INTERNATIONAL COMPUTER SCIENCE INSTITUTE
1947 Center St. • Suite 600 • Berkeley, California 94704-1198 • (510) 666-2900 • FAX (510) 666-2983

Equation-Based Congestion Control for Unicast Applications: the Extended Version*

Sally Floyd, Mark Handley[†] **Jitendra Padhye**[‡]

Jörg Widmer[§]

TR-00-003

March 2000

Abstract

This paper proposes a mechanism for equation-based congestion control for unicast traffic. Most best-effort traffic in the current Internet is well-served by the dominant transport protocol TCP. However, traffic such as best-effort unicast streaming multimedia could find use for a TCP-friendly congestion control mechanism that refrains from reducing the sending rate in half in response to a single packet drop. With our mechanism, the sender explicitly adjusts its sending rate as a function of the measured rate of loss events, where a *loss event* consists of one or more packets dropped within a single round-trip time. We use both simulations and experiments over the Internet to explore performance.

Equation-based congestion control is also a promising avenue of development for congestion control of multicast traffic, and so an additional reason for this work is to lay a sound basis for the later development of multicast congestion control.

[†] AT&T Center for Internet Research at ICSI (ACIRI)
[‡] University of Massachusetts at Amherst
[§] International Computer Science Institute (ICSI)

1 Introduction

TCP is the dominant transport protocol in the Internet, and the current stability of the Internet depends on its end-to-end congestion control, which uses an Additive Increase Multiplicative Decrease (AIMD) algorithm. For TCP, the 'sending rate' is controlled by a congestion window which is halved for every window of data containing a packet drop, and increased by roughly one packet per window of data otherwise.

End-to-end congestion control of best-effort traffic is required to avoid the congestion collapse of the global Internet [FF99]. While TCP congestion control is appropriate for applications such as bulk data transfer, some applications where the data is being played out in real-time find halving the sending rate in response to a single congestion indication to be unnecessarily severe, as it can noticeably reduce the user-perceived quality [TZ99]. TCP's abrupt changes in the sending rate have been a key impediment to the deployment of TCP's end-to-end congestion control by emerging applications such as streaming multimedia. In our judgement, equation-based congestion control is the leading candidate for a viable mechanism to provide relatively smooth congestion control for such traffic.

Equation-based congestion control was first proposed in [MF97]. Whereas AIMD congestion control backs off in response to a single congestion indication, equation-based congestion control uses a control equation that explicitly gives the maximum acceptable sending rate as a function of the recent *loss event rate*. The sender adapts its sending rate, guided by this control equation, in response to feedback from the receiver. For traffic that competes in the best-effort Internet with TCP, the appropriate control equation for equation-based congestion control is the TCP response function characterizing the steady-state sending rate of TCP as a function of the round-trip time and steady-state loss event rate.

Although there has been significant previous research on equation-based and other congestion control mechanisms [JE96, OR99, RHE99, TZ99, PKTK99, TPB, VRC98, SS98], we are still rather far from having deployable congestion control mechanisms for best-effort streaming multimedia. Section 3 presents the TCP-Friendly Rate Control (TFRC) proposal for equation-based congestion control for unicast traffic, In Section 5 we provide a comparative discussion of TFRC and previously proposed protocols. The benefit of TFRC is a more smoothly-changing sending rate than that of TCP; the cost is a more moderate response to transient changes in congestion.

One of our goals in this paper is to present a proposal for equation-based congestion control that lays the foundation for the near-term experimental deployment of congestion control for unicast streaming multimedia. Section 4 presents results from extensive simulations and experiments with the TFRC protocol, showing that equation-based congestion control using the TCP response function competes fairly with TCP. Both the simulator code and the real-world implementation are publically available. We believe that TFRC and related forms of equation-based congestion control can play a significant role in the Internet.

For most unicast flows that want to transfer data reliably and as quickly as possible, the best choice is simply to use TCP directly. However, equation-based congestion control is more appropriate for applications that need to maintain a slowly-changing sending rate, while still being responsive to network congestion over longer time periods (seconds, as opposed to fractions of a second). It is our belief that TFRC is sufficiently mature for a wider experimental deployment, testing, and evaluation.

A second goal of this work is to lay a foundation for further research within the network community on the development and evaluation of equation-based congestion control. We address a number of key concerns in the design of equation-based congestion control that have not been sufficiently addressed in previous research, including responsiveness to persistent congestion, avoidance of unnecessary oscillations, avoidance of the introduction of unnecessary noise, and robustness over a wide range of timescales.

The algorithm for calculating the loss event rate is the key design issue in equation-based congestion control, determining the tradeoffs between responsiveness to changes in congestion and the avoidance of oscillations or unnecessarily abrupt shifts in the sending rate. The discussion in Section 3 addresses these tradeoffs and describes the fundamental components of the TFRC algorithms that reconcile them.

A third goal of this work is to build a solid basis for the development of congestion control for multicast traffic. In a large multicast group, there will usually be at least one receiver that has experienced a recent packet loss. If the congestion control mechanisms require that the sender reduces its sending rate in response to each loss, as in TCP, then there is little potential for the construction of scalable multicast congestion control. Equation-based congestion control for multicast traffic has been an active area of research for several years [RMR]. As we describe in Section 6, many of the mechanisms in TFRC are directly applicable to multicast congestion control.

2 Foundations of equation-based congestion control

The basic decision in designing equation-based congestion control is to choose the underlying control equation. An application using congestion control that was significantly more aggressive than TCP could cause starvation for TCP traffic if both types of traffic were competing in a FIFO queue at a time of congestion [FF99]. From [BCC+98], a *TCP-compatible* flow is defined as a flow that, in steady-state, uses no more bandwidth than a conformant TCP running under comparable conditions. For best-effort traffic competing with TCP in the current Internet, in order to be TCP-compatible, the correct choice for the control equation is the TCP response function describing the steady-state sending rate of TCP [Flo99].

From [PFTK98], one formulation of the TCP response function is the following:

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

This gives an upper bound on the sending rate T in bytes/sec, as a function of the packet size s, round-trip time R, steady-state loss event rate p, and the TCP retransmit timeout value t_{RTO}.

An application wishing to send less than the TCP-compatible sending rate (e.g., because of limited demand) would still be characterized as TCP-compatible. However, if a significantly less aggressive response function were used, then the less aggressive traffic could encounter starvation when competing with TCP traffic in a FIFO queue. In practice, when two types of traffic compete in a FIFO queue, acceptable performance only results if the two traffic types have similar response functions.

For traffic that is not competing with TCP traffic in a FIFO queue, but is isolated from TCP traffic by some method (e.g., with per-flow scheduling, or in a separate differentiated-services class from TCP traffic), applications using equation-based congestion control could make a different choice for the underlying control equation. Issues about the merits or shortcomings of various control equations for equation-based congestion control are an active research area that we do not address further in this paper.

2.1 Viable congestion control does not require TCP

This paper proposes deployment of a congestion control algorithm that does not reduce its sending rate in half in response to a single congestion indication. Given that the stability of the current Internet rests on AIMD congestion control mechanisms in general, and on TCP in particular, a proposal for non-AIMD congestion control requires justification in terms of its suitability for the global Internet. We discuss two separate justifications, one practical and the other theoretical.

A practical justification is that the principle threat to the stability of end-to-end congestion control in the Internet comes not from flows using alternate forms of TCP-compatible congestion control, but from flows that do not use any end-to-end congestion control at all. For some of these flows (e.g., large-scale multicast, some real-time traffic), the only viable possibility for end-to-end congestion control is a mechanism that responds less drastically to a single packet drop than does TCP.

A more theoretical justification is that preserving the stability of the Internet does not require that flows reduce their sending rate by half in response to a single congestion indication. In particular, the prevention of congestion collapse simply requires that flows use some form of end-to-end congestion control to avoid a high sending rate in the presence of a high packet drop rate. Similarly, as we will show in this paper, preserving some form of "fairness" against competing TCP traffic also does not require such a drastic reaction to a single congestion indication.

For flows desiring smoother changes in the sending rate, alternatives to TCP include AIMD congestion control mechanisms that do not use a decrease-by-half reduction in response to congestion. In DECbit, which was also based on AIMD, flows reduced their sending rate to 7/8 of the old value in response to a packet drop [JRC87]. Similarly, in Van Jacobson's 1992 revision of his 1988 paper on Congestion Avoidance and Control [Jac88], the main justification for a decrease term of 1/2 instead of 7/8, in Appendix D of the revised version of the paper, is that the performance penalty for a decrease term of 1/2 is small. A related paper [FHP00] includes a relative evaluation of AIMD and equation-based congestion control.

3 The TCP-Friendly Rate Control (TFRC) Protocol

The primary goal of equation-based congestion control is not to aggressively find and use available bandwidth, but to maintain a relatively steady sending rate while still being responsive to congestion. To accomplish this, equation-based congestion control makes the tradeoff of refraining from *aggressively* seeking out available bandwidth in the manner of TCP. Thus, several of the design principles of equation-based congestion control can be seen in contrast to the behavior of TCP.

- Do not aggressively seek out available bandwidth. That is, increase the sending rate slowly in response to a decrease in the loss event rate.
- Do not reduce the sending rate in half in response to a single loss event. However, do reduce the sending rate in half in response to several successive loss events.

Additional design goals for equation-based congestion control for unicast traffic include:

- The receiver should report feedback to the sender at least once per round-trip time if it has received any packets in that interval.
- If the sender has not received feedback after several round-trip times, then the sender should reduce its sending rate, and ultimately stop sending altogether.

3.1 Protocol Overview

Applying the TCP response equation (Equation (1)) as the control equation for congestion control requires the following:

- The parameters R and p are determined. The loss event rate p must be calculated at the receiver, while the round-trip time R could be measured at either the sender or the receiver. (The other two values needed by the TCP response equation are the flow's packet size s and the retransmit timeout value t_{RTO}, which can be estimated from R.)

- The receiver sends either the parameter p or the calculated value of the allowed sending rate T back to the sender.
- The sender increases or decreases its transmission rate based on its calculation of T.

For multicast, it makes sense for the receiver to determine the relevant parameters and calculate the allowed sending rate. However, for unicast the functionality could be split in a number of ways. In our proposal, the receiver only calculates p, and feeds this back to the sender.

3.2 Sender functionality

In order to use the control equation, the sender determines the values for the round-trip time R and retransmit timeout value t_{RTO}.

The sender and receiver together use sequence numbers for measuring the round-trip time. Every time the receiver sends feedback, it echoes the sequence number from the most recent data packet, along with the time since that packet was received. In this way the sender measures the round-trip time through the network.

The sender smoothes the measured round-trip time using an exponentially weighted moving average. This weight determines the responsiveness of the transmission rate to changes in round-trip time.

The sender could derive the retransmit timeout value t_{RTO} using the usual TCP algorithm:

$$t_{RTO} = SRTT + 4 * RTT_{var}$$

where RTT_{var} is the variance of RTT and $SRTT$ is the round-trip time estimate. However, in practice t_{RTO} only critically affects the allowed sending rate when the packet loss rate is very high. Different TCPs use drastically different clock granularities to calculate retransmit timeout values, so it is not clear that equation-based congestion control can accurately model a *typical* TCP. Unlike TCP, TFRC does not use this value to determine whether it is safe to retransmit, and so the consequences of inaccuracy are less serious. In practice the simple empirical heuristic of $t_{RTO} = 4R$ works reasonably well to provide fairness with TCP.

The sender obtains the value of p in feedback messages from the receiver at least once per round-trip time.

Every time a feedback message is received, the sender calculates a new value for the allowed sending rate T using the control equation. If the actual sending rate T_{actual} is less than T, the sender may increase its sending rate.

If T_{actual} is greater than T, the sender must decrease the sending rate. We have several choices here:

- **Decrease exponentially.** Experiments show that this is undesirable because it can involve decreasing to less than T, and the resulting undershoot leads to oscillatory behavior.
- **Decrease towards T.** This might work, but there is already significant damping introduced in the measurement of p and in the smoothing of R, and so additional damping only confuses the effects of the existing damping without changing the behavior significantly.
- **Decrease to T.** This works well, and is the behavior used in all the results presented in this paper.

3.3 Receiver functionality

The receiver provides feedback to allow the sender to measure the round-trip time (RTT). The receiver also calculates the loss event rate p, and feeds this back to the sender. The calculation of the loss event rate is one of the most critical parts of TFRC, and the part that has been through the largest amount of evaluation and design iteration. There is a clear trade-off between measuring the loss event rate over a short period of time and being able to respond rapidly to changes in the available bandwidth, versus measuring over a longer period of time and getting a signal that is much less noisy.

The method of calculating the loss event rate has been the subject of much discussion and testing, and over that process several guidelines have emerged:

- The estimated loss event rate should track relatively smoothly in an environment with a stable steady-state loss event rate.
- The estimated loss rate should measure the *loss event rate* rather than the packet loss rate, where a *loss event* can consist of several packets lost within a round-trip time. This is discussed in more detail in Section 3.5.1.
- The estimated loss event rate should respond strongly to loss events in several successive round-trip times.
- The estimated loss event rate should increase only in response to a new loss event. (We note that this property is not satisfied by some of the methods described below.)
- Let a *loss interval* be defined as the number of packets between loss events. The estimated loss event rate should decrease only in response to a new loss interval that is longer than the previously-calculated average, or a sufficiently-long interval since the last loss event.

Obvious methods we looked at include the EWMA Loss Interval method, the Dynamic History Window method, and the Average Loss Interval method which is the method we chose.

- The EWMA Loss Interval method uses an exponentially weighted moving average of the number of packets between loss events. Depending on the weighting, this either puts too much weight on the most recent interval, or takes too much history into account and is slow to react to real changes.
- The Dynamic History Window method uses a history window of packets whose length is determined by the current transmission rate. This suffers from the effect that even with a perfectly periodic loss pattern, loss events entering and leaving the window cause changes to the measured loss rate, and hence add unnecessary noise to the loss signal.
- The Average Loss Interval method computes the average loss rate over the last n loss intervals. By itself, the naive Average Loss Interval method suffers from two problems: the interval since the most recent loss is not necessarily a reflection of the underlying loss event rate, and there can be sudden changes in the calculated rate due to unrepresentative loss intervals leaving the n intervals we're looking at. These concerns are addressed below.

The full Average Loss Interval method differs from the naive version in several ways. Let s_i be the number of packets in the i-th most recent loss interval, and let the most recent interval s_0 be defined as the interval containing the packets that have arrived *since the last loss*. The first difference addresses the most recent loss interval s_0. When a loss occurs, the loss interval that has been s_0 now becomes s_1, all of the following loss intervals are correspondingly shifted down one, and the new loss interval s_0 is empty. As s_0 is not terminated by a loss, it is different from the other loss intervals. It is important to ignore s_0 in calculating the average loss interval unless s_0 is large enough that including it would increase the average. This allows the calculated loss interval to track smoothly in an environment with a stable loss event rate.

The second difference from the naive method reduces the sudden changes in the calculated loss rate that could result from unrepresentative loss intervals leaving the set of loss intervals used to calculate the loss rate. The full Average Loss Interval method takes a weighted average of the last n intervals, with equal weights for the most recent $n/2$ intervals and smaller weights for older intervals. Thus the average loss interval $\hat{s}$ is calculated as follows:

$$\hat{s} = \frac{\sum_{i=1}^{n} w_i s_i}{\sum_{i=1}^{n} w_i},$$

for weights w_i:

$$w_i = 1, \ \ 1 \le i \le n/2,$$

and

$$w_i = 1 - \frac{i - n/2}{n/2 + 1}, \ \ n/2 < i \le n.$$

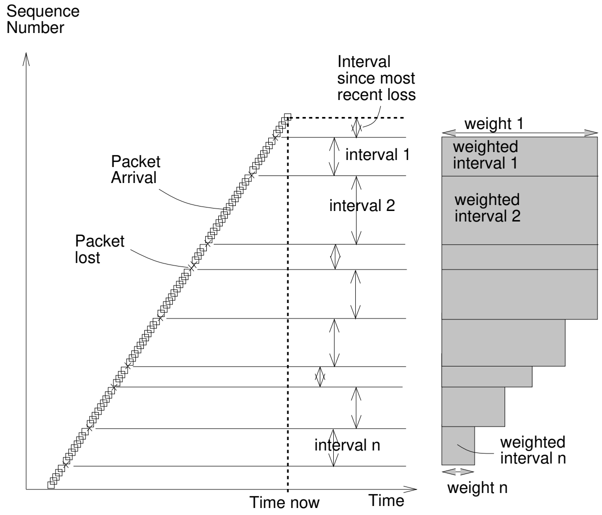


Figure 1: Weighted intervals between loss used to calculate loss probability.

For $n = 8$, this gives weights of: $w_1, w_2, w_3, w_4 = 1$; $w_5 = 0.8$; $w_6 = 0.6$; $w_7 = 0.4$; and $w_8 = 0.2$.

The full Average Loss Interval method also calculates $\hat{s}_{new}$, which is the average loss interval calculated over intervals s_0 to s_{n-1} rather than over s_1 to s_n

$$\hat{s}_{new} = \frac{\sum_{i=0}^{n-1} w_{i+1} s_i}{\sum_{i=1}^{n} w_i}$$

To include s_0 only at the correct times, as discussed above, the value actually used for the average loss interval is

$$\max(\hat{s}, \hat{s}_{new})$$

.

The sensitivity to noise of the calculated loss rate depends on the value of n. In practice a value of $n = 8$, with the most recent four samples equally weighted, appears to be a lower bound that still achieves a reasonable balance between resilience to noise and responding quickly to real changes in network conditions. Section 4.4 describes experiments that validate the value of $n = 8$. However, we have not carefully investigated alternatives for the relative values of the weights.

Because the Average Loss Interval method averages over a number of loss intervals, rather than over a number of packet arrivals, the naive Average Loss Interval method responds reasonably rapidly to a sudden increase in congestion, but is slow to respond to a sudden decrease in the loss rate. For this reason we deploy history discounting as a component of the full Average Loss Interval method, to allow a more timely response to a sustained decrease in congestion. History discounting is used by the TFRC receiver after the identification of a particularly long interval since the last dropped packet, to smoothly discount the weight given to older loss intervals.

The details of the discounting mechanism are as follows: If $s_0 > 2\hat{s}_{(i \ge 1)}$, then the most recent loss interval s_0 is considerably longer than the recent average, and the weights for the older loss intervals are discounted correspondingly. The weights for the older loss intervals are discounted by using the following discount factor:

$$d_i = \max\left(0.5, \frac{2\hat{s}_{(i\geq 1)}}{s_0}\right), \quad \text{for } \ i > 0,$$

$$d_0 = 1.$$

The lower bound of 0.5 on the discount factor ensures that past losses will never be completely forgotten, regardless of the number of packet arrivals since the last loss.

When history discounting is invoked, this gives the following estimated loss interval:

$$\hat{s} = \frac{\sum_{i=0}^{n-1} d_i w_{i+1} s_i}{\sum_{i=1}^{n} d_{i-1} w_i}.$$

When loss occurs and the old interval s_0 is shifted to s_1, then the discount factors are also shifted, so that once an interval is discounted, it is never un-discounted, and its discount factor is never increased. In normal operation, in the absence of history discounting, $d_i = 1$ for all values of i. We do not describe all the details of the history discounting mechanism in this paper, but the reader is referred to NS for a detailed implementation.[1] History discounting (also called proportional deweighing) is described in more detail in [Wid00] in Sections 3.7 and 4.8.1.

3.3.1 Illustrating the receiver's estimated loss rate

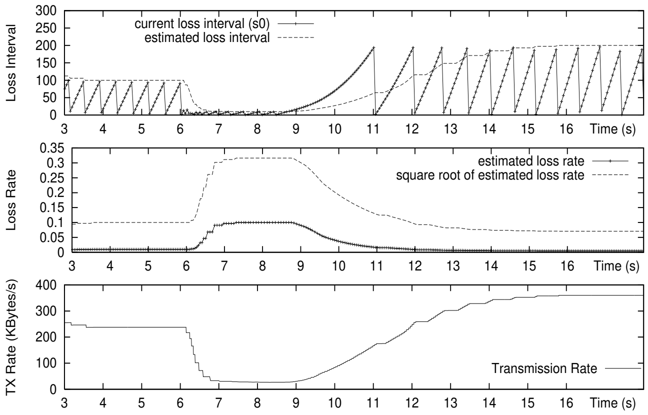


Figure 2: Illustration of the Average Loss Interval method with idealized periodic loss.

Figure 2 shows a simulation using the full Average Loss Interval method for calculating the loss event rate at the receiver. The link loss rate is 1% before time 6, then 10% until time 9, and finally 0.5% until the end of the run. This simulation is rather unrealistic because the loss is periodic, but this illustrates the mechanism more clearly.

[1]The history discounting mechanism is in the procedure est_loss() in the file tfrc-sink.cc in the NS distribution.

For the top graph, the solid line shows the number of packets in the most recent loss interval, as calculated by the receiver once per round-trip time before sending a status report. The smoother dashed line shows the receiver's estimate of the average loss interval. The middle graph shows the receiver's estimated loss event rate p, which is simply the inverse of the average loss interval, along with $\sqrt{p}$. The bottom graph shows the sender's transmission rate which is calculated from p.

Several things are noticeable from these graphs:

- Before t=6, the loss rate is constant and the Average Loss Interval method gives a completely stable measure of the loss rate.
- When the loss rate increases, the transmission rate is rapidly reduced.
- When the loss rate decreases, the transmission rate increases in a smooth manner, with no step increases even when older (10 packet) loss intervals are excluded from the history. With naive loss interval averaging we would have seen undesirable step-increases in the estimated loss interval, and hence in the transmission rate.

3.4 Improving Stability

One of the goals of the TFRC protocol is to avoid the characteristic oscillations in the sending rate that result from TCP's AIMD congestion control mechanisms. In controlling oscillations, a key issue in the TFRC protocol concerns the response function's specification of the allowed sending rate as inversely proportional to the measured RTT. A relatively prompt response to changes in the measured round-trip time is helpful to prevent flows from overshooting the available bandwidth after an uncongested period. On the other hand, an over-prompt response to changes in the measured round-trip time can result in unnecessary oscillations.

If the value of the EWMA weight for calculating the average RTT is set to a small value such as 0.1 (meaning that 10% of the weight is on the most recent sample) then TFRC does not react strongly to increases in RTT. In this case, we tend to see oscillations when a small number of TFRC flows share a high-bandwidth link with DropTail queuing; the TFRC flows overshoot the link bandwidth and then experience loss over several RTTs. The result is that they backoff together by a significant amount, and then all start to increase their rate together. This is shown for a single flow in Figure 3 as we increase the buffer size in Dummynet [Riz98]. Although not disastrous, the resulting oscillation is undesirable for applications and can reduce network utilization. This is similar in some respects to the global oscillation of TCP congestion control cycles.

If the EWMA weight is set to a high value such as 0.5, then TFRC reduces its sending rate strongly in response to an increase in RTT, giving a delay-based congestion avoidance behavior. However, because the sender's response is de-

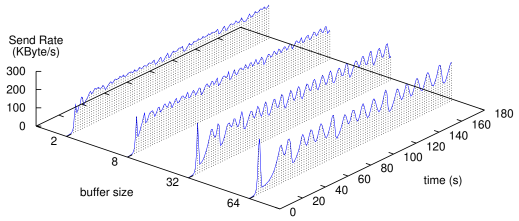


Figure 3: Oscillations of a TFRC flow over Dummynet, EWMA weight 0.05 for calculating the RTT.

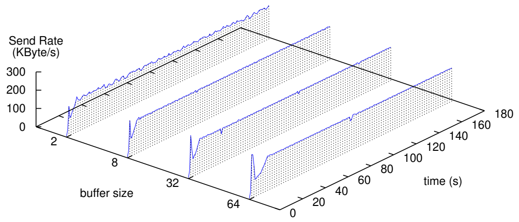


Figure 4: TFRC flow over Dummynet: oscillations prevented

layed and the sending rate is directly proportional to $1/R$, it is possible for short-term oscillations to occur, particularly with DropTail queues. While undesirable, these oscillations tend to be less of a problem than the oscillations with smaller values of the EWMA weight.

What we desire is a middle ground, where we gain some short-term delay-based congestion avoidance, but in a form that has less gain than simply making the rate inversely proportional to the most recent RTT measurement. To accomplish this, we use a small value for the EWMA weight in calculating the average round-trip time R in Equation (1), and apply the increase or decrease functions as before, but then set the interpacket-spacing as follows:

$$t_{inter-packet} = \frac{s\sqrt{R_0}}{T * M}$$

where R_0 is the most recent RTT sample, and M is the average of the square-roots of the RTTs, calculated using an exponentially weighted moving average with the same time constant we use to calculate the mean RTT. Thus, we gain the benefits of short-term delay-based congestion avoidance, but with a lower feedback loop gain so that oscillations in RTT damp themselves out, as shown in Figure 4. The experiments in Figure 3 did not use this adjustment to the interpacket spacing.

3.4.1 Slowstart

The initial rate-based slow-start procedure should be similar to the window-based slow-start procedure followed by TCP where the sender roughly doubles its sending rate each round-trip time. However, TCP's ACK-clock mechanism provides a limit on the overshoot during slow start. No more that two outgoing packets can be generated for each acknowledged data packet, so TCP cannot send at more than twice the bottleneck link bandwidth.

A rate-based protocol does not have this natural self-limiting property, and so a slow-start algorithm that doubles its sending rate every measured RTT can overshoot the bottleneck link bandwidth by significantly more than a factor of two. A simple mechanism to limit this overshoot is to have the receiver feed back the rate that packets arrived at the receiver during the last measured RTT. If loss occurs, slowstart is terminated, but if loss doesn't occur the sender sets its rate to:

$$T_{actual,i+1} = min\left(2T_{actual,i}, 2T_{received,i}\right)$$

This limits the slow-start overshoot to be no worse than that of TCP.

When the loss occurs that causes slowstart to terminate, there is no appropriate loss history from which to calculate the loss fraction for subsequent RTTs. The interval until the first loss is not very meaningful as the rate changes so rapidly during this time. The solution is to assume that the correct initial data rate is half of the rate when the loss occurred; the factor of one-half results from the delay inherent in the feedback loop. We then calculate the expected loss interval that would be required to produce this data rate, and use this synthetic loss interval to seed the history mechanism. Real loss-interval data then replaces this synthetic value when it becomes available.

3.5 Discussion of protocol features

3.5.1 Loss Fraction vs. Loss Event Fraction

The obvious way to measure loss is as a loss fraction calculated by dividing the number of packets that were lost by the number of packets transmitted. However this does not accurately model the way TCP responds to loss. Different variants of TCP cope differently when multiple packets are lost from a window; Tahoe, NewReno, and Sack TCP implementations generally halve the congestion window once in response to several losses in a window, while Reno TCP typically reduces the congestion window twice in response to multiple losses in a window of data.

Where routers use RED queue management, multiple packet drops in a window of data are less common, but with drop-tail queue management it is common for several packets in the same round-trip-time to be lost when the queue overflows. These multiple drops can result in multiple packets dropped from a window of data from a single flow, resulting in a significant difference between the loss fraction and the loss event fraction for that flow.

Because we are trying to emulate the best behavior of a conformant TCP implementation, we measure loss as a *loss event fraction*. Thus we explicitly ignore losses within a round-trip time that follow an initial loss, and model a transport protocol that reduces its window at most once for congestion notifications in one window of data. This closely models the mechanism used by most TCP variants.

To see how the loss-event fraction differs from the regular loss fraction in the presence of random packet loss, consider a flow that sends N packets per round-trip time, and assume a Bernoulli loss model with loss probability p_{loss}. The probability that at least one packet is lost in a given round-trip time is $1-(1-p_{loss})^N$. Therefore the loss-event fraction p_{event}, calculated as number of loss events per packet sent, is given by:

$$p_{event} = \frac{1-(1-p_{loss})^N}{N}$$

Note that for a fixed loss probability, the faster the sender transmits, the lower the loss-event fraction. However, the sending rate is determined by the congestion control scheme, and so itself depends on p_{event}. For a very high loss environment where the congestion window is rarely higher than one, and for a low loss environment, there will be little difference between the packet loss rate and the loss event rate for a flow. However, for a moderate loss environment where the congestion window is usually higher than one, there is some difference between the two. A more formal discussion of this problem is presented in [RR99].

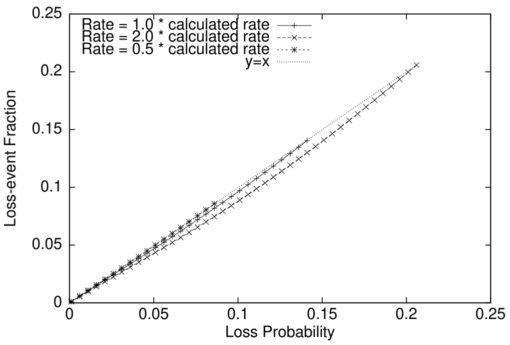


Figure 5: Loss-events per packet as a function of loss probability and error in the calculated transmission rate

Figure 5 shows the loss-event fraction as a function of loss probability for a flow that obeys Equation (1), and also for a flow transmitting at twice this rate and a flow transmitting at half this rate. From Equation (1), for a TCP retransmit timeout value t_{RTO} of $4RTT$, the average window size N as a function of p_{loss} is as follows:

$$N = \frac{1}{\sqrt{\frac{2p_{loss}}{3}} + 12\sqrt{\frac{3p_{loss}}{8}}p_{loss}(1+32p_{loss}^2)}. \qquad (2)$$

Given p_{loss}, it is then possible to calculate p_{event} for window sizes N, $2N$, and $N/2$, for flows transmitting at the calculated rate, twice this rate, and half the rate, respectively.

As Figure 5 shows, for high and low loss rates the difference between p_{loss} and p_{event} is small. For moderate loss rates, the difference between p_{loss} and p_{event} can be at most 10% for these flows. Thus, for congestion-controlled flows, the difference in the measured loss event rate is not very sensitive to variations about the correct data rate.

The version of the TCP response function in Equation (1) is based in some respects on the loss event rate, and in other respects on the packet loss rate. In particular, the response function in Equation (1) models Reno TCP, where multiple losses in a window cause a retransmission timeout. Ideally, this response function would be replaced with a TCP response function based on a model of Sack TCP and on loss event rates rather than on packet drop rates.

3.5.2 Increasing the Transmission Rate

One issue to resolve is how to increase the sending rate when the rate given by the control equation is greater than the current sending rate. As the loss rate is not independent of the transmission rate, to avoid oscillatory behavior it might be necessary to provide damping, perhaps in the form of restricting the increase to be small relative to the sending rate during the period that it takes for the effect of the change to show up in feedback that reaches the sender.

In practice, the calculation of the loss rate by the method above provides sufficient damping, and there is little need to explicitly bound the increase. As shown in Appendix A.1, given a fixed RTT and no history discounting, the increase in transmission rate is limited to about 0.14 packets per RTT every RTT (using Equation 1).

An increase in transmission rate can result from the inclusion of new packets in the most recent inter-loss interval at the receiver. If A is the number of packets in the TFRC flow's average loss interval, and w is the fraction of the weight on the most recent loss interval, then the transmission rate cannot increase by more than δ_T packets/RTT every RTT, where:

$$\delta_T = 1.2\left(\sqrt{A + w1.2\sqrt{A}} - \sqrt{A}\right)$$

The derivation is given in Appendix A.1 assuming the simpler TCP response function from [MF97] for the control equation. This behavior has been confirmed in simulations with TFRC. This behavior has also been numerically modeled for the TCP response function in Equation (1), giving similar results for low loss-rate environments but with significantly lower increase rates in high loss-rate environments.

As changes in measured RTT are already damped using an EWMA, even with the maximum history discounting ($w = 1$), this increase rate does not exceed one packet per RTT every RTT, which is the rate of increase of a TCP flow in congestion avoidance mode.

3.5.3 The response to persistent congestion

Simulations in Appendix A.2 show that, in contrast to TCP, TFRC requires from three to eight round-trip times to reduce its sending rate in half in response to persistent congestion. As discussed in Appendix A.1, this slower response to congestion is coupled with a slower increase in the sending rate than that of TCP. In contrast to TCP's increase of the sending rate by one packet/RTT for every round-trip time without congestion, TFRC generally does not increase its sending rate at all until a longer-than-average period has passed without congestion. At that point, given an environment with stable round-trip times, TFRC increases the sending rate by 0.14 packets per round-trip; after an extended absence of congestion, TFRC begins to increase its sending rate by 0.28 packets per round-trip time. Thus the milder decrease of TFRC in response to congestion is coupled with a considerably milder increase in the absence of congestion.

4 Experimental Evaluation

We have tested TFRC extensively across the public Internet, in the Dummynet network emulator [Riz98], and in the *ns* network simulator. These results give us confidence that TFRC is remarkably fair when competing with TCP traffic, that situations where it performs very badly are rare, and that it behaves well across a very wide range of network conditions. In the next section, we present a summary of ns simulation results, and in section 4.3 we look at behavior of the TFRC implementation over Dummynet and the Internet.

4.1 Simulation Results

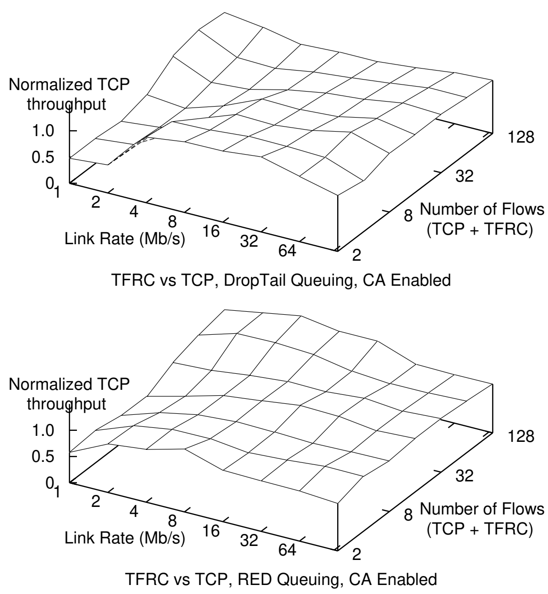


Figure 6: TCP flow sending rate while co-existing with TFRC

To demonstrate that it is feasible to widely deploy TFRC we need to demonstrate that it co-exists acceptably well when sharing congested bottlenecks of many kinds with TCP traffic of different flavors. We also need to demonstrate that it behaves well in isolation, and that it performs acceptably over a wide range of network conditions. There is only space here for a summary of our findings, but we refer the interested reader to [Pad00] for more detailed results, and to the simulator code in the *ns* distribution.

Figure 6 illustrates the fairness of TFRC when competing with TCP Sack traffic in both DropTail and RED queues. In these simulations n TCP and n TFRC flows share a common bottleneck; we vary the number of flows and the bottleneck bandwidth, and scale the queue size with the bandwidth. The graph shows the mean TCP throughput over the last 60 seconds of simulation, normalized so that a value of one would be a fair share of the link bandwidth. The network utilization is always greater than 90% and often greater than 99%, so almost all of the remaining bandwidth is used by the TFRC flows. These figures illustrate than TFRC and TCP co-exist fairly across a wide range of network conditions, and that TCP throughput is similar to what it would be if the competing traffic was TCP instead of TFRC.

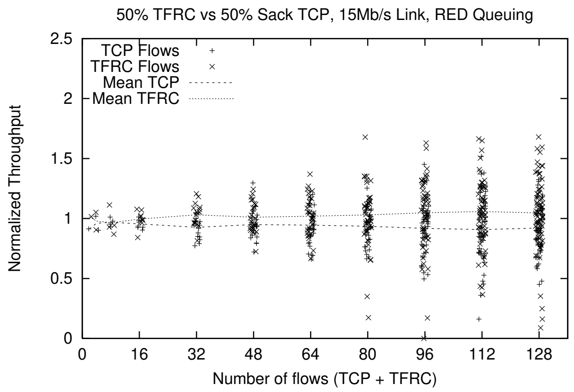
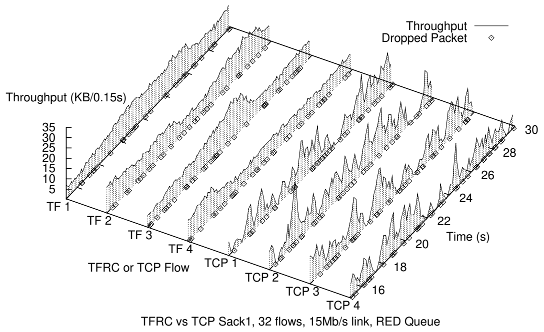


Figure 7: TCP competing with TRFC, with RED.

The graphs do show that there are some cases (typically where the mean TCP window is very small) where TCP suffers. This appears to be because TCP is more bursty than TFRC. When we modify TFRC to send two packets every two inter-packet intervals, TCP competes more fairly in these cases. However this is not something we would recommend for normal operation.

Although the mean throughput of the two protocols is rather similar, the variance can be quite high. This is illustrated in Figure 7 which shows the 15Mb/s data points from Figure 6. Each column represents the results of a single simulation, and each data point is the normalized mean throughput of a single flow. Typically, the TCP flows have higher variance than the TFRC flows, but if we replace all the flows with TCP flows this variance doesn't change greatly. In general, the variance between flows increases as the bandwidth per flow decreases. This is to be expected as Equation (1) indicates that TCP (and hence also TFRC) becomes more sensitive to loss as the loss rate increases, which it must do at lower bandwidths.

We have also looked at Tahoe and Reno TCP implementations and at different values for TCP's timer granularity. Although Sack TCP with relatively low timer granularity does better against TFRC than the alternatives, their performance



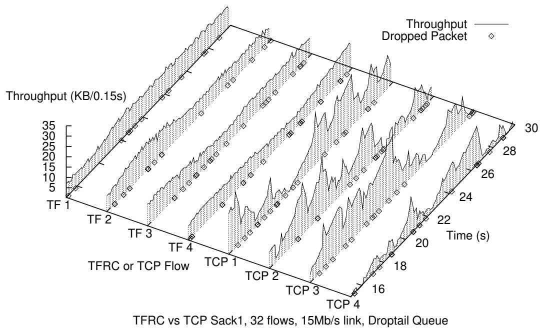


Figure 8: TFRC and TCP flows from Figure 6, for $n = 16$.

is still quite respectable.

Figure 8 shows the throughput for eight of the flows (four TCP, four TFRC) from Figure 6, for the simulations with a 15Mb/s bottleneck and 32 flows in total. The graphs depict each flow's throughput on the congested link during the second half of the 30-second simulation, where the throughput is averaged over 0.15 sec intervals; slightly more than a typical round-trip time for this simulation. In addition, a 0.15 sec interval seems to be plausible candidate for a minimum interval over which bandwidth variations would begin to be noticeable to multimedia users.[2]

Figure 8 clearly shows the main benefit for equation-based congestion control over TCP-style congestion control for unicast streaming media, which is the relative smoothness in the sending rate. A comparison of the RED and Drop-Tail simulations in Figure 8 also shows how the reduced queuing delay and reduced round-trip times imposed by RED require a higher loss rate to keep the flows in check.

4.1.1 Performance at various timescales

We are primarily interested in two measures of performance of the TFRC protocol. First, we wish to compare the average send rates of a TCP flow and a TFRC flow experiencing similar network conditions. Second, we would like to compare the "smoothness" of these send rates. Ideally, we would like for a TFRC flow to achieve the same average send rate as that of a TCP flow, and yet have less variability. The timescale at which the send rates are measured affects the values of these measures. Thus, we first define the send rate of a given data flow F at time t, measured at a timescale δ:

$$R_{\delta,F}(t) = \frac{s* \text{ packets sent by F between } t \text{ and } t+\delta}{\delta}, \quad (3)$$

for s the packet size in bytes. We characterize the send rate of the flow between time t_0 and t_1, where $t_1 = t_0 + n\delta$, by the time series: $\{R_{\delta,F}(t_0 + i*\delta)\}_{i=0}^{n}$. The coefficient of variation (CoV), which is the ratio of standard deviation to the average, of this time series can be used as a measure of variability [Jai91] of the sending rate of the flow at timescale δ. A lower value implies a smoother flow.

To compare the send rates of two flows at a given time scale, we define the equivalence at time t:

$$e_{\delta,a,b}(t) = \min\left(\frac{R_{\delta,a}(t)}{R_{\delta,b}(t)}, \frac{R_{\delta,b}(t)}{R_{\delta,a}(t)}\right), \quad (4)$$

$$R_{\delta,a}(t) > 0 \text{ or } R_{\delta,b}(t) > 0$$

Taking the minimum of the two ratios ensures that the resulting value remains between 0 and 1. Note that the equivalence of two flows at a given time is defined only when at least one of the two flows has a non-zero send rate. The equivalence of two flows between time t_0 and t_1 can be characterized by the time series: $\{e_{\delta,a,b}(t_0 + i*\delta)\}_{i=0}^{n}$. The average value of the defined elements of this time series is called the equivalence ratio of the two flows at timescale δ. The closer it is to 1, the more "equivalent" the two flows are. We choose to take average instead of the median to capture the impact of any outliers in the equivalence time series. We can compute the equivalence ratio between a TCP flow and a TFRC flow, between two TCP flows or between two TFRC flows. Ideally, the ratio would be very close to 1 over a broad range of timescales between two flows of the same type experiencing the same network conditions .

4.1.2 Performance with long-duration background traffic

For measuring the steady performance of the TFRC protocol, we consider the simple well-known single bottleneck (or "dumbbell") simulation scenario. The access links are sufficiently provisioned to ensure that any packet drops/delays due to congestion occur only at the bottleneck bandwidth.

We considered several simulation scenarios, but illustrate here a scenario with a bottleneck bandwidth of 15Mbps and a RED queue.[3] To plot the graphs, we monitor the performance

[2]The simulations in Figure 8 were run with RED queue management on the 15 Mbps congested link, with the RED parameters set as follows: *min_thresh* is set to 25 packets, *max_thresh* is set to five times min_thresh, max_p is set to 0.1, and the *gentle_* parameter is set to true.

[3]The bottleneck delay is 50ms, packet size is 1000 bytes, the bottleneck queue runs RED with *gentle* enabled, a total buffer of 100 packets, a $minthresh$ of 10 and a $maxthresh$ of 50. There are 16 SACK TCP and 16 TFRC flows. The simulation duration is 150 seconds, and the results are from the last 100 seconds of the simulation. The round-trip time of each flow, excluding the queuing delay, is random, uniformly distributed between 80 and 120 milliseconds. The flows are started at random times, uniformly distributed between 0 and 10 seconds.

of one flow belonging to each protocol. The graphs are the result of averaging 14 such runs, and the 90% confidence intervals are shown. The loss rate observed at the bottleneck router was about 0.1%.

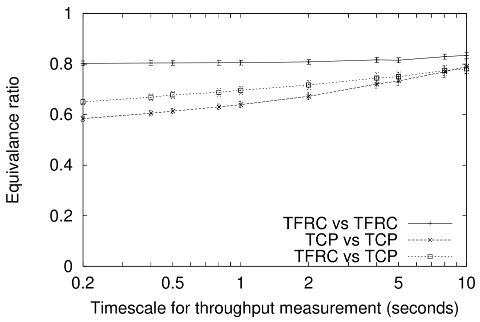


Figure 9: TCP and TFRC equivalence

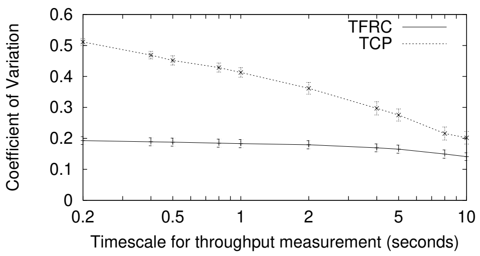


Figure 10: Coefficient of Variation of TCP and TFRC

Figure 9 shows the equivalence ratios of TCP and TFRC as a function of the timescale of measurement. Curves are shown for the mean equivalence ratio between pairs of TCP flows, between pairs of TFRC flows, and between pairs of flows of different types. The equivalence ratio of TCP and TFRC is between 0.6 to 0.8 over a broad range of timescales. The measures for TFRC pairs and TCP pairs show that the TFRC flows are "equivalent" to each other on a broader range of timescales than the TCP flows.

Figure 10 shows that the send rate of TFRC is smoother than that of TCP over a broad range of timescales. Both this and the better TFRC equivalence ratio are due to the fact that TFRC responds only to the aggregate loss rate, and not to individual loss events.

From these graphs, we conclude that in an environment dominated by long-duration flows, the TFRC transmission rate is comparable to that of TCP, and is smoother than an equivalent TCP flow across almost any timescale that might be important to an application.

4.1.3 Performance with ON-OFF flows as background traffic

In this simulation scenario, we model the effects of competing web-like traffic (very small TCP connections, some UDP flows). It has been reported in [PKC96] that WWW-related traffic tends to be self-similar in nature. In [WTSW95], it is shown that self-similar traffic may be created by using several ON/OFF UDP sources whose ON/OFF times are drawn from heavy-tailed distributions such as the Pareto distribution. Figures 11-13 present results from simulations in which we simulate such background traffic. The mean ON time is 1 second and the mean OFF time is 2 seconds, and during ON time each source sends at 500Kbps. The number of simultaneous connections is varied between 50 and 150 and the simulation is run for 5000 seconds. The results are averages of 10 runs. The bottleneck link characteristics are the same as in the previous simulation. There are two monitored connections: a long-duration TCP connection and a long-duration TFRC connection. We measure the send rates on several different timescales and show the results in Figures 12 and 13.

These simulations produce a wide range of loss rates, as shown in Figure 11. From the results in Figure 12, we can see that at low loss rates the equivalence ratio of TFRC and TCP connections is between 0.7 to 0.8 over a broad range of timescales, which is similar to the steady-state case. At higher loss rates the equivalence ratio is low at all but the longest timescales because packets are sent so rarely, and any interval in which only one of the flow sends no packets gives a value of zero in the equivalence time series, while the intervals in which neither flow sends any packets are not counted. This tends to result in a lower equivalence ratio. However, on long timescales, even at 40% loss (150 ON/OFF sources), the equivalence ratio is still 0.4, meaning that one flow gets about 40% more than its fair share and one flow got 40% less. Thus TFRC is seen to be comparable to TCP over a wide range of loss rates even when the background traffic is very variable.

Figure 13 shows that the send rate of TFRC is much smoother than the send rate of TCP, especially when the loss rate is high. Note that the CoV for both flows is much higher compared to the values in Figure 10 at comparable timescales. This is due to the hight loss rates and the variable nature of background traffic in these simulations.

4.2 Effects of TFRC on queue dynamics

Because TFRC increases its sending rate more slowly than TCP, and responds more mildly to a single loss event, it is reasonable to expect queue dynamics will be slightly different. However, because TFRC's slow-start procedure and long-term response to congestion are similar to those of TCP, we expect some correspondence between the queueing dynamics imposed by TRFC, and the queueing dynamics imposed by TCP.

Figure 14 shows 40 long-lived flows, with start times spaced out over the first 20 seconds. The congested link is 15 Mbps, and round-trip times are roughly 45 ms. 20% of the link bandwidth is used by short-lived, "background" TCP traffic, and there is a small amount of reverse-path traffic as well. Each graph in Figure 14 shows the queue size at the congested link. In the top graph the long-lived flows are TCP, and in the bottom graph they are TFRC. Both simulations have 99% link

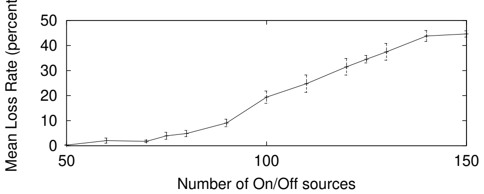


Figure 11: Loss rate at the bottleneck router, with ON-OFF background traffic

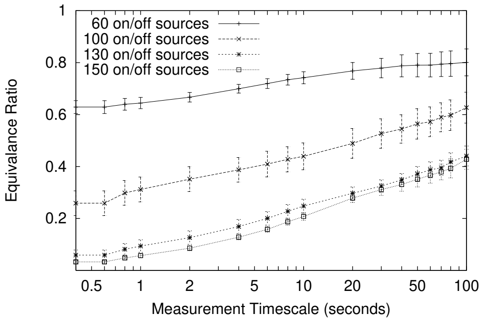


Figure 12: TCP equivalence with TFRC, with ON-OFF background traffic

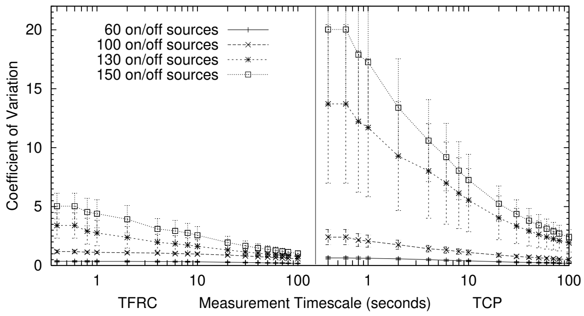


Figure 13: Coefficient of Variation of TFRC (left) and TCP (right), with ON-OFF background traffic

utilization; the packet drop rate at the link is 4.9% for the TCP simulations, and 3.5% for the TFRC simulations. As Figure 14 shows, the TFRC traffic does not have a negative impact on queue dynamics in this case.

We have run similar simulations with RED queue management, with different levels of statistical multiplexing, with a mix of TFRC and TCP traffic, and with different levels of background traffic and reverse-path traffic, and have compared link utilization, queue occupancy, and packet drop rates. While we have not done an exhaustive investigation, particularly at smaller time scales and at lower levels of link utilization, we do not see a negative impact on queue dynamics from TFRC traffic.

4.3 Implementation results

We have implemented the TFRC algorithm, and conducted many experiments to explore the performance of TFRC in the Internet. Our tests include two different transcontinental links, and sites connected by a microwave link, T1 link, OC3 link, cable modem, and dial-up modem. In addition, conditions unavailable to us over the Internet were tested against real TCP implementations in Dummynet. Full details of the experiments are available in [Wid00].

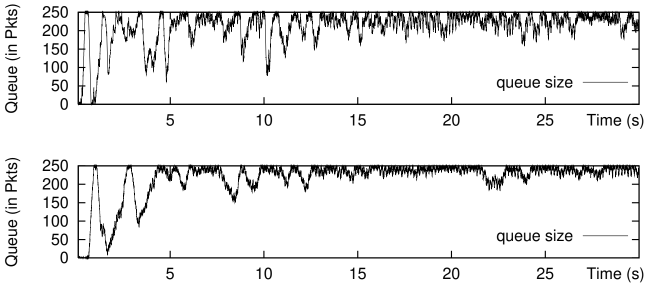


Figure 14: 40 long-lived TCP (top) and TFRC (bottom) flows, with Drop-Tail queue management.

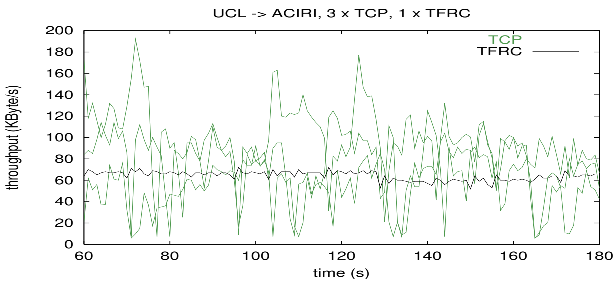


Figure 15: Three TCP flows and one TFRC flow over the Internet.

To summarise all the results, TFRC is generally fair to TCP traffic across the wide range of network types and conditions we examined. Figure 15 shows a typical experiment with three TCP flows and one TFRC flow running concurrently from London to Berkeley, with the bandwidth measured over one-second intervals. In this case, the transmission rate of the TFRC flow is slightly lower, on average, than that of the TCP flows. At the same time, the transmission rate of the TFRC flow is smooth, with a low variance; in contrast, the bandwidth used by each TCP flow varies strongly even over relatively short time periods, as shown in Figure 17. Comparing this with Figure 13 shows that, in the Internet, both TFRC and TCP perform very similarly to the lightly loaded (50 sources) "ON/OFF" simulation environment which had less than 1% loss. The loss rate in these Internet experiments ranges from 0.1% to 5%. Figure 16 shows that fairness is also rather similar in the real world, despite the Internet tests being performed with less optimal TCP stacks than the Sack TCP in the simulations.

We found only a few conditions where TFRC was less fair to TCP or less well behaved:

- In conditions where the network is overloaded so that flows achieve close to one packet per RTT, it is possible

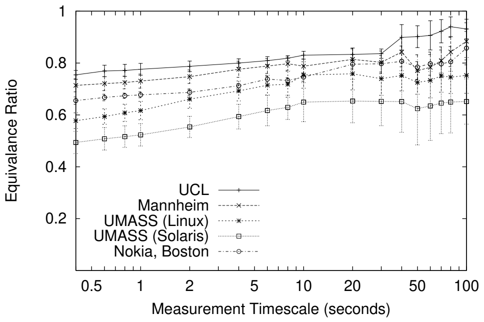


Figure 16: TCP equivalence with TFRC over different Internet paths

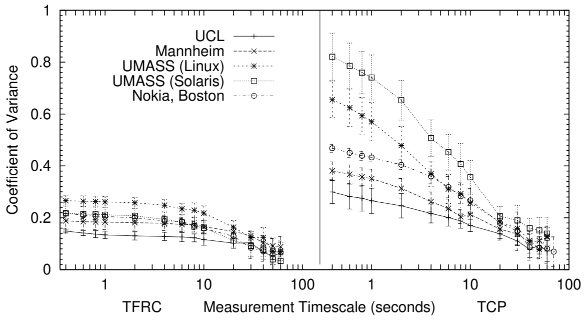


Figure 17: Coefficient of Variation of TFRC (left) and TCP (right) over different Internet paths

for TFRC to get significantly more than its fair share of bandwidth.

- Some TCP variants we tested against exhibited undesirable behavior that can only be described as "buggy".
- With an earlier version of the protocol we experienced what appears to be a real-world example of a phase effect over the T1 link from Nokia when the link was heavily loaded.

The first condition is interesting because in simulations we do not normally see this problem. This issue occurs because at low bandwidths caused by high levels of congestion, TCP becomes more sensitive to loss due to the effect of retransmission timeouts. The TCP throughput equation models the effect of retransmission timeouts moderately well, but the t_{RTO} (TCP retransmisson timeout) parameter in the equation cannot be chosen accurately. The FreeBSD TCP used for our experiments has a 500ms clock granularity, which makes it rather conservative under high-loss conditions, but not all TCPs are so conservative. Our TFRC implementation is tuned to compete fairly with a more aggressive SACK TCP with low clock granularity, and so it is to be expected that it out-competes an older more conservative TCP. Similarly unfair conditions are also likely to occur when different TCP variants compete under these conditions.

Experiments from UMass to California gave very different fairness depending on whether the TCP sender was running Solaris 2.7 or Linux. The Solaris machine has a very aggressive TCP retransmission timeout, and appears to frequently retransmit unnecessarily, which hurts its performance [Pax97]. Figure 16 shows the results for both Solaris and Linux machines at UMass; the Linux machine gives good equivalence results whereas Solaris does more poorly. That this is a TCP defect is more obvious in the CoV plot (Figure 17) where the Solaris *TFRC* trace appears normal, but the Solaris *TCP* trace is abnormally variable.

The apparent phase effect occured when a large number of TFRC flows compete with a TCP flow over the T1 bottleneck link out of Nokia. We don't have conclusive evidence but it appears that, without interpacket spacing adjustment as described in Section 3.4, the TFRC flows were sufficiently smooth that the TCP flow suffered from a poor interaction between its own burstiness and a full DropTail queue situated very close to the sources. Adding the interpacket spacing adjustment introduced sufficient small short-term variations in TFRC's throughput (and hence in the DropTail buffer utilization) due to small queuing variations downstream of the bottleneck that TCP's burstiness was less of a hinderence and fairness improved greatly. Figure 16 shows TFRC with this mechanism enabled, and the Nokia flow is performing normally.

We also ran simulations and experiments to look for the synchronization of sending rate of TFRC flows (i.e., to look for parallels to the synchronizing rate decreases among TCP flows when packets are dropped from multiple TCP flows at the same time [SZC90]). We found synchronization of TFRC flows only in a very small number of experiments with very low loss rates. When the loss rate increases, small differences in the experienced loss patterns causes the flows to desynchronize. This is discussed briefly in Section 6.3 of [Wid00].

4.4 Testing the Loss Predictor

As described in Section 3.3, the TFRC receiver uses eight inter-loss intervals to calculate the loss event rate, with the oldest four intervals having decreasing weights. One measure of the effectiveness of this estimation of the past loss event rate is to look at its ability to *predict the immediate future loss rate* when tested across a wide range of real networks. Figure 18 shows the average predictor error and the average of the standard deviation of the predictor error for different history sizes (measured in loss intervals) and for constant weighting (left) of all the loss intervals versus decreasing the weights of older intervals (right). The figure is an average across a large set of Internet experiments including a wide range of network conditions.

Prediction accuracy is not the only criteria for choosing a loss estimation mechanism, as stable steady-state throughput and quick reaction to changes in steady-state are perhaps equally important. However these figures provide experimental confirmation that the choices made in Section 3.3 are reasonable.

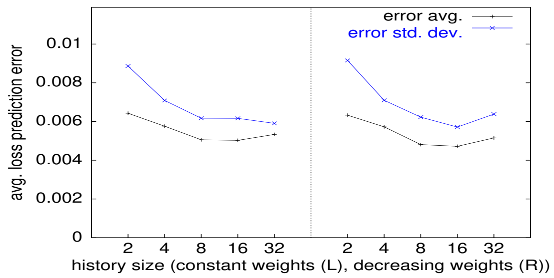


Figure 18: Prediction quality of TFRC loss estimation

5 Summary of related work

The unreliable unicast congestion control mechanisms closest to TCP maintain a congestion window which is used directly [JE96] or indirectly [OR99] to control the transmission of new packets. We believe that since [JE96] uses TCP mechanisms directly, comparison results will not be much different than those described in the previous section. In the TEAR protocol (TCP Emulation at the Receivers) from [OR99], which can be used for either unicast or multicast sessions, the receiver emulates the congestion window modifications of a TCP sender, but then makes a translation from a window-based to a rate-based congestion control mechanism. The receiver maintains an exponentially weighted moving average of the congestion window, and divides this by the estimated round-trip time to obtain a TCP-friendly sending rate. At the time of writing this paper, we did not have access to sufficient information about TEAR to allow us to perform comparative studies.

A class of unicast congestion control mechanisms one step removed from those of TCP are those that use additive increase, multiplicative decrease (AIMD) in some form, but do not apply AIMD to a congestion window. The Rate Adaptation Protocol (RAP) [RHE99] uses an AIMD rate control scheme based on regular acknowledgments sent by the receiver which the sender uses to detect lost packets and estimate the RTT. The authors use the ratio of long-term and short-term averages of the RTT to fine-tune the sending rate on a per-packet basis. This translation from a window-based to a rate-based approach also includes a mechanism for the sender to stop sending in the absence of feedback from the receiver. Pure AIMD protocols like RAP do not account for the impact of retransmission timeouts, and hence we believe that TFRC will coexist better with TCP in the regime where the impact of timeouts is significant. Another AIMD protocol has been proposed in [SS98]. This protocol makes use of RTP [SCFJ96] reports from the receiver to estimate loss rate and round-trip times.

Equation-based congestion control [MF97] is probably the class of TCP-compatible unicast congestion control mechanisms most removed from the AIMD mechanisms of TCP. As already described in this paper, in unicast equation-based congestion control the sender uses an equation such as those proposed in [MF97, PFTK98] that specifies the allowed sending rate as a function of the RTT and packet drop rate, and adjusts its sending rate as a function of those measured parameters.

In [TZ99] the authors describe a simple equation-based congestion control mechanism for unicast, unreliable video traffic. The receiver measures the RTT and the loss rate over a fixed multiple of the RTT. The sender then uses this information, along with the version of the TCP response function from [MF97], to control the sending rate and the output rate of the associated MPEG encoder. The main focus of [TZ99] is not the congestion control mechanism itself, but the coupling between congestion control and error-resilient scalable video compression.

The TCP-Friendly Rate Control Protocol (TFRCP) [PKTK99] uses an equation-based congestion control mechanism for unicast traffic where the receiver acknowledges each packet. At fixed time intervals, the sender computes the loss rate observed during the previous interval and updates the sending rate using the TCP response function described in [PFTK98]. Since the protocol adjusts its send rate only at fixed time intervals, the transient response of the protocol is poor at lower time scales. In addition, computing loss rate at fixed time intervals make the protocol vulnerable to changes in RTT and sending rate. We have compared the performance TFRC against the TFRCP using simulations. With the metrics described in Section 3, we find TFRC to be better over a wide range of timescales.

TCP-Friendly multicast protocols have been proposed in [TPB, VRC98]. These scheme rely on data layering and use of multiple multicast groups. The congestion control mechanisms in these papers are specific to multicast, and are discussed briefly in Appendix C.2.

6 Issues for Multicast Congestion Control

Many aspects of unicast equation-based congestion control are suitable to form a basis for sender-based multicast congestion control. In particular, the mechanisms used by a receiver to estimate the packet drop rate and by the sender to adjust the sending rate should be directly applicable to multicast. However, a number of clear differences exist that require design changes and further evaluation.

Firstly, there is a need to limit feedback to the multicast sender to prevent response implosion. This requires either hierarchical aggregation of feedback or a mechanism that supresses feedback except from the receivers calculating the lowest transmission rate. Both of these add some delay to the feedback loop that may affect protocol dynamics.

Depending on the feedback mechanism, the slow-start mechanism for unicast may also be problematic for multicast as it requires timely feedback to safely terminate slowstart.

Finally, in the absence of synchronized clocks, it can be difficult for multicast receivers to determine their round-trip time to the sender in a rapid and scalable manner.

Addressing these issues will typically result in multicast congestion control schemes needing to be a little more conservative than unicast congestion control to ensure safe operation.

7 Conclusion and Open Issues

In this paper we have outlined a proposal for equation-based unicast congestion control for unreliable, rate-adaptive applications. We have evaluated the protocol extensively in simulations and in experiments, and have made both the *ns* implementation and the real-world implementation publically available [FHPW00]. We would like to encourage others to experiment with and evaluate the TFRC congestion control mechanisms, and to propose appropriate modifications.

The current implementations of the TFRC congestion control mechanisms (in *ns* and in the actual implementation) have an omission that we are planning to correct. The current congestion control mechanisms are designed for a sender that always has data available to send (until the last packet has been sent). When we began this work, our intention was to emulate the behavior of TCP as much as possible; however, there was no consensus on the appropriate response of TCP congestion control to a quiescent or application-limited period, where the previously-authorized congestion window or sending rate was not fully used. A proposal for modification of TCP congestion control to deal with a quiscent sender has been described in [HPF99]. Our plan is to implement a rate-based variant of this approach in TFRC. Our current simulations and experiments have also been with a one-way tranfer of data, and we plan to explore duplex TFRC traffic in the future.

While the current implementation of TFRC gives robust behavior in a wide range of environments, we certainly do not claim that this is the optimal set of mechanisms for unicast, equation-based congestion control. Active areas for further work include the mechanisms for the receiver's update of the packet drop rate estimate after a long period with no packet drops, and the sender's adjustment of the sending rate in response to short-term changes in the round-trip time. We assume that, as with TCP's congestion control mechanisms, equation-based congestion control mechanisms will continue to evolve based both on further research and on real-world experiences. As an example, we are interested in the potential of equation-based congestion control in an environment with Explicit Congestion Notification (ECN) [RF99].

We have run extensive simulations and experiments, reported in this paper and in other technical reports under preparation, comparing the performance of TFRC with that of standard TCP, with TCP with different parameters for AIMD's additive increase and multiplicative decrease, and with other proposals for unicast equation-based congestion control. In our results to date, TFRC compares very favorably with other congestion control mechanisms for applications that would prefer a smoother sending rate than that of TCP. There have also been proposals for increase/decrease congestion control mechanisms that reduce the sending rate in response to each loss event, but that do not use AIMD; we would like to compare TFRC with these congestion control mechanisms as well. We believe that the emergence of congestion control mechanisms for relatively-smooth congestion control for unicast traffic can play a key role in preventing the degradation of end-to-end congestion control in the public Internet, by providing a viable alternative for unicast multimedia flows that would otherwise be tempted to avoid end-to-end congestion control altogether [FF99].

Our view is that equation-based congestion control is also of considerable potential importance apart from its role in unicast congestion control. In our view, equation-based congestion control provides the foundation for scalable congestion control for multicast protocols. In particular, because AIMD and related increase/decrease congestion control mechanisms require that the sender decrease its sending rate in response to each packet drop, these congestion control families do not provide promising building blocks for scalable multicast congestion control. Our hope is that, in contributing to a more solid understanding of equation-based congestion control for unicast traffic, the paper contributes to a more solid development of multicast congestion control as well.

8 Acknowledgements

We would like to acknowledge feedback and discussions on equation-based congestion control with a wide range of people, including the members of the Reliable Multicast Research Group, the Reliable Multicast Transport Working Group, and the End-to-End Research Group, along with Dan Tan and Avideh Zhakor. We also thank Jim Kurose, Brian Levin, Dan Rubenstein, and Scott Shenker for specific feedback on the paper. Sally Floyd and Mark Handley would like to thank ACIRI, and Jitendra Padhye would like to thank the Networks Research Group at UMass, for providing supportive environments for pursuing this work.

References

[BCC+98] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. "Recommendations on Queue Management and Congestion Avoidance in the Internet". RFC 2309, Informational, Apr. 1998.

[FF99] S. Floyd and K. Fall. "Promoting the Use of End-to-End Congestion Control in the Internet". *IEEE/ACM Transactions on Networking*, Aug. 1999.

[FHP00] S. Floyd, M. Handley, and J. Padhye. "A Comparison of Equation-Based and AIMD Congestion Control", February 2000. URL http://www.aciri.org/tfrc/.

[FHPW00] S. Floyd, M. Handley, J. Padhye, and J. Widmer. "Equation-Based Congestion Control for Unicast Applications", February 2000. URL http://www.aciri.org/tfrc/.

[FHPW00] S. Floyd, M. Handley, J. Padhye, and J. Widmer. "TFRC, Equation-Based Congestion Control for Unicast Applications: Simulation Scripts and Experimental Code", February 2000. URL http://www.aciri.org/tfrc/.

[Flo91] S. Floyd. "Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic". *ACM Computer Communication Review*, 21(5):30–47, Oct. 1991.

[Flo99] S. Floyd. "Congestion Control Principles", Oct. 1999. Internet draft draft-floyd-cong-00.txt, work-in-progress.

[HPF99] M. Handley, J. Padhye, and S. Floyd. "TCP Congestion Window Validation". Sep. 1999. UMass CMPSCI Technical Report 99-77.

[Jac88] V. Jacobson. "Congestion Avoidance and Control". *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 314–329, 1988. An updated version is available via ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z.

[Jai91] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley and Sons, 1991.

[JE96] S. Jacobs and A. Eleftheriadis. "Providing Video Services over Networks without Quality of Service Guarantees". In *World Wide Web Consortium Workshop on Real-Time Multimedia and the Web*, 1996.

[JRC87] R. Jain, K. Ramakrishnan, and D. Chiu. "Congestion Avoidance in Computer Networks with a Connectionless Network Layer". Tech. Rep. DEC-TR-506, Digital Equipment Corporation, August 1987.

[MF97] J. Mahdavi and S. Floyd. "TCP-friendly Unicast Rate-based Flow Control". Note sent to end2end-interest mailing list, Jan. 1997.

[MSMO97] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. "The macroscopic behavior of the TCP congestion avoidance algorithm". *Computer Communication Review*, 27(3), July 1997.

[OKM] T. Ott, J. Kemperman, and M. Mathis. "The stationary behavior of ideal TCP congestion avoidance". ftp://ftp.bellcore.com/pub/tjo/TCPwindow.ps.

[OR99] V. Ozdemir and I. Rhee. "TCP Emulation At the Receivers (TEAR)". Presentation at the RM meeting, November 1999.

[Pad00] J. Padhye, Mar. 2000. Ph.D. thesis, University of Massachusetts at Amherst.

[Pax97] V. Paxson. "Automated packet trace analysis of TCP implementations". In *Proceedings of SIGCOMM'97*, 1997.

[PFTK98] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. "Modeling TCP Throughput: A Simple Model and its Empirical Validation". *SIGCOMM Symposium on Communications Architectures and Protocols*, Aug. 1998.

[PKC96] K. Park, G. Kim, and M. Crovella. "On the Relationship between File Sizes, Transport Protocols and Self-Similar Network Tarffic". In *Proceedings of ICNP'96*, 1996.

[PKTK99] J. Padhye, J. Kurose, D. Towsley, and R. Koodli. "A Model Based TCP-Friendly Rate Control Protocol". In *Proceedings of NOSSDAV'99*, 1999.

[RF99] K. K. Ramakrishnan and Sally Floyd. "A Proposal to add Explicit Congestion Notification (ECN) to IP". Jan. 1999. RFC 2481.

[RHE99] R. Rejaie, M. Handley, and D. Estrin. "An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet". In *Proceedings of INFOCOMM 99*, 1999.

[Riz98] L. Rizzo. "Dummynet and Forward Error Correction". In *Proc. Freenix 98*, 1998.

[RMR] "Reliable Multicast Research Group". URL http://www.east.isi.edu/RMRG/.

[RR99] S. Ramesh and I. Rhee. "Issues in Model-based Flow Control". Technical Report 99-15, Department of Computer Science, North Carolina State University, 1999.

[SCFJ96] H. Schulzrinne, S. Casner, R. Fredrick, and V. Jacobson. "RTP: A Transport Protocl for Real-Time Applications". RFC 1889, Jan. 1996.

[SS98] D. Sisalem and H. Schulzrinne. "The Loss-Delay Based Adjustment Algorithm: A TCP-Friendly Adaption Scheme". In *Proceedings of NOSSDAV'98*, 1998.

[SZC90] S. Shenker, L. Zhang, and D.D. Clark. "Some Observations on the Dynamics of a Congestion Control Algorithm". *ACM Computer Communication Review*, pages 30–39, Oct. 1990.

[TCP] "The TCP-Friendly Web Page". URL http://www.psc.edu/networking/tcp_friendly.html.

[TPB] T. Turletti, S. Parisis, and J. Bolot. "Experiments with a Layered Transmission Scheme over the Internet". Technical report RR-3296, INRIA, France.

[TZ99] D. Tan and A. Zakhor. "Real-time Internet Video Using Error Resilient Scalable Compression and TCP-friendly Transport Protocol". *IEEE Transactions on Multimedia*, May 1999.

[VRC98] L. Vicisano, L. Rizzo, and J. Crowcroft. "TCP-like Congestion Control for Layered Multicast Data Transfer". In *Proceedings of INFOCOMM'98*, 1998.

[Wid00] J. Widmer. "Equation-based Congestion Control", Feb. 2000. Diploma Thesis, URL http://www.aciri.org/tfrc/.

[WTSW95] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson. "Self-Similarity through High Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level". In *Proceedings of SIGCOMM'95*, 1995.

A Analysis of TFRC

A.1 Upper bound on the increase rate

In this section we show that, given a fixed round-trip time and in the absence of history discounting, the TFRC mechanism increases its sending rate by at most 0.14 packets/RTT.

History discounting is a component of the full Average Loss Interval method that is invoked after the most recent loss interval is greater than twice the average loss interval, to smoothly discount the weight given to older loss intervals. In this section we show that with fixed round-trip times and the invocation of history discounting, the TFRC mechanism increases its sending rate by at most 0.28 packets/RTT.

For simplicity of analysis, in this section we assume that TFRC uses the deterministic version of the TCP response function [FF99] as the control equation, as follows:

$$T = \frac{\sqrt{1.5}}{(R\sqrt{p})}.$$

This gives the sending rate T in packets/sec as a function of the round-trip time R and loss event rate p. Thus, the allowed sending rate is at most

$$\sqrt{1.5}/\sqrt{p} \approx 1.2/\sqrt{p}$$

packets/RTT.

To explore the maximum increase rate for a TFRC flow with a fixed round-trip time, consider the simple case of a single TFRC flow with a round-trip time of R seconds, on a path with no competing traffic. Let A be the TFRC flow's average loss interval in packets, as calculated at the receiver. The reported loss event rate is $1/A$, and the allowed sending rate is $1.2\sqrt{A}$ pkts/RTT.

After a round-trip time with no packet drops, the receiver has received $1.2\sqrt{A}$ additional packets, and the most recent loss interval increases by $1.2\sqrt{A}$ packets. Let the most recent loss interval be weighted by weight w in calculating the average loss interval, for $0 \leq w \leq 1$ (with the weights expressed in normalized form so that the sum of the weights is one). For our TFRC implementation in the normal case, when history discounting is not invoked, $w = 1/6$. The calculated average loss interval increases from A to at most $A + w1.2\sqrt{A}$ packets. The allowed sending rate increases from $1.2\sqrt{A}$ to at most $1.2\sqrt{A + w1.2\sqrt{A}}$ packets/RTT.

Therefore, given a fixed round-trip time, the sending rate increases by at most δ_T packets/RTT, for

$$1.2\sqrt{A + w1.2\sqrt{A}} = 1.2\sqrt{A} + \delta_T.$$

This gives the following solution for δ_T:

$$\delta_T = 1.2\left(\sqrt{A + w1.2\sqrt{A}} - \sqrt{A}\right) \tag{5}$$

Solving this numerically for $w = 1/6$, as in TFRC without history discounting, this gives $\delta_T \approx 0.12$ for $A \geq 1$. Thus, given a fixed round-trip time, and without history discounting, the sending rate increases by at most 0.12 packets/RTT.

This analysis assumes TFRC uses the simple TCP control equation [FF99], but we have also numerically modeled the increase behavior using Equation 1. Due to slightly different constants in the equation, the upper bound now becomes 0.14 packets/RTT. With the simple equation the usual increase is close to the upper bound; with Equation 1 this is still the case for flows where the loss rate is less that about 5% but at higher loss rates the increase rate is significantly lower than this upper bound.

When history discounting is invoked, the relative weight for the most recent interval can be increased up to $w = 0.4$; this gives $\delta_T \approx 0.28$, giving an increase in the sending rate of at most 0.28 packets/RTT in that case.

As this section has shown, the increase rate at the TFRC sender is controlled by the mechanism for calculating the loss event rate at the TFRC receiver. If the average loss rate was calculated simply as the most recent loss interval, this would mean a weight w of 1, resulting in $\delta_T \approx 0.7$. Thus, even if all the weight was put on the most recent interval, TFRC would increase its sending rate by less than one packet/RTT, given a fixed measurement for the round-trip time.

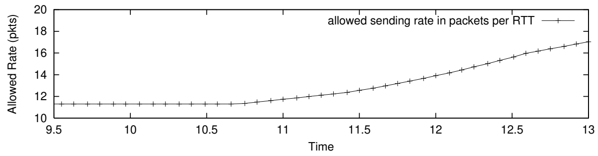


Figure 19: A TFRC flow with an end to congestion at time 10.0.

To informally verify the analysis above, we have run simulations exploring the increase in the sending rate for the actual TRFC protocol. Figure 19 shows a TFRC flow with every 100-th packet being dropped, from a simulation in the *ns* simulator. Then, after time 10.0, no more packets are dropped. Figure 19 shows the sending rate in packets per RTT; this simulation uses 1000-byte packets. As Figure 19 shows, the TFRC flow does not begin to increase its rate until time 10.75; at this time the current loss interval exceeds the average loss interval of 100 packets. Figure 19 shows that, starting at time 10.75, the sender increases its sending rate by 0.12 packets each RTT. Starting at time 11.5, the TFRC receiver invokes history discounting, in response to the detected discontinuity in the level of congestion, and the TFRC sender slowly changes its rate of increase, increasing its rate by up to 0.29 packets per RTT. The simulation in Figure 19 informally confirms the analysis in this section.

A.2 The lower bound on TFRC's response time for persistent congestion

This section uses both simulations and analysis to explore TFRC's response time for responding to persistent congestion. We consider the following question: for conditions with the slowest response to congestion, how many round-trip times n of persistent congestion are required before TFRC congestion control reduces its sending rate in half? For the simplified model in this section, we assume a fixed round-trip time; thus, we do not consider the effect of changes in round-trip time on the sending rate. We assume that, for an extended period, all loss intervals have been of length $1/p$ packets, for some loss event rate p. When congestion begins, we assume that at least one packet is successfully received by the receiver each round-trip time, and that the status reports transmitted each round-trip time by the receiver are successfully received by the sender. Thus, we are not considering the TFRC sender's mechanisms for reducing its sending rate in the absence of feedback from the receiver.

Given this model, assume that n round-trip times of persistent congestion are required before the TFRC sender reduces its sending rate by at least half. (That is, let n be a lower bound on the number of round-trip times of persistent congestion required before the TFRC sender reduces its sending rate by at least half.)

The control equation used in TFRC is nonlinear in $\sqrt{p}$ for higher values of p. A higher pre-existing loss event rate results in a stronger response by the TFRC sender to an increase in the reported loss event rate. In order to explore the slowest possible response of the TFRC sender to congestion, we assume that we are in the region of the control equation where the sending rate is essentially proportional to $\frac{1}{\sqrt{p}}$, for loss event rate p. This is true in the region of small to moderate loss event rates.

In this model of fixed round-trip times, for the region of moderate congestion, if the sending rate is reduced at least in half, this can only have been caused by the loss event rate increasing by at least a factor of four, and therefore by the average loss interval decreasing to at most $1/4$-th of its previous value. We note that in an environment where the round-trip time increases with the onset of persistent congestion, the TFRC sender would decrease its sending rate more strongly in response to congestion.

For this model of fixed round-trip times, what is the most drastic possible reduction in the average loss interval in response to n small loss intervals from persistent congestion? The most drastic possible reduction, not in fact achievable in practice, would be when the small loss intervals were each of size 0. We consider a model where the average loss interval is computed as described in Section 3.3. After one small loss interval, the average loss interval calculated by the receiver is still at least

$$\frac{3 + 0.8 + 0.6 + 0.4 + 0.2}{6} \frac{1}{p} = \frac{5}{6p}.$$

After two small loss intervals, the average loss interval is at least $\frac{2}{3p}$. Similarly, after four small loss intervals the average loss interval is at least $\frac{1}{3p}$. That is, it is not possible for the average loss interval to have reduced by a factor $1/4$ over only four loss intervals. However, after five small loss intervals the lower bound on the average loss interval is $\frac{1.2}{6p} = \frac{1}{5p}$; thus, in this simple model, it is possible for the average loss interval to be reduced by a factor of four after five loss intervals. Thus, in this model with fixed round-trip times and mild congestion, it might be possible for the sending rate to be cut in half after five consecutive round-trip times of congestion, but it is not possible for the sending rate to be cut in half after four consecutive round-trip times of congestion.

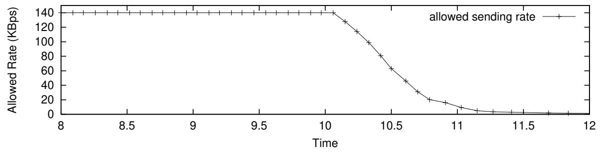


Figure 20: A TFRC flow with persistent congestion at time 10.

In fact this lower bound is close to the expected case. To informally verify this lower bound, which applies only to the simplified model described above with equal loss intervals before the onset of persistent congestion, we have run simulations exploring the decrease in the sending rate for the actual TRFC protocol. This is illustrated in the simulation shown in Figure 20 which consists of a single TFRC flow. From time 0 until time 10, every 100th packet dropped, and from time 10 on, every other packet is dropped. Figure 20 shows the TFRC flow's allowed sending rate as calculated at the sender every round-trip time, with a mark each round-trip time, when the sender receives a new report from the receiver and calculates a new sending rate. As Figure 20 shows, when persistent congestion begins at time 10, it takes five round-trip times for the sending rate of the TFRC flow to be reduced by half.

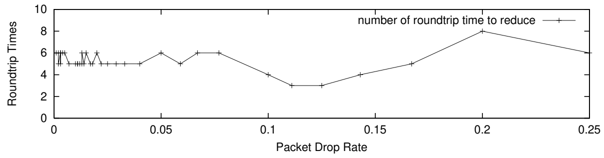


Figure 21: Number of round-trip times to reduce the sending rate in half.

Figure 21 plots the number of round-trip times of persistent congestion before the TFRC sender cuts its sending rate in half, using the same scenario as in Figure 20 with a range of values for the initial packet drop rate. For the TFRC simulations in Figure 21, the number of round-trip times required to reduce the sending rate by half ranges from three to eight. We note that for all of the simulations with lower packet drop rates, the TFRC sender takes at least five round-trip times to reduce its sending rate by half. Therefore, Figure 21 doesn't contradict the result earlier in this section.

This does not imply that the TFRC flow's response to congestion, for a TFRC flow with round-trip time R, is as disruptive to other traffic as that of a TCP flow with a round-trip time $5R$, five times larger. The TCP flow with a round-trip time of $5R$ seconds sends at an unreduced rate for the entire $5R$ seconds, while the TFRC flow reduces its sending rate, although somewhat mildly, after only R seconds.

A.3 The effect of increasing queueing delay

In this section we consider the effect of increasing queueing delay on the sending rate of a TFRC flow. In particular, we consider the sending rate of a single flow at the point when a queue has just begun to build at the congested link.

As described in Section A.1, given a fixed round-trip time, the TFRC sender increases its sending rate each round-trip time by δ_T packets/RTT, for δ_T given in Equation (5). In this section we show that, once queueing delay begins to build, the increase in queueing delay serves to inhibit this increase in the TFRC sending rate, and the TFRC sending rate stabilizes.

We show that this is similar to the role of the ACK-clock in limiting the sending rate of TCP. While TCP increases its congestion window by one packet per round-trip time when in congestion avoidance phase, this does not result in an unbounded increase in the sending rate. In particular, because of the role of the ACK-clock, a TCP sender in the congestion avoidance phase never sends more that one pkt/RTT above the receive rate for that flow.

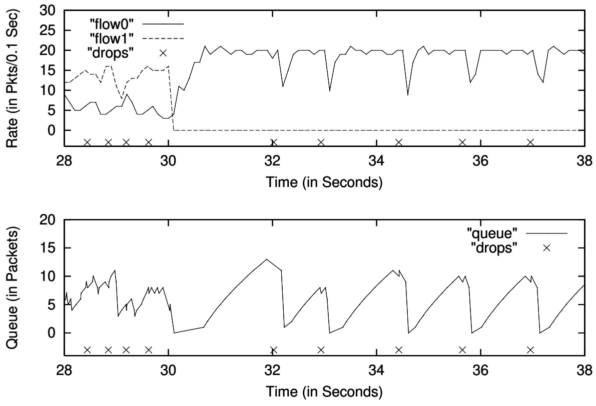


Figure 22: Two TCP flows on a single link.

Figure 22 shows a simulation with two TCP flows on a congested link, where flow 1 terminates at time 30. In the top graph, a line shows the number of packets transmitted by each flow over 0.1 sec intervals. There is an 'x' at the bottom of each graph for each packet drop; as the graph shows, there are no packet drops from time 30 to 32, but the sending rate of flow 0 never exceeds 21 packets/0.1 seconds. The bottom graph of Figure 22 shows the queue size in packets. The queue is using RED queue management.

For the simulation with TCP, TCP's ACK clock limits the sending rate of the TCP sender. That is, no matter how fast the sender is sending, the available bandwidth on the congested link limits the rate of data packets transmitted on the forward path, and therefore the rate of acknowledgements transmitted on the reverse path. The TCP sender in congestion avoidance sends one extra data packet each round-trip time, each time the congestion window is increased by one packet. Thus, when a TCP flow is the only active traffic on a path, and has achieved 100% throughput, as after time 30 in Figure 22, the TCP sender receives ACKs at exactly the rate of the bandwidth of the congested link in the forward path. Every round-trip time the TCP sender sends one packet above the rate allowed by the bandwidth of the congested link over the previous round-trip time. As a result, the queue at the congested link increases by one packet each round-trip time.

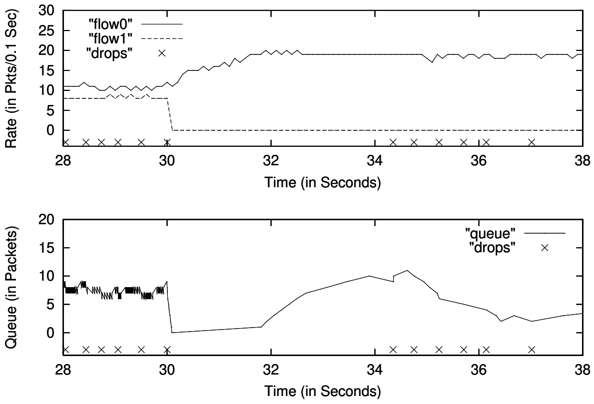


Figure 23: Two TFRC flows on a single link.

Figure 23 shows the same simulation with two TRFC flows. Although no packets are dropped from time 30 to time 34, the TFRC sending rate never exceeds 20 packets/0.1 second. For the TFRC flow in Figure 23, the sending rate increases from time 30 until time 31.6, when the queue begins to build at the congested link. At this point, the sending rate of the TFRC flattens out, even in the absence of new packet drops.[4] We show below that for TFRC, the slow increase in the measured round-trip time counterbalances the slow decrease in the reported packet drop rate, stabilizing the TFRC sending rate.

Let W be the delay-bandwidth product of the path, in packets, in the absence of queueing delay, and let R be the round-trip time. Then R/W is a single packet transmission time on the congested link. For simplicity, in this simple analysis we assume that the sender uses the instantaneous measurement of the round-trip time in calculating the allowed sending rate. Assume that at time t_0, the sender's sending rate has increased to exactly the link bandwidth. (This happens at time 31.6 in Figure 23.) At this time, the queue is empty, the measured RTT is R, and the sending rate is $\frac{W}{R}$ packets/sec. Let $1/A$ be the reported packet drop rate from the receiver at this time. That is, A is the average loss interval calculated at the receiver. Because at time t_0 the allowed

[4] In the TFRC code in NS, there is an upper bound on the TFRC sending rate that is either twice the receiver's reported receive rate, or one packet per round-trip time, whichever is larger. However, this upper bound is not reached in this simulation.

sending rate $\frac{1.2\sqrt{A}}{R}$ equals the link bandwidth $\frac{W}{R}$, it follows that

$$W = 1.2\sqrt{A}. \quad (6)$$

Recall that with a fixed round-trip time, the TFRC sender increases its sending rate each round-trip time by δ_T packets each round-trip time, for δ_T given in Equation (5). Assume that the queue increases by δ_T packets each round-trip time. After n round-trip times the queue has increased by $n\delta_T$ packets. This increases the round-trip time by $n\delta_T R/W$ seconds, increasing the most recent loss interval by $n1.2\sqrt{A} + \delta_T n(n-1)/2$ packets. As a result, the average loss interval calculated at the sender increases from A to roughly $A + w(n1.2\sqrt{A} + \delta_T n(n-1)/2)$ packets. The new allowed sending rate calculated at the TFRC sender after n round-trip times of a slow increase in the queue size is as follows:

$$\frac{1.2\sqrt{A + w(n1.2\sqrt{A} + \frac{n(n-1)}{2}\delta_T)}}{R + n\delta_T R/(1.2\sqrt{A})}.$$

Exploring this numerically, this allowed sending rate remains fairly constant as n ranges from 0 to 100. We used w is 1/6 and δ_T is 0.14, as described earlier in this section, and considered a wide range of values for A and R.

This section shows that as queueing delay starts to build, the decrease in the measured packet drop rate is balanced by the increase in queueing delay, and the TFRC sending rate stabilizes. For a simulation with multiple TFRC flows, there is a similar stabilization in the TFRC sending rate as a result of the increase in the measured round-trip time. We have not attempted to give a rigorous proof in this section, but have simply tried to lend insight into the stabilization of a TFRC flow's sending rate in response to the onset of queueing delay.

B Effect of TFRC on queue dynamics, extended version

In this section we continue an examination begin in Section 4.2 of the effects of TFRC on queue dynamics.

There are two significant differences between TCP-imposed and TFRC-imposed queue dynamics. TFRC responds more slowly and mildly to a single loss event than does TCP, and TFRC does not probe as aggressively for available bandwidth as does TCP. As a consequence, in an Drop-Tail environment queue busy periods with TCP traffic can be shorter but more frequent than with TFRC traffic.

Figure 24 shows results from two simulations with four long-lived flows, with start times spaced out over the first 20 seconds, with a congested link of 1.5 Mbps, and round-trip times of roughly 45 ms. The simulations include the same random background and reverse-path traffic as in the simulations in Section 4.2. long-range shows a simulation where all four long-lived flows are TCP, and the bottom graph shows a simulation where all four long-lived flows are TFRC. Both

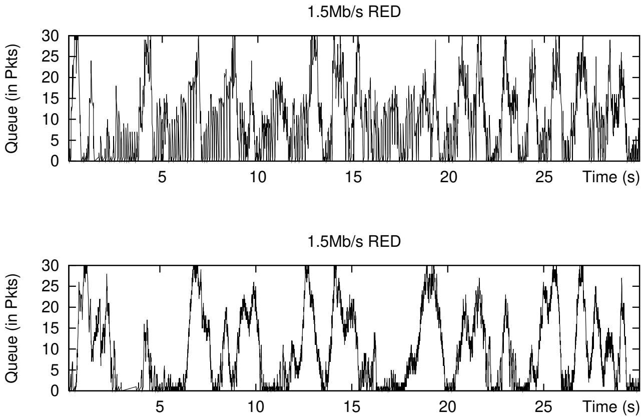


Figure 24: Four long-lived TCP (top) and TFRC (bottom) flows.

of these simulations use RED queue management. The first two lines of Table 25 show the link utilization (averaged over the 30-second simulation) and packet drop rates for these two simulations.

Queue	Traffic	Num. Flows	Link Util.	Drop Rate
RED	TCP	4	83	4.7
RED	TFRC	4	89	5.9
DropTail	TCP	4	89	5.2
DropTail	TFRC	4	96	4.9
RED	TCP	40	98	4.1
RED	TFRC	40	98	5.0
DropTail	TCP	40	99	4.9
DropTail	TFRC	40	99	3.5

Figure 25: Link utilization and packet drop rates.

As Figure 24 shows, the simulation with TFRC traffic has a slightly higher packet drop rate and link utilization than the simulation with TCP. Although we have not quantified it, it is clear that the simulation with TFRC traffic has lower-frequency oscillations of the instantaneous queue size, with longer busy periods, as we might expect with TFRC.

Figure 26 illustrates the queue dynamics with the same simulation scenario, but with Drop-Tail instead of RED queue management. For the simulations with Drop-Tail queue management, the TFRC simulation gives a higher link utilization and a lower drop rate than the TCP simulation. The third and fourth lines of Table 25 show the link utilization and packet drop rates for these two simulations. A chart of the average queueing delay would show a higher average queueing delay with DropTail than with RED queue management.

Queue dynamics can be expected to be considerably different with higher levels of statistical multiplexing. Figure 27 shows simulations with forty long-lived flows, starting over a 20-second interval, competing over a 15 Mbps link using

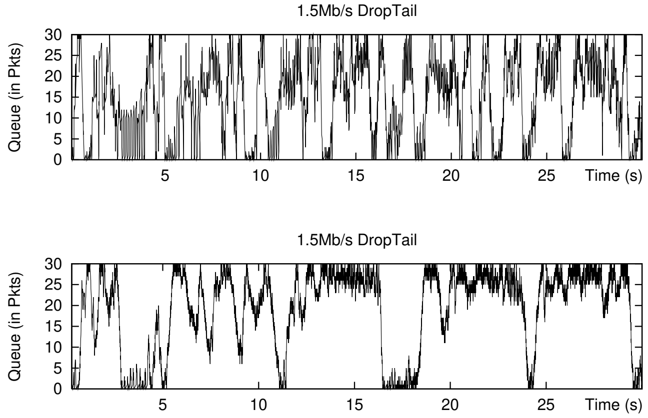


Figure 26: Four long-lived TCP (top) and TFRC (bottom) flows.

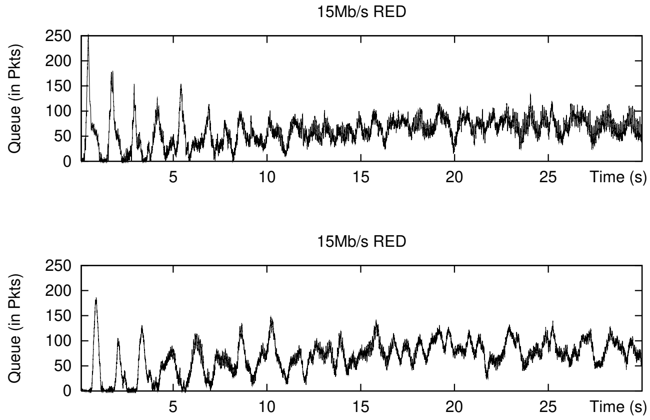


Figure 27: 40 long-lived TCP (top) and TFRC (bottom) flows.

RED queue management. The link bandwidth is ten times that in Figures 24 and 26, and there are also ten times as many flows, resulting in the same overall level of congestion but a higher level of statistical multiplexing than in Figures 24 and 26.

The bottom half of Table 25 shows the link utilization and packet drop rates for the simulations in Figures 27 and 14. Note that for the simulations with Drop-Tail queue management, the queue is nearly full for most of the simulations, whether with TCP or with TFRC traffic. In both cases, the packet drop rates are low, but the average queueing delay is much higher than with the simulations with RED queue management.

We note that it is possible to construct simulations with medium-scale statistical multiplexing with more pronounced oscillations in the queue size than those shown in Figures 27 and 14. For the simulations in Figures 27 and 14, 20% of the link bandwidth is used by short TCP flows, where for each flow the number of packets to transmit is randomly chosen between zero and twenty. Similarly, in these simulations there is some variation in the round-trip times for the long-lived connections. If we remove these elements of randomization, and look at a simulation with forty long-lived flows that start over the first twenty seconds, each with the exact same round-trip time, we find a more pronounced oscillations in the queue size.

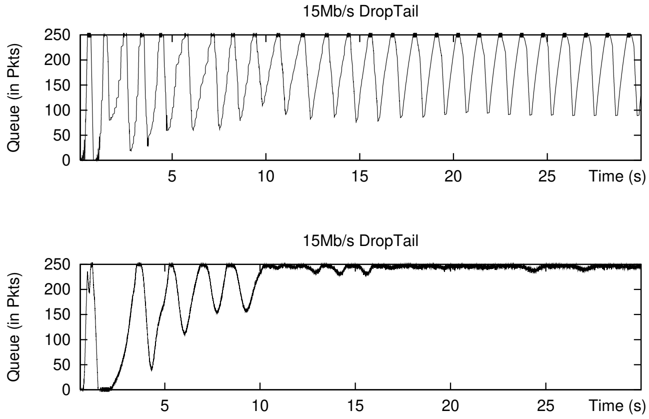


Figure 28: 40 long-lived TCP (top) and TFRC (bottom) flows, no randomization.

As an example, Figure 28 shows oscillations with both TCP and TFRC flows in simulations with no small TCP flows, no reverse-path background traffic, no variations in the round-trip times of the long-lived connections, and Drop-Tail queue management. This is a scenario designed to elicit fixed oscillations in queue size, and for the TCP traffic there are indeed standing oscillations in the queue size, though the queue does not go idle in this scenario. We note that neither TCP nor TFRC flows exhibited oscillations in this scenario with RED queue management. We also note that TFRC's delay-based congestion avoidance mechanism described in Section 3.4, which improves stability in the presence of short-time-scale changes in the round-trip time, is instrumental in preventing more pronounced oscillations with TFRC.

C More related work

C.1 Related work on TCP response functions

The simple TCP response function in [MF97] is as follows:

$$T = \frac{\sqrt{1.5}s}{R\sqrt{p}}. \tag{7}$$

This gives an upper bound on the sending rate T in bytes/sec, as a function of the packet size s, round-trip time R, and steady-state packet drop rate p. This version of the TCP response function is derived from a simple deterministic model, explored in [Flo91] and elsewhere, where the TCP connection receives regular deterministic packet drops. In this deterministic model, a packet is dropped each time the congestion window reaches W packets. The congestion window is multiplicatively decreased to $W/2$ in response to the packet drop, and then additively increased until it again reaches W. We note that this model does not take into account probabilistic drops. In addition, this simple model does not take into account the retransmit timeouts, or exponential backoffs of the retransmit timers, that are a key component of TCP's congestion control in the high-packet-drop-rate regime.

An earlier, more sophisticated derivation of the TCP response function from [MSMO97, OKM] analyzes an probabilistic AIMD-based model of TCP where each packet is dropped with a fixed probability p. Again, this probabilistic model does not take into account the role of TCP's retransmission timeouts.

In [MSMO97, PFTK98] the authors have shown that in many real-world TCP connections a large percentage of window reduction events are due to timeouts, and that the models in [MF97, MSMO97, OKM] overestimate the sending rate for packet loss rates greater than 5%. The TCP response function in Equation (1) is based on a model of TCP that takes into account the impact of retransmission timeouts [PFTK98]

Additional papers discussing the TCP response function can be found on the TCP-Friendly Web Page [TCP].

C.2 Related work on multicast congestion control mechanisms

This section discusses briefly some of the TCP-compatible congestion control mechanisms for multicast traffic.

The Loss-Delay based Adjustment (LDA) algorithm described in [SS98] applies AIMD directly to the sending rate rather than to a congestion window. The protocol in [SS98] relies on regular RTP/RTCP [SCFJ96] reports to estimate the loss rate and the RTT. An AIMD scheme based on these estimates is then used to control the sending rate. The receiver-based, multicast congestion control mechanism in [VRC98] uses data layering and multiple multicast groups to achieve a TCP-like AIMD effect. The sending rate of each layer is a multiple of sending rates of lower layers. Upon detecting losses, the receiver joins or leaves multicast groups to receive specific layers. The approach in [VRC98] uses periodic synchronization points for receivers to synchronize in the joining of additional layers.

The receiver-based, multicast congestion control mechanism described in [TPB] applies equation-based congestion control in an environment with data layering and multiple multicast groups. Each receiver estimates the packet loss rate and RTT, and uses a version of the TCP response function to compute the permitted reception rate. Based on this rate, the receiver decides which layers to receive by joining or leaving layered multicast groups.