CLR. To avoid this we adjust the RTT estimate between actual measurements. Since data packets carry a send timestamp t_{data}, a receiver that gets a RTT measurement at time t_{now} can also compute the one-way delay from sender to receiver (including clock skew) as

$$d_{S \to R} = t_{now} - t_{data}$$

and the one-way from receiver to sender as

$$d_{R \to S} = t_{RTT}^{inst} - d_{S \to R}$$

Due to clock skew, these values are not directly meaningful, but $d_{R \to S}$ can be used to modify the RTT estimate between real RTT measurements. When in a later data packet the one-way delay from sender to receiver is determined as $d'_{S \to R}$, it is possible to compute an up-to-date RTT estimate

$$t_{RTT}^{inst\ \prime} = d_{R \to S} + d'_{S \to R}$$

Clock skew between sender and receiver cancels out, provided that clock drift between real RTT measurements is negligible. The modified RTT estimates are smoothed with an EWMA just like normal RTT measurements, albeit with a smaller decay factor for the EWMA since the one-way delay adjustments are possible with each new data packet. One-way delay adjustments are used as an indicator that the RTT may have changed significantly and thus a real RTT measurement is necessary. If the receiver is then selected as CLR, it measures its RTT with the next packet and all interim one-way delay adjustments are discarded. For this reason it proved to be unnecessary to filter out flawed one-way delay estimates.

[2] To be able to infer an accurate RTT from the timestamps it is necessary to also take into account the offset between receipt of a timestamp and echoing it back.

2.4.4 Sender-side RTT Measurements

While a preconfigured initial RTT value can be used at the receiver for loss aggregation and rate computation, it should not be used to set the sending rate. Using a high initial RTT would result in a very low sending rate, followed by a high sending rate when the CLR gets the first RTT measurement, then a CLR change to a receiver with no previous RTT measurement, and so on. Such rate oscillations should be avoided. On the other hand, if the sender only accepted a receiver with a valid RTT as CLR, receivers with a very high loss rate might never receive their feedback echo, and so never become CLR.

For these reasons, TFMCC supports additional sender-based RTT measurements. A receiver report also echoes the timestamp of the last data packet, and so the sender and receivers are both able to measure RTT. The sender *only* computes the RTT when it has to react to a receiver report without a valid RTT, and it uses this to adjust the calculated rate in the receiver report.

2.5 Receiver Feedback

As TFMCC is designed to be used with receiver sets of perhaps several thousand receivers, it is critical to ensure that the sender gets feedback from the receivers experiencing the worst network conditions without being overwhelmed by feedback from all the other receivers. Congestion may occur at any point in the distribution tree, from the sender's access link through to a single receiver's tail circuit. Thus any mechanism must be able to cope when conditions change from a single receiver being lightly congested to all the receivers being equally heavily congested, and other similarly pathological cases. At the same time we would like the feedback delay to be relatively small in the steady state. The latter can be achieved through the concept of a CLR, which can send feedback immediately.

However, a CLR is of no help during a change in network conditions that affect receivers other than the CLR. Thus, we will ignore the influence of the CLR on the feedback process in this section, but we note that the CLR generates relatively little feedback traffic and both strictly improves the responsiveness to congestion and reduces the amount of feedback sent by other receivers.

Various reliable multicast protocols incorporate feedback trees, where the receivers are organized into a tree hierarchy, and internal nodes in the tree aggregate feedback. Such trees largely solve the feedback implosion problem, but are difficult to build and maintain. If such a tree exists it should clearly be used, but in this paper we will assume that is not the case, and examine pure end-to-end suppression mechanisms.

Several mechanisms using randomized timers for feedback suppression in multicast protocols have been proposed before

[6, 7, 9, 14]. Time is divided into feedback rounds, which are either implicitly or explicitly indicated to the receivers. At the start of each feedback round, each receiver sets a randomized timer. If the receiver hears feedback from another receiver that makes it unnecessary for it to send its own feedback, it cancels its timer. Otherwise when the timer expires, a feedback message is sent.

For TFMCC, we use such a mechanism based on exponentially distributed random timers. When the feedback timer expires, the receiver unicasts its current calculated sending rate to the sender. If this rate is lower than previous feedback received, the sender echoes the feedback to all receivers. With respect to the intended application of finding the correct CLR, we improve upon the original concept by biasing feedback in favor of low-rate receivers. The dynamics of such a mechanism depend both on the way that the timers are initialized, and on how one receiver's feedback suppresses another's.

2.5.1 Randomized Timer Values

The basic exponentially distributed random timer mechanism initializes a feedback timer to expire after t seconds, with

$$t = \max\left(T(1 + \log_N x), 0\right) \tag{2}$$

where

x is a uniformly distributed random variable in $(0, 1]$,
T is an upper limit on the delay before sending feedback,
N is an estimated upper bound on the number of receivers.

T is set to a multiple of the maximum RTT of the receivers; $T = b\, t_{RTT}^{max}$. The choice of b determines the number of feedback packets per round that will be sent in worst-case conditions and the feedback delay under normal conditions. In Section 2.5.4 we show that for our purpose useful values for b lie between 3 and 6. We use a default value of 4.

The mechanism is relatively insensitive to overestimation of the receiver set size N, but underestimation may result in a feedback implosion. Thus, a sufficiently large value for N should be chosen. In our simulations we use $N = 10,000$, which seems reasonable given our scaling goals.

Whilst this basic algorithm is sufficient to prevent a feedback implosion, it does not ensure that receivers with low expected rates will be more likely to respond than receivers with high rates. Even if a receiver can only respond when its rate is less than the current sending rate, this does not ensure that the lowest-rate receiver will respond quickly when congestion worsens rapidly.[3] Thus the sender would be insufficiently responsive to increased congestion.

To avoid this problem, we bias the feedback timers in favor of receivers with lower rates, while still allowing sufficient randomization to avoid implosion when all the receivers calculate the same low rate. Since a receiver knows the sending rate but not the calculated rate of other receivers, a good measure of the importance of its feedback is the ratio r of the calculated rate to the current sending rate.[4] There are several ways to use r to bias the timers:

- **Modify N**: reduce the upper bound on the receiver set.
- **Offset**: subtract an offset value from the feedback time.
- **Modify x**: reduce the random value x.

All three alternatives cause low-rate receivers to report earlier but they differ with respect to the degree of biasing they cause and the circumstances under which a feedback implosion might be possible.

When modifying N, its value should never be reduced to less than the actual number of receivers n, since receivers send an immediate response with a probability of $1/N$. In case $n < N$, the number of feedback responses increases linearly in relation to n. If N is known to be too large and it is thus possible to safely reduce N, it makes sense to always use the reduced N for the feedback suppression instead of using it for the biasing.

Using an offset decreases the time for *all* congested receivers to respond, but the probability of a very short timer value is not greatly increased and so suppression still works.

$$t' = \gamma r T + (1 - \gamma)T \cdot (1 + \log_N x) \tag{3}$$

γ determines the fraction of T that should be used to spread out the feedback responses with respect to the reported rate. Care has to be taken to ensure that $(1 - \gamma)T$ is sufficiently large to prevent a feedback implosion.

With $T(1 + \log_N rx) = T(1 + \log_N x) + T \log_N r$, the third case is similar to the second case with a different offset value. Also here it is important to bound the impact of r on the feedback time.

Figure 1 shows how the cumulative distribution function (CDF) of the feedback time changes from the original CDF when biasing the feedback. A decrease in N corresponds to shifting the CDF up, thus increasing the probability of early responses that cannot be suppressed. In contrast, using a fraction of T as an offset reduces the time over which the responses are spread out, assuming the worst case of all receivers reporting an optimal value.

Thus, in TFMCC the feedback timers are biased in favor of low-rate receivers with an offset as in Equation 3. To clarify how this method affects the feedback time, the time-value distribution of the receiver set without biasing and with timers biased with an offset is depicted in Figure 2. Suppressed Feedback is marked with a dot, feedback received at the sender

[3] In fact, receivers with lower RTTs are incorrectly favored since they receive the feedback request earlier.

[4] Note that $0 < r < 1$ since only receivers with lower rates than the current rate send reports.

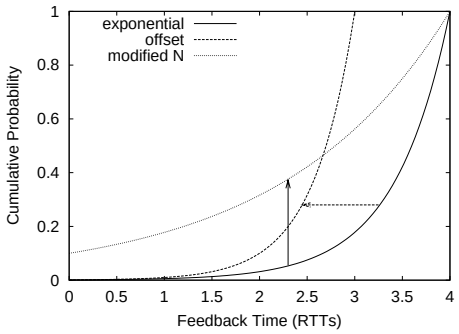

Figure 1: Different feedback biasing methods

is marked with a cross and the best value of the feedback received is marked with a square. Note that a uniform distribution of the feedback value r as was used for the graph is unlikely to occur in reality and is used here only for the purpose of demonstrating the properties of feedback biasing.

With the offset method, the time interval available for suppression is smaller than with unbiased feedback if the original worst case delay is to be maintained. As a consequence, the number of feedback messages is higher when biasing the feedback timers. However, through the biasing, early feedback messages and thus also the best feedback value received are closer to optimal.

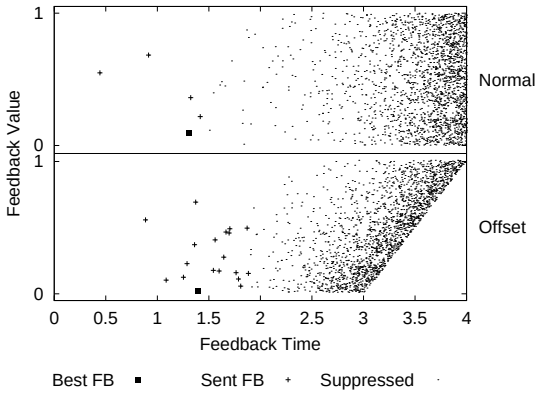

Figure 2: Time-value distribution

We can further optimize the offset method by truncating the range of r to likely values, and normalizing the resulting interval to [0,1]. In the implementation, instead of r, we use

$$r' = (\max(\min(r, 0.9), 0.5) - 0.5)/0.4)$$

The effect of this is to start biasing feedback only when a receiver's rate is less than 90% of the sender's rate (this doesn't significantly affect fairness), and to saturate the bias if the receiver's rate is 50% of the sender's rate (since receivers with even lower rates will take several rounds for their loss measures to change anyway).

2.5.2 Canceling Feedback

When a receiver sees echoed feedback from another receiver, it must decide whether or not to cancel its feedback timer. One possibility is to rely completely on the feedback timer bias, and cancel the timer on receipt of the first feedback for this round. Another possibility is to cancel the timer only if the echoed feedback indicates a rate lower than the rate the receiver wanted to report. The latter guarantees that the receiver with the lowest rate will always get to send its feedback, but the former results in significantly less feedback traffic in the worst case.

A spectrum lies between these two extremes: if the receiver's calculated rate is R_{calc} and the rate from the echoed feedback is R_{fb}, then the timer is canceled if $R_{fb} - R_{calc} < \theta\ R_{fb}$. The former method discussed above corresponds to $\theta = 1$ and the latter to $\theta = 0$. As we change θ from zero to one, we reduce the chance of hearing from the absolute lowest-rate receiver, but also reduce the increase in the number of feedback messages. As shown in [19], the expected number of feedback messages increases logarithmically with n for $\theta = 1$. For values of $\theta < 1$, this number becomes approximately constant in the limit for large n.

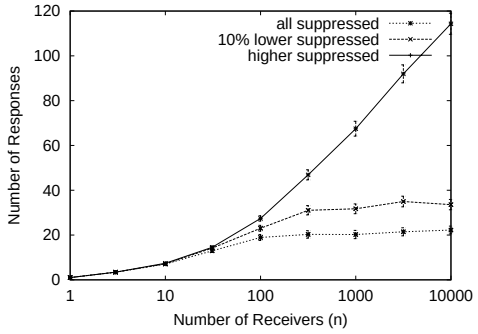

Figure 3: Different feedback cancellation methods

These results are corroborated by the simulations depicted in Figure 3. The graph shows the number of feedback messages in the first round of the worst-case scenario, where n receivers (except the CLR) suddenly experience congestion. The effects of θ being 0.0, 0.1, and 1.0 are shown. Values of θ around 0.1 result in the desired behavior of only a marginally higher number of feedback messages, while the resulting transient transmission rate is no worse than 10% higher than it should be.

The improvement in sent feedback values caused by the biasing in combination with the above feedback cancellation method results in a significant improvement of the characteristics of the feedback process over normal exponential feedback timers.

2.5.3 Feedback at Low Sending Rates

At very low sending rates and high loss rates (which usually go together), it is still possible to get a feedback implosion. The feedback echo from the sender that suppresses other feedback is sent with the next data packet. Thus, when the delay before the next data packet is sent is close to the feedback delay, it will arrive too late for suppression to work.

This problem can be prevented by increasing the feedback delay T in proportion to the time interval between data packets when the sending rate R_{send} is low:

$$T = b \max\left(t_{RTT}^{max}, (c+1)\frac{s}{R_{send}}\right)$$

c being the number of consecutive data packets that can be lost without running the risk of implosion, and s the packet size. We recommend using values of c between 2 and 4.

2.5.4 Expected Number of Feedback Messages, Feedback Delay, and Feedback Quality

The expected number of duplicate feedback messages $E[f]$ for exponential feedback suppression is given in [7] as

$$E[f] = N^{\tau/T'}\left(\frac{n}{N} + \left(1-\frac{1}{N}\right)^n - \left(1-\frac{1}{N^{\tau/T'}}\right)^n\right)$$

where

n is the actual number of receivers,

τ is the network delay (for unicast feedback channels $\tau = t_{RTT}^{max}$),

T' is the maximum feedback delay used for suppression.

Assuming the worst case of $r = 0$ for all receivers, $T' = (1-\gamma)T$.

Whilst our primary concern is to avoid implosion, a very low number of responses (say 1 or 2) is also undesirable. Some additional responses greatly increase the probability of not having a *low*-rate but the *lowest*-rate receiver respond and also provide RTT measurements to a larger number of receivers.

Figure 4 shows a plot of $E[f]$ for different values of T' and n, with $N = 10,000$. Values of T' in the range of roughly 3 to 4 RTTs result in the desired number of feedback messages, particularly in the common range for n of one to two orders of magnitude below N. For this reason, the values chosen for γ and T in the TFMCC implementation are $1/4$ and $4\,t_{RTT}^{max}$ respectively. Given those choices for γ and T, we now examine how well the feedback biasing methods achieve the additional goal of low response time and how close the reported rate is to that of the true lowest-rate receiver.

Figure 5 compares the feedback delay for unbiased exponential timers with the basic offset bias and the modified offset

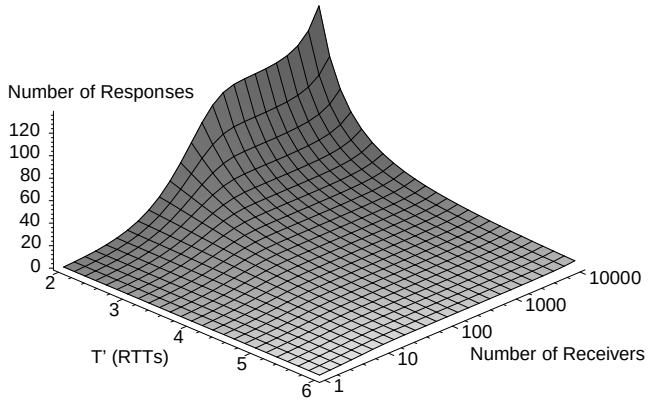

Figure 4: Expected number of feedback messages

that uses r' instead of r. All three show the logarithmic decrease in response time with the number of receivers typical for feedback suppression based on exponential timers. The difference between the methods is not great, with the modified offset algorithm having a slight edge over the regular offset.

When examining the rates that are reported in the feedback messages, the advantage of the offset methods becomes apparent. Figure 6 compares the lowest reported rate of the feedback messages of a single feedback round to the actual lowest rate of the receiver set. For example, a value of 0.1 indicates that the lowest reported rate is on average 10% higher after one feedback round than it should be in the ideal case. Rates reported with the offset methods are considerably closer to the real minimum than those reported with unmodified exponential timers. Particularly when r is adjusted appropriately by the modified offset method, feedback will normally be within a few percent of the minimum rate. Plain exponential feedback shows average deviations of nearly 20% above the minimum rate.

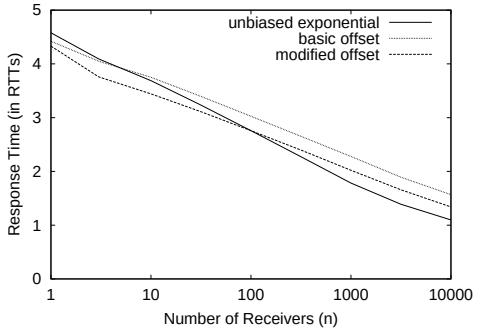

Figure 5: Comparison of methods to bias feedback

2.6 Slowstart

TFMCC uses a slowstart mechanism to more quickly approach its fair bandwidth share at the start of a session. During

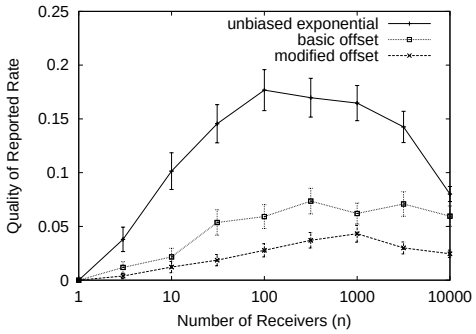

Figure 6: Comparison of methods to bias feedback

slowstart, the sending rate increases exponentially, whereas normal congestion control allows only a linear increase. An exponential increase can easily lead to heavy congestion, so great care has to be taken to design a safe increase mechanism. A simple measure to this end is to limit the increase to a multiple d of the minimum rate R_{recv}^{min} received by any of the receivers. Since a receiver can never receive at a rate higher than its link bandwidth, this effectively limits the overshoot to d times that bandwidth. The target sending rate is calculated as

$$R_{send}^{target} = d\, R_{recv}^{min}$$

and the current sending rate is gradually adjusted to the target rate over the course of a RTT. In our implementation we use a value of $d = 2$. Slowstart is terminated as soon as any one of the receivers experiences its first packet loss.

It is necessary to use a different feedback bias for slowstart since receivers cannot calculate a TCP-friendly rate. For this reason we use:

$$r = R_{recv}/R_{send}$$

A report from the receiver that experiences the first loss event can only be suppressed by other reports also indicating packet loss, but not by reports from receivers that did not yet experience loss. Thus, slowstart will be terminated no later than one feedback delay after the loss was detected.

In practice, TFMCC will seldomly reach the theoretical maximum of a doubling of the sending rate per RTT for two reasons:

- The target sending rate is increased only when feedback from a new feedback round is received. Thus, doubling is not possible every RTT, but every feedback delay, which is usually much larger than a RTT.
- Measuring the receive rate over several RTTs and gradually increasing R_{send} to R_{send}^{target} gives a minimum receive rate at the end of a feedback interval that is lower than the sending rate during that interval. Thus, setting R_{send}^{target} to twice the minimum receive rate does not double the current sending rate.

As is desirable for a multicast protocol, TFMCC slowstart behaves more conservatively than comparable unicast slowstart mechanisms.

3 Protocol Behavior with Very Large Receiver Sets

The loss path multiplicity problem is a well-known characteristic of multicast congestion control mechanisms that react to single loss indications from receivers on different network paths. It prevents the scaling of those mechanisms to large receiver sets. In [3], the authors propose as a possible solution tracking the most congested path and taking only loss indications from that path into account. Since the reports of a TFMCC receiver contain the expected rate based on the loss event rate and RTT on the single path from sender to that receiver, the protocol implicitly avoids the loss path multiplicity problem. Yet TFMCC (and all other single-rate congestion control schemes) may be confined to a rate below the fair rate if, rather than there being a single most congested path, there is a path that changes over time. The faster a multicast congestion control protocol responds to transient congestion, the more pronounced is the effect of tracking the minimum of stochastic variations in the calculated rate at the different receivers. For example, if loss to several receivers independently varies fairly quickly between 0% and 10% with the average being 5%, a congestion control protocol may always track the worst receiver, giving a loss estimate that is twice what it should be.

A worst-case scenario in this respect is a high number of receivers with independent loss and a calculated rate in the range of the lowest-rate receiver. If n receivers experience independent packet loss with the same loss probability, the loss intervals will have an exponential distribution. The expected value of the minimum of n exponentially distributed random variables is proportional to $1/n$. Thus, if TFMCC based its rate calculations on a single loss interval, the average sending rate would scale proportionally to $1/\sqrt{n}$ (in the case of moderate loss rates, otherwise even worse). The rate calculation in TFMCC is based on a weighted average of m loss intervals. Since the average of exponentially distributed random variables is gamma distributed, the expected loss rate in TFMCC is inversely proportional to the expected value for the minimum of n gamma distributed random variables. [5]

This effect is shown in Figure 7 for different numbers of receivers n with a constant loss probability. For uncorrelated loss at a rate of 10% and a RTT of 50 ms, the fair rate for the TFMCC transmission is around 300 KBit/s. This sending rate is reached when the receiver set consists of only a single

[5] For first order statistics of the gamma distribution, no simple closed form expressions exists. Details about the distribution of the minimum of gamma distributed random variables can be found in [8].

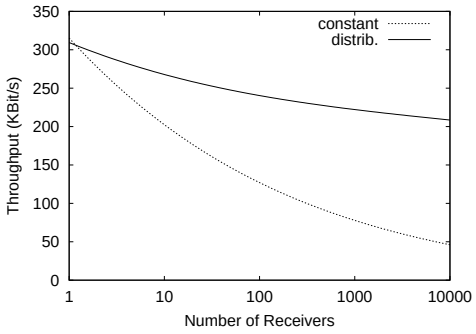

Figure 7: Scaling

receiver but it quickly drops to a value of only a fraction of the fair rate for larger n. For example, for 10,000 receivers, only 1/6 of the fair rate is achieved.

Fortunately, such a loss distribution is extremely unlikely in real networks. Multicast data is transmitted along the paths of the distribution tree of the underlying multicast routing protocol. A lossy link high up in the tree may affect a large number of receivers but the losses are correlated and so the above effect does not occur. When some of those receivers have additional lossy links, the loss rates are no longer correlated, rather the values are spread out over a larger interval, thus decreasing the number of receivers with similar loss rates. To demonstrate this effect, we choose a distribution of loss rates that is closer to actual loss distributions in multicast trees in that there are only a limited number of high loss receivers while the majority of receivers will have moderate loss rates.[6] Here, a small number of receivers (proportional to $a\log(n)$, where a is a constant) is in the high loss range of 5-10%, some more are in the range of 2%-5%, and the vast majority have loss rates between 0.5% and 2%. Under such network conditions the throughput degradation with 10,000 receivers is merely 30%. Thus, the throughput degradation plays a significant role only when the vast majority of packet loss occurs on the last hop to the receivers and those losses amount to the same loss rates.

It is impossible to distinguish between a "stochastic" decrease in the sending rate and a "real" decrease caused by an increased congestion level (otherwise it would be possible to estimate the effect and adjust the sending rate accordingly). The degradation effect can be alleviated by increasing the number of loss intervals used for the loss history, albeit at the expense of less responsiveness.

[6] By no means do we claim that the chosen distribution exactly reflects network conditions in multicast distribution trees.

4 Protocol Simulations

We implemented TFMCC in the *ns*2 network simulator [2] to investigate its behavior under controlled conditions. In this paper, we can only report a small fraction of the simulations that were carried out. In all simulations below, drop-tail queues were used at the routers to ensure acceptable behavior in the current Internet. Generally, both fairness towards TCP and intra-protocol fairness improve when active queuing (e.g. RED) is used instead.

4.1 Fairness

Fairness towards competing TCP flows was analyzed using the well-known single-bottleneck topology (Figure 8) where a number of sending nodes are connected to as many receiving nodes through a common bottleneck. Figure 9 shows the

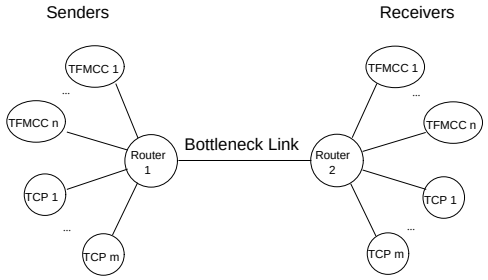

Figure 8: Topology

throughput of a TFMCC flow and two sample TCP flows (out of 15) from a typical example of such simulations. The average throughput of TFMCC closely matches the average TCP throughput but TFMCC achieves a smoother rate. Similar results can be obtained for many other combinations of flows. In general, the higher the level of statistical multiplexing, the better the fairness among competing flows. Only in scenarios where the number of TFMCC flows greatly exceeds the number of TCP flows is TFMCC more aggressive than TCP. The reason for this lies in the spacing of the data packets and buffer requirements: TFMCC spaces out data packets, while TCP sends them back-to-back if it can send multiple packets, making TCP more sensitive to nearly-full queues typical of drop-tail queue management.

If instead of one bottleneck the topology has separate bottlenecks on the last hops to the receivers, then we observe the throughput degradation predicted in Section 3. When the scenario above is modified such that TFMCC competes with single TCP flows on sixteen identical 1 MBit/s tail circuits, then TFMCC achieves only 70% of TCP's throughput (see Figure 10).

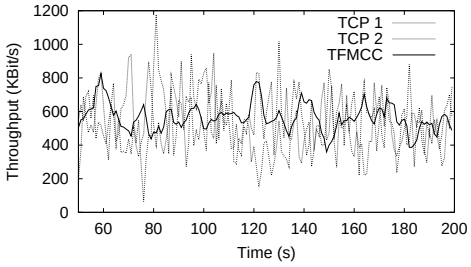

Figure 9: One TFMCC flow and 15 TCP flows over a single 8 MBit/s bottleneck

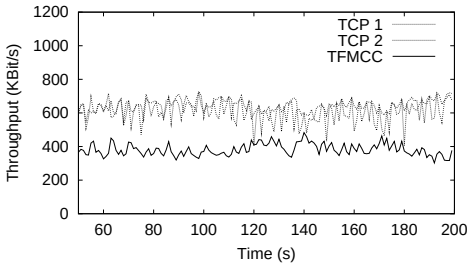

Figure 10: 1 TFMCC flow and 16 TCP flows (individual bottlenecks)

4.2 Responsiveness to Changes in the Loss Rate

An important concern in the design of congestion control protocols is their responsiveness to changes in network conditions. Furthermore, when receivers join and leave the session it is important that TFMCC react sufficiently fast should a change of CLR be required. This behavior is investigated using a star topology with four links having a RTT of 60 ms and loss rates of 0.1%, 0.5%, 2.5%, and 12.5% respectively. At the beginning of the simulation the receiver set consists only of the receiver with the lowest loss rate. Other receivers join the session after 100 seconds at 50 second intervals in the order of their loss rates (lower-loss-rate receivers join first). After 250 seconds, receivers leave the transmission in reverse order, again with 50 second intervals in between. To verify that TFMCC throughput is similar to TCP throughput, an additional TCP connection to each receiver is set up for the duration of the whole experiment.

As show in Figure 11, TFMCC matches closely the TCP throughput at all four loss levels. Adaption of the sending rate when a new higher-loss receiver joins is fast. The receiver needs 500-1000 ms after the join to get enough packets to compute a meaningful loss rate. The major part of the delay is caused by the exponential timer for the feedback suppression, which increases the overall delay before a new CLR is chosen to roughly one to three seconds.[7] The experiment

[7]Note that this high delay is caused by the use of the initial RTT in the feedback suppression mechanism. Once all receivers have a valid RTT estimate, the delay caused by feedback suppression is much shorter.

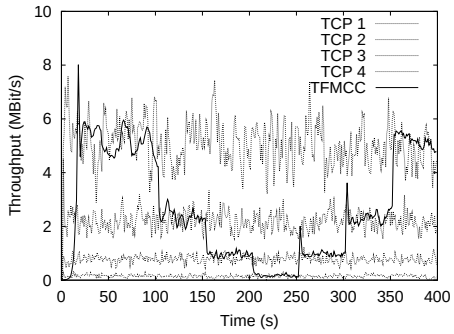

Figure 11: Responsiveness to changes in the loss rate

demonstrates TFMCC's very good reactivity to changes in congestion level.

The delay before TFMCC assumes that a rate-limiting receiver left the group and the sending rate can be increased is configurable. Currently, an absence of feedback from the CLR for 10 $\times$ the feedback delay is used as an indication that this receiver left the group. In case explicit leave messages are used with the TFMCC protocol the delay can be reduced to one RTT.

The same simulation setting can be used to investigate responsiveness to changes in the RTT. The results (not shown here) are similar to those above, since all four receivers have measured their RTT by the time the RTT changes, and the one-way RTT adjustments immediately indicate this change.

With larger receiver sets, the amount of time that expires until a high RTT receiver is found may be greater. This effect is investigated in the next section.

4.3 Initial RTT Measurements and Responsiveness to Changes in the RTT

The number of receivers that measure their RTT each feedback round depends on the number of feedback messages and thus on the parameters used for feedback suppression. Figure 12 shows how the number of receivers with a valid RTT estimate evolves over time for a large receiver set and a high initial RTT value. The link RTTs for the 1000 receivers vary between 60 ms and 140 ms and the initial RTT value is set to 500 ms. A single bottleneck is used to produce highly correlated loss for all receivers. This is the worst case, since if loss estimates at the receivers vary, it is often unnecessary to measure the RTT to the low-loss receivers. Since the calculated rate of the receivers still using the initial RTT is below the current sending rate, at least one receiver will get its first RTT measurement per feedback round until all receivers have measured their RTT.

At the beginning of the simulation, the number of receivers obtaining initial RTT measurements is close to the expected

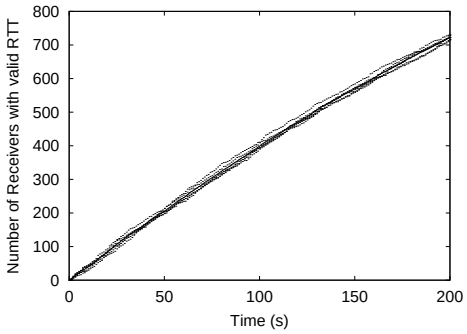

Figure 12: Rate of initial RTT measurements

number of feedback messages per feedback round. Over time, as more and more receivers have a valid RTT, the number of receivers that want to give feedback decreases, and the rate of initial RTT measurements gradually drops to one new measurement per feedback round. While a delay of 200 seconds until 700 of the 1000 receivers have measured their RTT seems rather large, one should keep in mind that this results from having the same congestion level for all receivers. If some receivers experience higher loss rates, those receivers will measure their RTT first and TFMCC can adapt to their calculated rate. Under most real network conditions it will not be necessary to measure the RTT to all receivers.

In scenarios with 40, 200 and 1000 receivers respectively, we investigate how long it takes until a high RTT receiver is found among receivers with a low RTT when all receiver experience independent loss with the same loss probability. The x-axis of the graph in Figure 13 denotes the point of time when the RTT is increased during the experiment and the y-axis shows the amount of time after which this change in RTT is reacted upon by choosing the correct CLR. The later the increase in RTT, the greater the number of receivers already having valid RTT estimates, and the expected time until the high-RTT receiver is selected as CLR decreases.

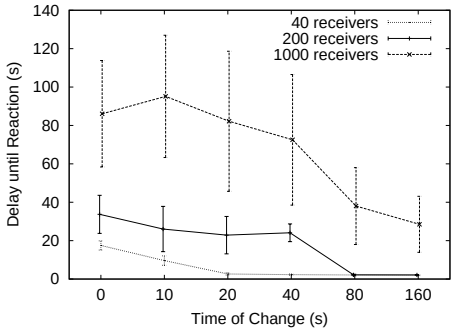

Figure 13: Responsiveness to changes in the RTT

4.4 Slowstart

The highest sending rate achieved during slowstart is largely determined by the level of statistical multiplexing. On an otherwise empty link, TFMCC will reach roughly twice the bottleneck bandwidth before leaving slowstart, as depicted in Figure 14. When TFMCC competes with a single TCP flow, slowstart is terminated at a rate below the fair rate[8] of the TFMCC flow and this rate is relatively independent of the number of TFMCC receivers. Already in the case of two competing TCP flows, and even more so when the level of statistical multiplexing is higher, the slowstart rate decreases considerably when the number of receivers increases. Most of the increase to the fair rate takes place after slowstart in normal congestion control mode.

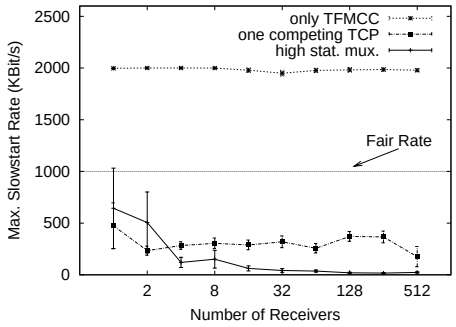

Figure 14: Maximum slowstart rate

We do not include an extra graph of the exact increase behavior of TFMCC compared to TCP, since it can be seen for example in Figures 15 and 16. TFMCC and TCP are started at the same time. TCP's increase to the fair rate is very rapid, while it takes TFMCC roughly 20 seconds to reach that level of bandwidth.

4.5 Late-join of Low-rate Receiver

In the previous experiments we investigated congestion control with moderate loss rates, expected to be prevalent in the applications domains for which TFMCC is well suited. Under some circumstances, the loss rate at a receiver can initially be much higher. Consider an example where TFMCC operates at a fair rate of several MBit/s and a receiver with a low-bandwidth connection joins. Immediately after joining, this receiver may experience loss rates close to 100%. While such conditions are difficult to avoid, TFMCC should ensure that they exist only for a limited amount of time and quickly choose the new receiver as CLR.

The initial setup for this simulation is a eight-member TFMCC session competing with seven TCP connections on a 8 MBit/s link, giving a fair rate of 1 MBit/s. During the simulation, a

[8] The fair rate for TFMCC in all three simulations is 1 MBit/s.

new receiver joins the session behind a separate 200 KBit/s bottleneck from the sender from time 50 to 100 seconds.

TFMCC does not have any problems coping with this scenario, choosing the joining receiver as CLR within a very few seconds. Although the loss rate for the joining receiver is initially very high, the TFMCC rate does not drop to zero. As soon as the buffer of the 200 KBit/s connection is full, the receiver experiences the first loss event and the loss history is initialized. Details about the loss history initialization process can be found in Appendix B. When the first loss occurs, the receiver gets data at a rate of exactly the bottleneck bandwidth. Thus, the loss rate will be initialized to a value below the 80% value and from there adapt to the appropriate loss event rate such that the available bandwidth of 200 KBit/s is used.

When an additional TCP flow is set up using the 200 KBit/s link for the duration of the experiment, this flow inevitably experiences a timeout when the new receiver joins the multicast group and the link is flooded with packets. However, shortly afterwards, TFMCC adapts to the available capacity and TCP recovers with bandwidth shared fairly between TFMCC and TCP.

We conclude that TFMCC shows good performance and fairness, even under unfavorable network conditions.

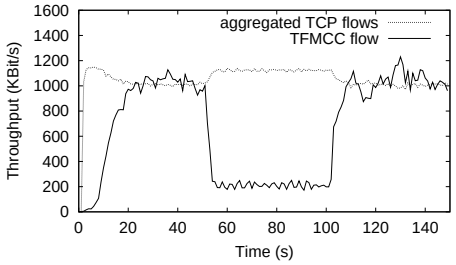

Figure 15: Late-join of low-rate receiver

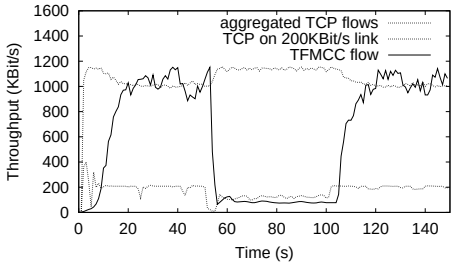

Figure 16: Additional TCP flow on the slow link

5 Related Work

To date, a number of single-rate multicast congestion control schemes have been proposed. A prominent recent example is PGMCC [17]. It selects the receiver with the worst network conditions as a group representative, called the *acker*. The selection process for the acker mainly determines the fairness of the protocol, and is based on a simplified version of the TCP throughput model in Equation (4). Similar to TFMCC, each receiver tracks the RTT and the smoothed loss rate, and feeds these values into the model. The results are communicated to the sender using normal randomized feedback timers to avoid an implosion. If available, PGMCC also makes use of network elements to aggregate feedback.

Once an acker is selected, a TCP-style window-based congestion control algorithm is run between the sender and the acker. Minor modifications compared to TCP concern the separation of congestion control and reliability to be able to use PGMCC for reliable as well as unreliable data transport and the handling of out of order packets and RTT changes when a different receiver is selected as the acker.

As evidenced by the simulations in [17], PGMCC competes fairly with TCP for many different network conditions. The basic congestion control mechanism is simple and its dynamics are well understood from the analysis of TCP congestion control. This close mimicking of TCP's window behavior produces rate variations that resemble TCP's sawtooth-like rate. This makes PGMCC suited for applications that can cope with larger variations in the sending rate. In contrast, the rate produced by TFMCC is generally smoother and more predictable, making TFMCC well suited to applications with more constraints on acceptable rate changes. Since the acker selection process is critical for PGMCC's performance, PGMCC might benefit from using a feedback mechanism similar to that of TFMCC, based on biased exponentially weighted timers. To summarize, we believe that both PGMCC and TFMCC present viable solutions for single-rate multicast congestion control, targeted at somewhat different application domains.

While PGMCC relies on a congestion window, TCP-Emulation at Receivers (TEAR) [16] is a combination of window- and rate-based congestion control. It features a TCP-like window emulation algorithm at the receivers, but the window is not used to directly control transmission. Instead, the average window size is calculated and transformed into a smoothed sending rate, which is used by the sender to space out data packets. So far, only a unicast version of TEAR exists, but the mechanism can be made multicast-capable by implementing a TFMCC-like scalable feedback suppression scheme to communicate the calculated rate to the sender as well as scalable RTT measurements. The advantage of TEAR lies in the fact that it does not require a model of TCP with all the necessary assumptions to compute a rate. However, for low levels of statistical multiplexing, TEAR's emulation assumptions about the independence of loss timing from transmit rate and of timeout emulation mean that it shares many of the limitations of the TCP models we use. Thus we do not expect a multicast variant of TEAR to behave significantly better or worse than TFMCC.

6 Conclusions

We have described TFMCC, a single-rate multicast congestion control mechanism intended to scale to groups of several thousand receivers. Performing multicast congestion control whilst remaining TCP-friendly is difficult, in particular because TCP's transmission rate depends on the RTT, and measuring RTT in a scalable manner is a hard problem. Given the limitations of end-to-end protocols, we believe that TFMCC represents a significant improvement over previous work in this area.

We have extensively evaluated TFMCC through analysis and simulation, and believe we have a good understanding of its behavior in a wide range of network conditions. To summarize, we believe that under the sort of conditions TFMCC will experience in the real-world it will behave rather well. However we have also examined certain pathological cases; in these cases the failure mode is for TFMCC to achieve a slower than desired transmission rate. Given that all protocols have bounds to their good behavior, this is the failure mode we would desire, as it ensures the safety of the Internet.

An important part of any research is to identify the limitations of a new design. TFMCC's main weakness is in the startup phase – it can take a long time for sufficiently many receivers to measure their RTT (assuming we cannot use NTP to provide approximate default values). In addition, with large receiver sets, TCP-style slowstart is not really an appropriate mechanism, and a linear increase can take some time to reach the correct operating point. However these weaknesses are not specific to TFMCC – any *safe* single-rate multicast congestion control mechanism will have these same limitations if it is TCP-compatible. The implication is therefore that single-rate multicast congestion control mechanisms like TFMCC are only really well-suited to relatively long-lived data streams. Fortunately it also appears that most current multicast applications such as stock-price tickers or video streaming involve just such long-lived data-streams.

6.1 Future Work

We plan to pursue this work further on several fronts. While large-scale multicast experiments are hard to perform in the real world, we plan to deploy TFMCC in a multicast filesystem synchronization application (e.g. rdist) to gain small-scale experience with a real application.

Some reliable multicast protocols build an application-level tree for acknowledgment aggregation. We have devised a hybrid rate/window-based variant of TFMCC that uses implicit RTT measurement combined with suppression within the aggregation nodes. This variant does not need to perform explicit RTT measurements or end-to-end feedback suppression. Whilst at first glance this would seem to be a big improvement over the variant in this paper, in truth it moves the complex initialization problem from RTT measurement to scalable ack-tree construction, which shares many of the problems posed by RTT measurement. Still, this seems to be a promising additional line of research.

Finally, the basic equation-based rate controller in TFMCC would also appear to be suitable for use in receiver-driven layered multicast, especially if combined with dynamic layering [4] to eliminate problems with unpredictable multicast leave latency.

7 Acknowledgements

We would like to thank Sally Floyd and Luigi Rizzo for their invaluable comments. We would also like to acknowledge feedback and suggestions received from RMRG members on earlier versions of TFMCC.

References

[1] M. Allman. A web server's view of the transport layer. *ACM Computer Communication Review*, 30(5), Oct. 2000.

[2] S. Bajaj, L. Breslau, D. Estrin, K. Fall, S. Floyd, P. Haldar, M. Handley, A. Helmy, J. Heidemann, P. Huang, S. Kumar, S. McCanne, R. Rejaie, P. Sharma, K. Varadhan, Y. Xu, H. Yu, and D. Zappala. Improving simulation for network research. Technical Report 99-702b, University of Southern California, March 1999. revised September 1999, to appear in IEEE Computer.

[3] S. Bhattacharyya, D. Towsley, and J. Kurose. The loss path multiplicity problem in multicast congestion control. In *Proc. of IEEE Infocom*, volume 2, pages 856 – 863, New York, USA, March 1999.

[4] J. Byers, M. Frumin, G. Horn, M. Luby, M. Mitzenmacher, A. Roetter, and W. Shaver. FLID-DL: Congestion control for layered multicast. In *Proc. Second Int'l Workshop on Networked Group Communication (NGC 2000)*, pages 71–81, Palo Alto, CA, USA, Nov. 2000.

[5] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proc. ACM SIGCOMM*, pages 43 – 56, Stockholm, Sweden, Aug. 2000.

[6] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5(6):784 – 803, Dec. 1997.

[7] T. Fuhrmann and J. Widmer. On the scaling of feedback algorithms for very large multicast groups. *Computer Communications*, 24(5-6):539 – 547, Mar. 2001.

[8] S. S. Gupta. Order statistics from the gamma distribution. *Technometrics*, 2:243 – 262, 1960.

[9] M. Handley. Session directories and scalable Internet multicast address allocation. In *Proc. ACM Sigcomm*, pages 105 – 116, Vancouver, B.C., Canada, Sept. 1998.

[10] A. Mankin, A. Romanow, S. Bradner, and V. Paxson. RFC 2357: IETF criteria for evaluating reliable multicast transport and application protocols, June 1998. Obsoletes RFC1650. Status: INFORMATIONAL.

[11] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the congestion avoidance algorithm. *Computer Communications Review*, 1997.

[12] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *Proc. of ACM SIGCOMM*, pages 117 – 130, Palo Alto, CA, USA, Aug. 1996.

[13] D. L. Mills, A. Thyagarajan, and B. C. Huffman. Internet timekeeping around the globe. *Proc. Precision Time and Time Interval (PTTI) Applications and Planning Meeting*, pages 365 – 371, Dec. 1997.

[14] J. Nonnenmacher and E. W. Biersack. Scalable feedback for large groups. *IEEE/ACM Transactions on Networking*, 7(3):375 – 386, June 1999.

[15] J. Padhye, V. Firoiu, D. F. Towsley, and J. F. Kurose. Modeling TCP Reno performance: a simple model and its empirical validation. *IEEE/ACM Transactions on Networking*, 8(2):133–145, April 2000.

[16] I. Rhee, V. Ozdemir, and Y. Yi. TEAR: TCP emulation at receivers - flow control for multimedia streaming. Technical report, Department of Computer Science, North Carolina State University, Apr. 2000.

[17] L. Rizzo. pgmcc: A TCP-friendly single-rate multicast congestion control scheme. In *Proc. ACM SIGCOMM*, pages 17 – 28, Stockholm, Sweden, August 2000.

[18] L. Vicisano, J. Crowcroft, and L. Rizzo. TCP-like congestion control for layered multicast data transfer. In *Proc. of IEEE INFOCOM*, volume 3, pages 996 – 1003, March 1998.

[19] J. Widmer and T. Fuhrmann. Extremum feedback for very large multicast groups. Technical Report TR 12-2001, Praktische Informatik IV, University of Mannheim, Germany, May 2001.

[20] J. Widmer and M. Handley. Extending equation-based congestion control to multicast applications. In *Proc. ACM SIGCOMM*, San Diego, CA, Aug. 2001.

A Using the Initial RTT for the Aggregation of Loss Events

Using the initial RTT for the rate computation before a valid RTT measurement is obtained is safe since it leads to a lower calculated rate. In contrast, using the initial RTT for the aggregation of lost packets to loss events results in more aggressive protocol behavior. In this section we argue that these two effects cancel each other out in most cases and the initial RTT can be used for both purposes.

The initial RTT only has an impact on the loss event rate when separate loss intervals are merged into a single loss interval (i.e. more than one packet is lost per RTT). From Equation (1), the number of loss events per RTT is

$$l_{RTT} = \frac{1}{\sqrt{\frac{2}{3p}} + 12\sqrt{\frac{3p}{8}}\,(1 + 32p^2)}$$

The corresponding curve is plotted in Figure 17. The maximum value is approximately 0.13 loss events per RTT. Thus, when multiple losses are aggregated to form a loss event and a loss event occurs during each RTT, this condition can persists only for a short period of time. TFMCC will reduce the sending rate due to the high loss event rate until the number of loss events per RTT is smaller than 0.13.

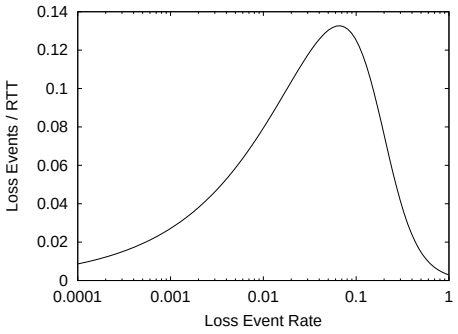

Figure 17: Loss Events per RTT

Even during the transition time, a TFMCC flow with an RTT estimate that is too high will behave more conservatively than a similar flow with a correct RTT estimate. The size of the loss intervals can only increase in proportion to the ratio of the initial RTT to the true RTT. Using Equation (4), an initial RTT that is too high by a factor of c will allow for a loss rate that is too low by a factor of c^2 resulting in the same throughput. The rate calculated at the receiver will therefore still be conservative. Numerical analysis indicates that this also holds for the complex TCP model (1) when loss event rates are less than approximately 10%.

For these reasons, it is safe to use a high initial RTT to both aggregate losses to loss events as well as to compute the rate.

The loss history must be remodeled after the first valid RTT measurement is obtained, otherwise the rate calculated by the receiver will be too high. When the lost packets and their timestamps are known, the correct loss intervals can easily be determined based on the measured RTT rather than of the initial RTT. This process can be optimized by storing information about some of the more recently lost packets and approximating the correct distribution of loss intervals.

B Initializing the Loss History

When a receiver registers its first loss event, the number of packets received thus far usually does not reflect the current loss rate. For example, when the sending rate is constrained by a lower-rate CLR, a receiver may not experience packet loss for a long period of time. Instead of the number of packets received before the first loss event, the sending rate at which the first packet loss is experienced can be used as an indicator of the bottleneck bandwidth. Slowstart results in an overshoot to a maximum of at most twice the bottleneck bandwidth. Thus, a more meaningful initial loss interval l_0 can be obtained by using the inverse of Equation (1) with half the sending rate when the first loss event occurred.

The mechanism can be facilitated by using the inverse of a simplified TCP Equation (4) presented in [11], which is easier to compute than the inverse of Equation (1) and results in a slightly more conservative estimate:

$$\begin{aligned} T_{TCP} &= \frac{cs}{t_{RTT}\sqrt{p}} \\ p &= \left(\frac{cs}{T_{TCP} \cdot t_{RTT}}\right)^2, \text{with} \, l_0 = 1/p \end{aligned}$$

where c is a constant usually set to $\sqrt{3/2}$.

However, if a receiver is still using the initial RTT when the first loss event occurs, it will underestimate the loss event rate and the initial loss interval will be too large. When the correct RTT is determined later, the receiver will consequently overestimate the fair rate. The initial loss interval must be adjusted if it is still in the loss history when the first RTT measurement is obtained. The adjusted first loss interval l_0' can be calculated as

$$l_0' = l_0 \cdot \left(\frac{t_{RTT}}{t_{RTT}^{initial}}\right)^2$$

using the simplified TCP equation.

C Storing the Previous CLR

As an option, the sender can keep information about the previous CLR after switching to a new CLR. In case the switchover is only temporary, it is possible to immediately switch back to the old CLR without the need of further feedback. Possible causes for transient switching of the CLR include short-term congestion or inaccurate one-way delay RTT adjustments. Here, the new expected rate may quickly increases above the expected rate of the previous CLR.

Storing this additional information will always result in more conservative TFMCC behavior. In particular, when network conditions for the new CLR as well as the old CLR improve simultaneously, TFMCC will switch back to the old CLR before increasing the sending rate. Since this results in a delayed reaction to improved network conditions, the information about the old CLR should be timed out after a short amount of time (on the order of a few RTTs).

D Additional Simulations

In this section we present a number of simulatons left out of the main TFMCC paper [20] for lack of space.

D.1 Asymmetric Paths

Simulations with additional traffic or different network conditions on the return path from the receiver help to verify that the protocol behaves as expected under the conditions encountered in real networks. Figure 18 shows throughput of 4 TCP flows and a TFMCC flows with an additional 0, 1, 2, and 4 TCP flows on the return path from the 4 receivers.

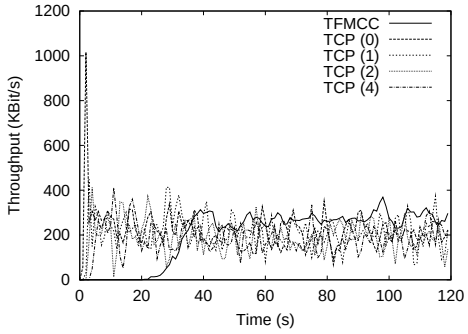

Figure 18: Competing traffic on return paths

None of the simulations differ from the case where no return traffic is present. Due to the cumulative nature of TCP ACKs, TCP throughput decreases only when the congestion on the return path reaches very high levels. The throughput of TCP flows with 0% loss, 10% loss, 20% loss, and 30% loss is depicted in Figure 19, together with a TFMCC flow with receivers at each of the four nodes. In contrast to TCP, TFMCC is insensitive to the loss of receiver reports and thus throughput is unaffected.

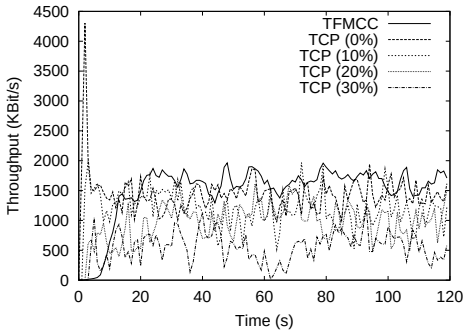

Figure 19: Lossy return paths

D.2 Responsiveness

Section 4.2 gave an overview of TFMCC's behavior in case receivers with different loss rates join the session. In addition, it is of interest how TFMCC reacts to changes in the RTT and to changes in the number of flows.

For the responsivenes to delay, a simulation setup similar to the one in Section 4.2 is used, where instead of the loss rate the delay of the links is set to 30ms, 60ms, 120ms, and 240ms respectively. Again, receivers join the multicast session in the order of their RTT. From Figure 20 we can see a behavior very similar to the one depicted in Figure 11. The small number of receivers guarantees that the correct CLR is chosen almost instantaneously. For simulations with several hundreds or thousands of receivers, the delay before the correct CLR is chosen increases, in case a large number of the receivers have not yet measured their RTT (see Figure 13). Since TFMCC is suitable for long lived flows rather than short connections, we expect this delay to be noticeable only for a limited amount of time after the startup of the flow.

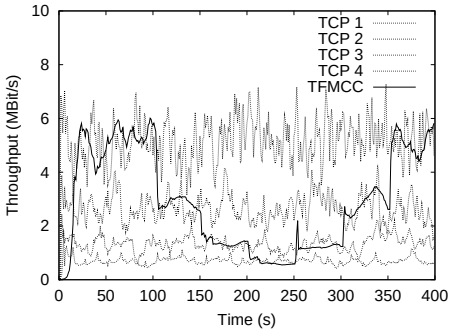

Figure 20: Responsiveness to network delay

To demonstrate TFMCC's reaction to an increase in the number of flows (and the corresponding increase in the loss rate), a TFMCC flow is run over a link with 60ms round-trip delay and 16MBit/s capacity. At 50 second intervals, several additional TCP flows are started. First 1 additional flow is set up, then 2, then 4, and then 8, such that the total number of flows doubles every 50 seconds. In Figure 21, the rates of TCP flows started at the same point in time are aggregated to improve readability. TFMCC as well as TCP show the desired behavior of settling at an average bandwidth of half of that of the previous interval. As expected, TFMCC reacts on a longer timescale than TCP, whereby the time it takes to adapt decreases with an increasing number of flows. Overall fairness is acceptable with a slightly too aggressive TFMCC.

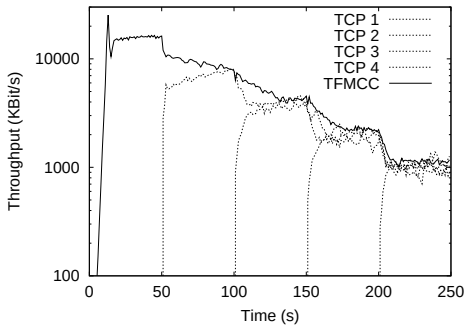

Figure 21: Responsiveness to increased congestion