Equation-Based Congestion Control for Unicast Applications*

Sally Floyd, Mark Handley
AT&T Center for Internet Research at ICSI (ACIRI)

Jitendra Padhye
University of Massachusetts at Amherst

Jörg Widmer
International Computer Science Institute (ICSI)

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.

We consider equation-based congestion control a promising avenue of development for congestion control of multicast traffic, and so an additional motivation for this work is to lay a sound basis for the further development of multicast congestion control.

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 [3]. While TCP congestion control is appropriate for applications such as bulk data transfer, some real-time applications (that is, 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 [22]. TCP's abrupt changes in the sending rate have been a significant 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 a viable mechanism to provide relatively smooth congestion control for such traffic.

Equation-based congestion control was proposed informally in [12]. 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 [8, 18, 17, 22, 15, 21], 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 mechanisms. The benefit of TFRC, relative to TCP, is a more smoothly-changing sending rate. The corresponding cost of TFRC is a more moderate response to transient changes in congestion, including a slower response to a sudden increase in the available bandwidth.

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 publicly 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

*This material is based upon work supported by AT&T, and by the National Science Foundation under grants NCR-9508274, ANI-9805185 and CDA-9502639. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of AT&T or the National Science Foundation.

May 2000. To appear in SIGCOMM 2000.

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 contribute to 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 a 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. Section 3 addresses these tradeoffs and describes the fundamental components of the TFRC algorithms that reconcile them.

Equation-based congestion control for multicast traffic has been an active area of research for several years [20]. A third goal of this work is to build a solid basis for the further 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. 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 congested FIFO queue [3]. From [2], 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.

From [14], 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 for both types of traffic only results if the two traffic types have similar response functions.

Some classes of traffic might not compete with TCP in a FIFO queue, but could instead be isolated from TCP traffic by some method (e.g., with per-flow scheduling, or in a separate differentiated services class from TCP traffic). In such traffic classes, applications using equation-based congestion control would not necessarily be restricted to the TCP response function 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 halve its sending rate 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 much current traffic, the alternatives have been between TCP, with its reduction of the sending rate in half in response to a single packet drop, and no congestion control at all. We believe that the development of congestion control mechanisms with smoother changes in the sending rate will increase incentives for applications to use end-to-end congestion control, thus contributing to the overall stability of the Internet.

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 [11]. Similarly, in Van Jacobson's 1992 revision of his 1988 paper on Congestion Avoidance and Control [9], 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 relative evaluation of AIMD and equation-based congestion control in [4] explores the benefits of 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 halve the sending rate in response to a single loss event. However, do halve the sending rate 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 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 function (Equation (1)) as the control equation for congestion control requires that 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 then 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.

3.1.1 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 then 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 loss event rate 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 response function from equation (1). 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 decreases the sending rate to T.

3.1.2 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 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 responding 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:

1. 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 3.2.1.
2. The estimated loss event rate should track relatively smoothly in an environment with a stable steady-state loss event rate.
3. The estimated loss event rate should respond strongly to loss events in several successive round-trip times.
4. The estimated loss event rate should increase only in response to a new loss event.
5. 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 Dynamic History Window method, the EWMA Loss Interval method, and the Average Loss Interval method which is the method we chose.

- The Dynamic History Window method uses a history window of packets, with the window length determined by the current transmission rate.
- The EWMA Loss Interval method uses an exponentially-weighted moving average of the number of packets between loss events.
- The Average Loss Interval method computes a weighted average of the loss rate over the last n loss intervals, with equal weights on each of the most recent $n/2$ intervals.

The Dynamic History Window method 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. In particular, the

Dynamic History Window method does not satisfy properties (2), (3), (4), and (5) above. The EWMA Loss Interval performs better than the Dynamic History Window method. However, it is difficult to choose an EWMA weight that responds sufficiently promptly to loss events in several successive round-trip times, and at the same time does not over-emphasize the most recent loss interval. The Average Loss Interval method satisfies properties (1)-(5) above, while giving equal weights to the most recent loss intervals.

We have compared the performance of the Average Loss Interval method with the EWMA and Dynamic History Window methods. In these simulation scenarios we set up the parameters for each method so that the response to an increase in packet loss is similarly fast. Under these circumstances it is clear that the Average Loss Interval method results in smoother throughput [13].

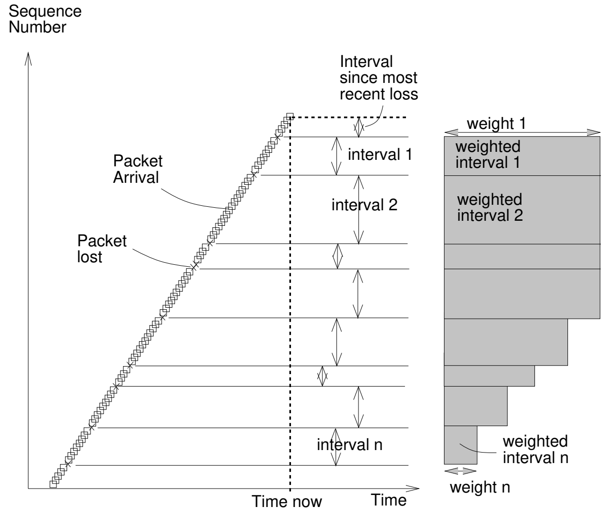

Figure 1: Weighted intervals between loss used to calculate loss probability.

The use of a weighted average by the Average Loss Interval method reduces sudden changes in the calculated rate that could result from unrepresentative loss intervals leaving the set of loss intervals used to calculate the loss rate. The average loss interval $\hat{s}_{(1,n)}$ is calculated as a weighted average of the last n intervals as follows:

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

for weights w_i:

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

and

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

For $n = 8$, this gives weights of 1, 1, 1, 1, 0.8, 0.6, 0.4, and 0.2 for w_1 through w_8, respectively.

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.

Any method for calculating the loss event rate over a number of loss intervals requires a mechanism to deal with the interval since the most recent loss event, as this interval is not necessarily a reflection of the underlying loss event rate. 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*. 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.

To determine whether to include s_0, the interval since the most recent loss, the Average Loss Interval method also calculates $\hat{s}_{(0,n-1)}$:

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

The final average loss interval $\hat{s}$ is $\max(\hat{s}_{(1,n)}, \hat{s}_{(0,n-1)})$, and the reported loss event rate is $1/\hat{s}$.

Because the Average Loss Interval method averages over a number of loss intervals, rather than over a number of packet arrivals, this method with the given fixed weights responds reasonably rapidly to a sudden increase in congestion, but is slow to respond to a sudden decrease in the loss rate represented by a large interval since the last loss event. To allow a more timely response to a sustained decrease in congestion, we deploy *history discounting* with the Average Loss Interval method, to allow the TFRC receiver to adapt the weights in the weighted average in the special case of a particularly long interval since the last dropped packet, to smoothly discount the weights given to older loss intervals. This allows a more timely response to a sudden absence in congestion. History discounting is described in detail in [5], and is only invoked by TFRC after the most recent loss interval s_0 is greater than twice the average loss interval. We have not yet explored the possibility of allowing more general adaptive weights in the weighted average.

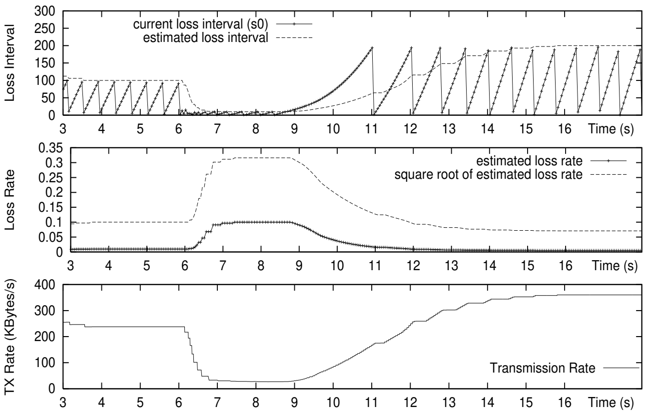

Figure 2: Illustration of the Average Loss Interval method with idealized periodic loss.

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. Because the losses in this simulation are perfectly periodic, the scenario is not realistic; it was chosen to illustrate the underlying properties of the Average Loss Interval method.

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. Finally, 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.

3.1.3 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 TCP 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. The response to changes in round-trip times is of particular concern in environments with Drop-Tail queue management and small-scale statistical multiplexing, where the round-trip time can vary significantly as a function of changes in a single flow's sending rate.

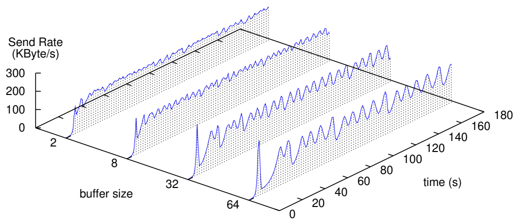

Figure 3: Oscillations of a TFRC flow over Dummynet, EWMA weight 0.05 for calculating the RTT.

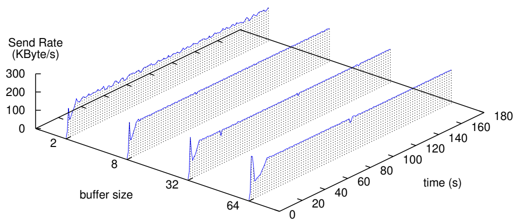

Figure 4: TFRC flow over Dummynet: oscillations prevented

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 RTT 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 Drop-Tail 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 [19]. 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 delayed and the sending rate is directly proportional to $1/R$, it is possible for short-term oscillations to occur, particularly with small-scale statistical multiplexing at Drop-Tail queues. While undesirable, the oscillations from large EWMA weights 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:

$$t_{inter-packet} = \frac{s}{T} \frac{\sqrt{R_0}}{M} \quad (2)$$

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. (The use of the square-root function in Equation (2) is not necessarily optimal; it is likely that other sub-linear functions would serve as well.) With this modification of the interpacket-spacing, 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, unlike the experiments in Figure 4.

3.1.4 Slowstart

TFRC's 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 for the receiver to 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:

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

This limits the slow-start overshoot to be no worse than TCP's overshoot on slow-start.

When a loss occurs causing 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 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 as it becomes available.

3.2 Discussion of Protocol Features

3.2.1 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.

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.

In [5] we explore the difference between the loss-event fraction and the regular loss fraction in the presence of random packet loss. We show that for a stable steady-state packet loss rate, and a flow sending within a factor of two of the rate allowed by the TCP response function, the difference between the loss-event fraction and the loss fraction is at most 10%.

Where routers use RED queue management, multiple packet drops in a window of data are not very common, but with Drop-Tail queue management it is common for multiple packets to be lost when the queue overflows. This can result in a significant difference between the loss fraction and the loss event fraction of a flow, and it is this difference that requires us to use the loss event fraction so as to better model TCP's behavior under these circumstances.

A transient period of severe congestion can also result in multiple packets dropped from a window of data for a number of round-trip times, again resulting in a significant difference between the loss fraction and the loss event fraction during that transient period. In such cases TFRC will react more slowly using the loss event fraction, because the loss event fraction is significantly smaller than the loss fraction. However, this difference between the loss fraction and the loss event fraction dimishes if the congestion persists, as TFRC's rate decreases rapidly towards one packet per RTT.

3.2.2 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 event rate provides sufficient damping, and there is little need to explicitly bound the increase in the transmission rate. As shown in Appendix A.1, given a fixed RTT and no history discounting, TFRC's increase in the transmission rate is limited to about 0.14 packets per RTT every RTT. After an extended absence of congestion, history discounting begins, and TFRC begins to increase its sending rate by up to 0.22 packets per round-trip time.

An increase in transmission rate due to a decrease in measured loss can only result from the inclusion of new packets in the most recent 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 [12] for the control equation. This behavior has been confirmed in simulations with TFRC, and has also been numerically modeled for the TCP response function in Equation (1), giving similar results with low loss rates and giving lower increase rates in high loss-rate environments.

3.2.3 Response to persistent congestion

In order to be smoother than TCP, TFRC cannot reduce its sending rate as drastically as TCP in response to a single packet loss, and instead responds to the average loss rate. The result of this is that in the presence of persistent congestion, TFRC reacts more slowly than TCP. Simulations in Appendix A.2 and analysis in [5] indicate that TFRC requires from four to eight round-trip times to halve its sending rate in response to persistent congestion. However, as we noted above, TFRC's milder response to congestion is balanced by a considerably milder increase in the sending rate than that of TCP, of about 0.14 packets per round-trip time.

3.2.4 Response to quiescent senders

Like TCP, TFRC's mechanism for estimating network conditions is predicated on the assumption that the sender is sending data at the full rate permitted by congestion control. If a sender is application limited rather than network-limited, these estimates may no longer reflect the actual network conditions. Thus, when sufficient data becomes available again, the protocol may send it at a rate that is much too high for the network to handle, leading to high loss rates.

A remedy for this scenario for TCP is proposed in [7]. TFRC is well behaved with an application-limited sender, because a sender is never allowed to send data at more than twice the rate at which the receiver has received data in the previous round-trip time. Therefore, a sender that has been sending below its permissible rate can not more than double its sending rate.

If the sender stops sending data completely, the receiver will no longer send feedback reports. When this happens, the sender halves its permitted sending rate every two round trip times, preventing a large burst of data being sent when data again becomes available. We are investigating the option of reducing less aggressively after a quiescent period, and of using slow-start to more quickly recover the old sending rate.

4. EXPERIMENTAL EVALUATION

We have tested TFRC extensively across the public Internet, in the Dummynet network emulator [19], 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

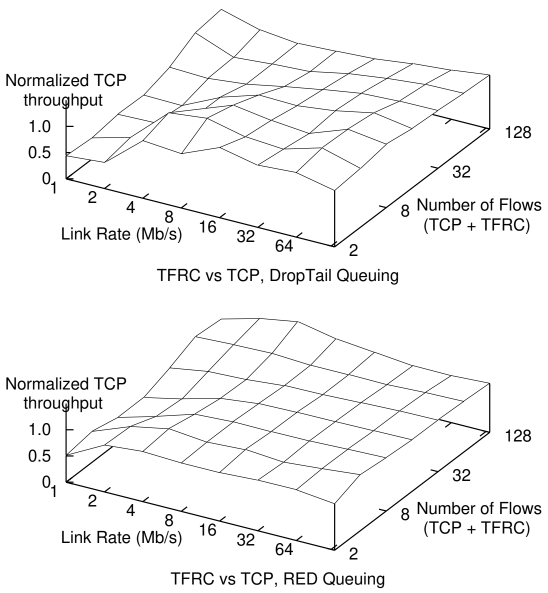

Figure 5: TCP flow sending rate while co-existing with TFRC

To demonstrate that it is feasible to widely deploy TFRC we need to demonstrate that TFRC 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 [13, 5] for more detailed results and simulation details, and to the code in the *ns* simulator [6].

Figure 5 illustrates the fairness of TFRC when competing with TCP Sack traffic in both Drop-Tail 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.

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. An open question that we have not yet investigated includes short- and medium-term fairness with TCP in an environment with abrupt changes in the level of congestion.

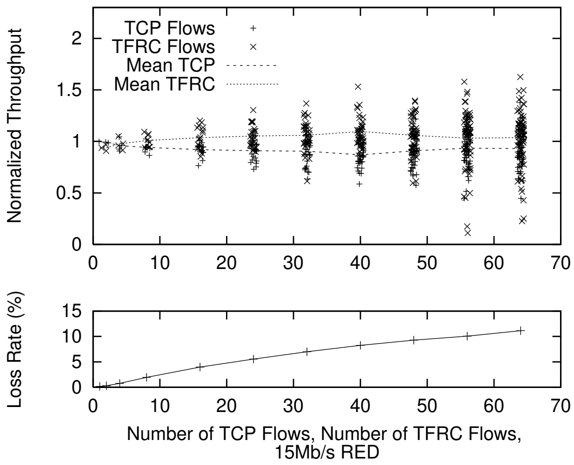

Figure 6: TCP competing with TRFC, with RED.

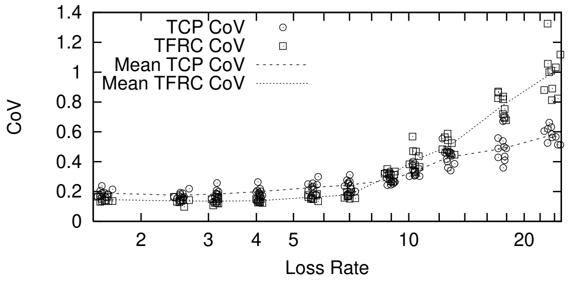

Figure 7: Coefficient of variation of throughput between flows

Although the mean throughput of the two protocols is rather similar, the variance can be quite high. This is illustrated in Figure 6 which shows the 15Mb/s data points from Figure 5. Each column represents the results of a single simulation, and each data point is the normalized mean throughput of a single flow. The variance of throughput between flows depends on the loss rate experienced. Figure 7 shows this by graphing the coefficient of variation (CoV[1]) between flows against the loss rate in simulations with 32 TCP and 32 TFRC flows as we scale the link bandwidth and buffering. In this case, we take the mean throughput of each individual flow over 15 seconds, and calculate the CoV of these means. The results of ten simulation runs for each set of parameters are shown. The conclusion is that on medium timescales and typical network loss rates (less than about 9%), the inter-flow fairness of individual TFRC flows is better than that of TCP flows. However, in heavily overloaded network conditions, although the mean TFRC throughput is similar to TCP, TFRC flows show a greater variance between their throughput than TCP flows do.

We have also looked at Tahoe and Reno TCP implementations and at different values for TCP's timer granularity. Although Sack TCP

[1]Coefficient of Variation is the standard deviation divided by the mean.

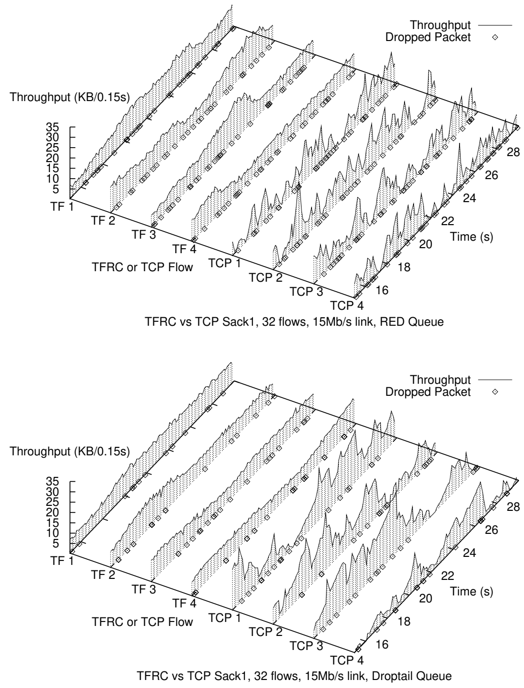

Figure 8: TFRC and TCP flows from Figure 5.

with relatively low timer granularity does better against TFRC than the alternatives, the performance of Tahoe and Reno TCP is still quite respectable.

Figure 8 shows the throughput for eight of the flows (four TCP, four TFRC) from Figure 5, 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 a plausible candidate for a minimum interval over which bandwidth variations would begin to be noticeable to multimedia users.

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 and variability 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.

We define the send rate $R_{\delta,F}(t)$ of a given data flow F using s-byte packets at time t, measured at a timescale δ:

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

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) of this time series is standard deviation divided by the mean, and can be used as a measure of variability [10] of the sending rate with an individual flow at timescale δ. For flows with the same average sending rate of one, the *coefficient of variation* would simply be the standard deviation of the time series; a lower value implies a flow with a range of sending rates more closely clustered around the average.

To compare the send rates of two flows a and b at a given time scale δ, we define the *equivalence* $e_{\delta,a,b}(t)$ at time t:

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

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

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 the 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 .

In [4] we also investigate the *smoothness* of TCP and TFRC by considering the change in the sending rate from one interval of length δ to the next. We show that TFRC is considerably smoother than TCP over small and moderate timescales.

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 many simulation parameters, but illustrate here a scenario with 16 SACK TCP and 16 TFRC flows, with a bottleneck bandwidth of 15Mbps and a RED queue. To plot the graphs, we monitor the performance 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 7 has shown that for these low loss rates, TCP shows a greater variance in mean throughput that does 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

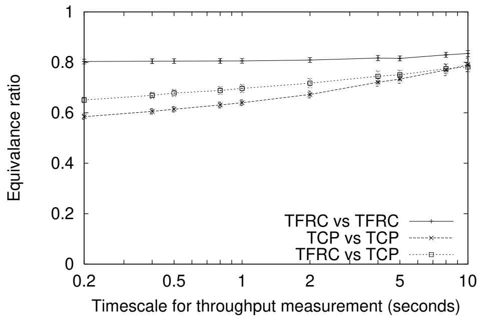

Figure 9: TCP and TFRC equivalence

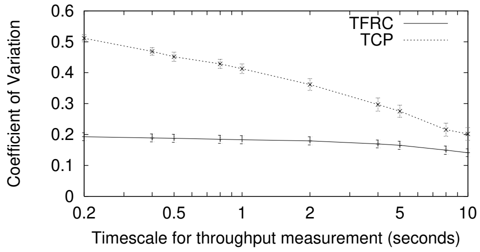

Figure 10: Coefficient of Variation of TCP and TFRC

of TFRC flows, and between pairs of flows of different types. The equivalence ratio of TCP and TFRC is between 0.6 and 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 less variable 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 this low-loss environment dominated by long-duration flows, the TFRC transmission rate is comparable to that of TCP, and is less variable 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 with very small TCP connections and some UDP flows. Figures 11-13 present results from simulations with background traffic provided by ON/OFF UDP flows with ON and OFF times drawn from a heavy-tailed distribution. The mean ON time is one second and the mean OFF time is two seconds, with each source sending at 500Kbps during an ON time. The number of simultaneous connections is varied between 50 and 150 and the simulation is run for 5000 seconds. 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. The results shown in Figures 12 and 13 are averages of ten runs.

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 on all but the longest timescales because packets are sent rarely. Any interval with only one flow sending a packet gives a value of zero in the equivalence time series, while intervals with neither flow sending a packet 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 gets 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 less variable 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 high loss rates and the variable nature of background traffic in these simulations.

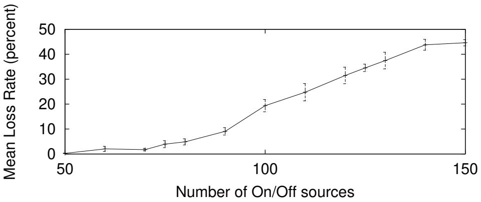

Figure 11: Loss rate at the bottleneck router, with ON-OFF background traffic

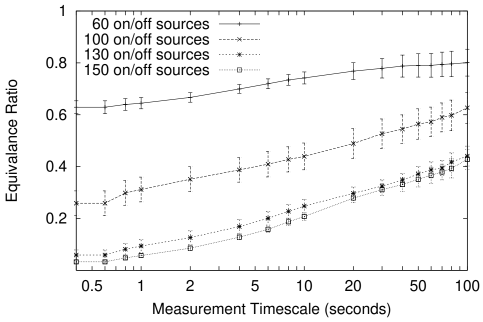

Figure 12: TCP equivalence with TFRC, with ON-OFF background traffic

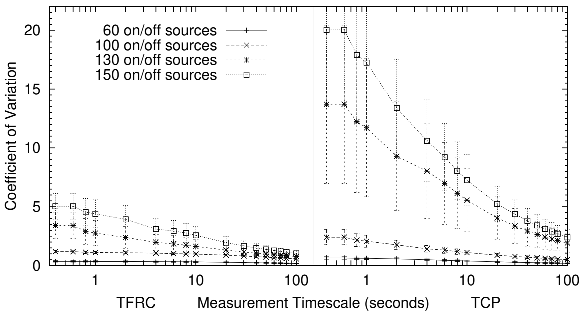

Figure 13: Coefficient of Variation of TFRC (left) and TCP (right), with ON-OFF background traffic

4.2 Effects of TFRC on Queue Dynamics

Because TFRC increases its sending rate more slowly than TCP, and responds more mildly to a 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 both similar to those of TCP, we expect some correspondence as well between the queueing dynamics imposed by TRFC and by TCP.

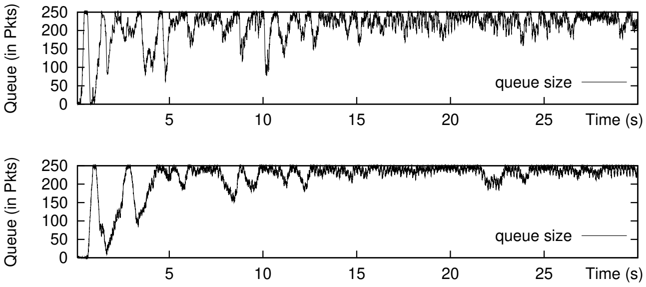

Figure 14: 40 long-lived TCP (top) and TFRC (bottom) flows, with Drop-Tail queue management.

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. 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 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 [5, Appendix B]. 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. In particular, in simulations using RED queue management we see little difference in queue dynamics imposed by TFRC and by TCP.

An open question includes the investigation of queue dynamics with traffic loads dominated by short TCP connections, and the duration of persistent congestion in queues given TFRC's longer time before halving the sending rate. As Appendix A.2 shows, TFRC takes roughly five round-trip times of persistent congestion to halve its sending rate. This does not necessarily imply that TFRC'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 following a loss, while the TFRC flow reduces its sending rate, although somewhat mildly, after only R seconds.

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 [23].

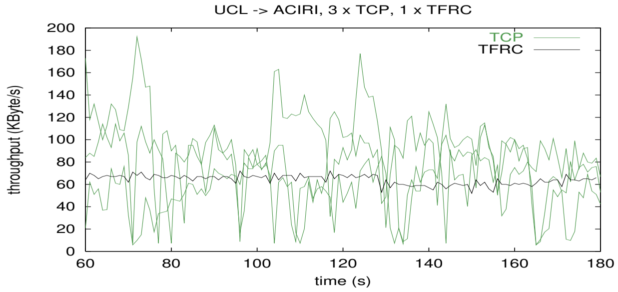

Figure 15: Three TCP flows and one TFRC flow over the Internet.

To summarize 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.

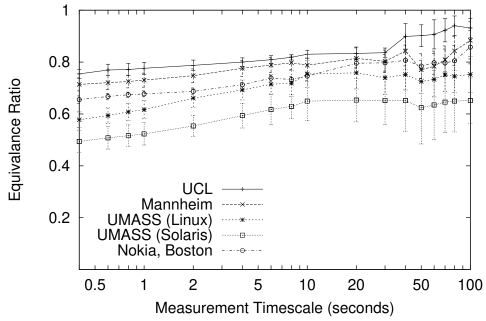

Figure 16: TCP equivalence with TFRC over different Internet paths

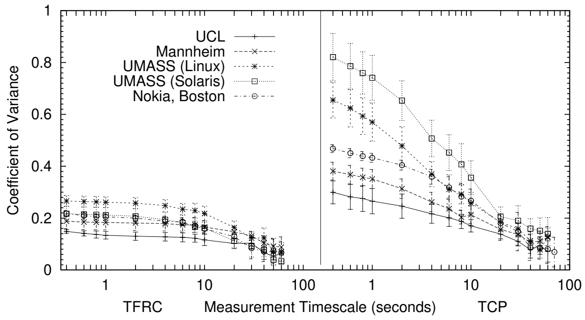

Figure 17: Coefficient of Variation of TFRC (left) and TCP (right) over different Internet paths

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 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 TFRC 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. This is discussed further in [5].

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 retransmission 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.

The effects of buggy TCP implementations can be seen in experiments from UMass to California, which 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 [16]. 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.

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). 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 [23].

4.4 Testing the Loss Predictor

As described in Section 3.1.2, 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 TFRC's mechanism for 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 3.1.2 are reasonable.

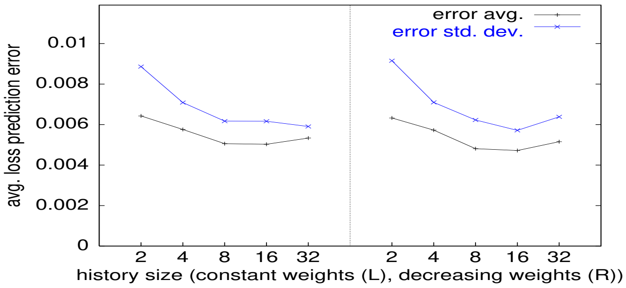

Figure 18: Prediction quality of TFRC loss estimation

5. SUMMARY OF RELATED WORK

The unreliable, unicast congestion control mechanisms closest to TCP maintain a congestion window which is used directly [8] or indirectly [18] to control the transmission of new packets. In [8] the sender uses TCP's congestion control mechanisms directly, and therefore its congestion control behavior should be similar to that of TCP. In the TEAR protocol (TCP Emulation at the Receivers) from [18], 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.

A class of unicast congestion control mechanisms one step removed from those of TCP are rate-based mechanisms using AIMD. The Rate Adaptation Protocol (RAP) [17] 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. RAP uses 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. An AIMD protocol proposed in [21] uses RTP reports from the receiver to estimate loss rate and round-trip times.

Bansal and Balakrishnan in [1] consider binomial congestion control algorithms, where a *binomial algorithm* uses a decrease in response to a loss event that is proportional to a power l of the current window, and otherwise uses an increase that is inversely proportional to the power k of the current window. AIMD congestion control is a special case of binomial congestion control that uses $l = 1$ and $k = 0$. [1] considers several binomial congestion control algorithms that are TCP-compatible and that avoid TCP's drastic reduction of the congestion window in response to a loss event.

Equation-based congestion control [12] is probably the class of unicast, TCP-compatible congestion control mechanisms most removed from the AIMD mechanisms of TCP. In [22] 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 [12], to control the sending rate and the output rate of the associated MPEG encoder. The main focus of [22] 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) [15] 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 [14]. 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 the loss rate at fixed time intervals make the protocol vulnerable to changes in RTT and sending rate. [13] compares the performance of TFRC and TFRCP, and finds that TFRC gives better performance over a wide range of timescales.

TCP-Friendly mechanisms for multicast congestion control are discussed briefly in [5].

6. ISSUES FOR MULTICAST CONGESTION CONTROL

Many aspects of TFRC are suitable to form a basis for sender-based multicast congestion control. In particular, the mechanisms used by a receiver to estimate the loss event rate and by the sender to adjust the sending rate should be directly applicable to multicast. However, a number of clear differences exist for multicast 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 suppresses 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, TFRC's slow-start mechanism may 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 publicly available [6]. We would like to encourage others to experiment with and evaluate the TFRC congestion control mechanisms, and to propose appropriate modifications.

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 loss event rate after a long period with no losses, 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). Similarly, our current simulations and experiments have been with a one-way transfer of data, and we plan to explore duplex TFRC traffic in the future.

We have run extensive simulations and experiments, reported in this paper and in [5], [4], [13], and [23], 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.

Our view is that equation-based congestion control is also of considerable potential importance apart from its role in unicast congestion control. Equation-based congestion control can provide 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 loss event, 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 Hari Balakrishnan, Jim Kurose, Brian Levin, Dan Rubenstein, Scott Shenker, and the anonymous referees 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.

9. REFERENCES

[1] D. Bansal and H. Balakrishnan. TCP-Friendly Congestion Control for Real-time Streaming Applications, May 2000. MIT Technical Report MIT-LCS-TR-806.

[2] 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.

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

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

[5] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based Congestion Control for Unicast Applications: the Extended Version, March 2000. ICSI Technical Report TR-00-03, URL http://www.aciri.org/tfrc/.

[6] 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/.

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

[8] 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.

[9] 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.

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

[11] 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.

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

[13] J. Padhye. Model-based Approach to TCP-friendly Congestion Control. Ph.D. thesis, University of Massachusetts at Amherst, Mar. 2000.

[14] 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.

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

[16] V. Paxson. Automated Packet Trace Analysis of TCP Implementations. In *Proceedings of SIGCOMM'97*, 1997.

[17] 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.

[18] I. Rhee, V. Ozdemir, and Y. Yi. TEAR: TCP Emulation at Receivers – Flow Control for Multimedia Streaming, Apr. 2000. NCSU Technical Report.

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

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

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

[22] 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.

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

APPENDIX

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 0.22 packets/RTT.

For simplicity of analysis, in this section we assume that TFRC uses the deterministic version of the TCP response function [3] 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 $1.2/\sqrt{p}$ packets/RTT.

To explore the maximum increase rate for a TFRC flow with a fixed round-trip time, consider the 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:

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

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 [3], 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, given TFRC's minimum discount factor of 0.5, the relative weight for the most recent interval can be increased up to $w = 0.29$; this gives $\delta_T \approx 0.22$, giving an increase in the sending rate of at most 0.22 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.

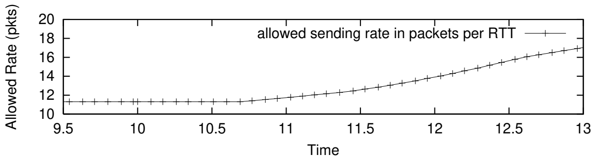

Figure 19: A TFRC flow with an end to congestion at time 10.0.

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 halves the sending rate? [5, Appendix A.2] shows that, given a model with fixed round-trip times and a control equation with the sending rate proportional to $\frac{1}{\sqrt{p}}$, at least five round-trip times of persistent congestion are required before TFRC halves the sending rate.

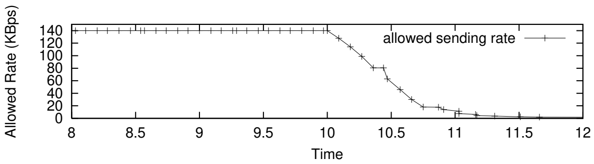

Figure 20: A TFRC flow with persistent congestion at time 10.

Simulations show that this lower bound of five round-trip times is close to the actual number of round-trip times of persistent congestion required for the sender to halve its sending rate. 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 halved.

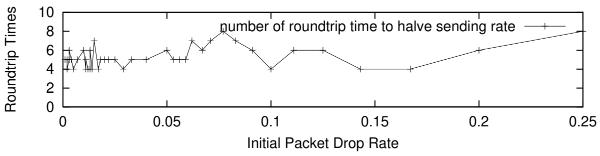

Figure 21: Number of round-trip times to halve the sending rate.

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 halve the sending rate ranges from four to eight round-trip times. For higher packet drop rates, the TFRC sender's control equation is nonlinear in $\frac{1}{\sqrt{p}}$, so it is not surprising that the lower bound of five round-trip times does not always apply.

We leave an upper bound on the number of round-trip times required to halve the sending rate as an open question. One possible scenario to investigate would be a scenario with a very large loss interval just before the onset of persistent congestion.