Dynamic Behavior of Slowly-Responsive Congestion Control Algorithms

Deepak Bansal and Hari Balakrishnan
MIT Laboratory for Computer Science
{bansal,hari}@lcs.mit.edu

Sally Floyd and Scott Shenker
AT&T Center for Internet Research at ICSI
{floyd,shenker}@aciri.org

Abstract

The recently developed notion of *TCP-compatibility* has led to a number of proposals for alternative congestion control algorithms whose long-term throughput as a function of a steady-state loss rate is similar to that of TCP. Motivated by the needs of some streaming and multicast applications, these algorithms seem poised to take the current TCP-dominated Internet to an Internet where many congestion control algorithms co-exist. An important characteristic of these alternative algorithms is that they are *slowly-responsive*, refraining from reacting as drastically as TCP to a single packet loss.

However, the TCP-compatibility criteria explored so far in the literature considers only the static condition of a fixed loss rate. This paper investigates the behavior of slowly-responsive, TCP-compatible congestion control algorithms under more realistic dynamic network conditions, addressing the fundamental question of whether these algorithms are safe to deploy in the public Internet. We study persistent loss rates, long- and short-term fairness properties, bottleneck link utilization, and smoothness of transmission rates.

1. Introduction

In the Internet's current congestion control paradigm, routers play a relatively passive role: they merely indicate congestion through packet drops or explicit congestion notification. It is the end-systems that perform the crucial role of responding appropriately to these congestion signals. This paradigm of passive routers and active hosts has been spectacularly successful; the congestion management mechanisms of TCP developed by Jacobson [10], based on the principles of packet conservation, slow-start, and additive-increase / multiplicative-decrease (AIMD) [3], is in large part responsible for the remarkable stability of the Internet despite rapid (to say the least) growth in traffic, topology, and applications.

Balakrishnan and Bansal were supported in part by an NSF CAREER Award, by DARPA Grant No. MDA972-99-1-0014, and by a research grant from the NTT Corporation. Bansal was also supported for a summer by ACIRI.

SIGCOMM'01, August 27-31, 2001, San Diego, California, USA.

An important property of the TCP congestion control algorithm is that similarly situated end-systems receive roughly equal bandwidths. TCP does not assure equality of bandwidth between end-systems with different round-trip times, or with multiple congested hops, or which use different packet sizes, but it does assure users that similarly situated flows using the same packet sizes will receive roughly the same bandwidth. We will call such bandwidth allocations *equitable* (to avoid the overloaded term *fair*) and it is the bandwidth allocation goal that we pursue in this paper.

Because routers don't exercise active control over bandwidth, the resulting bandwidth allocations are a function of the congestion control mechanisms used by the various end-systems. Before the advent of the TCP-compatible paradigm, which we describe below, the only way to reliably achieve equitable bandwidth allocations was for the end-systems to all use the same congestion control mechanism. Thus, for fairness reasons, TCP was seen not only as a *sufficient* condition but also as a *necessary* one.

The TCP congestion control mechanism produces rapidly varying transmission rates in the way it probes for spare capacity and reacts to congestion. While several classes of best-effort Internet traffic tolerate these variations quite well, other applications such as best-effort, unicast streaming video and audio are better served by congestion control mechanisms that respond more slowly to congestion and thereby produce a smoother bandwidth usage profile. The Internet community has struggled with this tension between the uniformity needed so that fairness can be obtained, and the desire to meet the demands of applications for whom TCP is a far-from-ideal solution. For multicast traffic, for example, TCP congestion control would be a particularly bad fit because it requires acknowledgements from all receivers in the multicast group.

A recently proposed resolution to this dilemma is the *TCP-compatible* paradigm. The cornerstone of this approach is the observation, made by a number of researchers [11, 13, 14], that one can characterize the bandwidth usage of a TCP flow in the presence of a constant packet loss rate p; to first order the bandwidth is proportional to $1/\sqrt{p}$. A congestion control mechanism is TCP-compatible if its bandwidth usage, in the presence of a constant loss rate, is the same as TCP [11]. The TCP-compatible paradigm simply transforms the requirement that all congestion control mechanisms be TCP into the looser requirement that all congestion control algorithms must be TCP-compatible.

This approach is a dramatic change from the earlier notions of congestion control. It could take us from an almost exclusively TCP-controlled world to one where there is no single dominant congestion control mechanism and instead there is a wide variety of mechanisms tailored to different application requirements. Already several alternative congestion control mechanisms have been

This is also known as *TCP-friendliness* [11].

proposed, including TCP-Friendly Rate Control (TFRC) [6] and other forms of equation-based congestion control, AIMD with different linear constants from TCP [20], binomial congestion control [2], and TCP Emulation at Receivers (TEAR) [17]. Unlike TCP, such mechanisms refrain from halving their congestion window (or transmission rate) in response to a single packet loss, and are more *slowly responsive* to packet loss events compared to TCP. These proposals are no mere academic exercises: the IETF has already adopted as Best Current Practice a document discussing and suggesting TCP-compatibility as a requirement for the standardization of new congestion control procedures for traffic likely to compete with best-effort TCP traffic [4]. In addition, the process of standardization of one mechanism for equation-based congestion control is already underway in the IETF, at the moment as an Internet-Draft [9].

Thus, we are possibly on the edge of a rather significant change in the set of congestion control mechanisms deployed on the Internet. However, this new approach is based on a condition—the TCP compatibility condition—that refers only to the behavior of the congestion control mechanism under *static* conditions. The Internet is clearly a very dynamic environment, and certainly static equivalence to TCP does not imply dynamic equivalence. The fundamental question we address here is: are these new congestion control algorithms safe to deploy in the current Internet? That is, even though they were developed with the static equivalence in mind, are they still TCP-compatible under more *dynamic* conditions?

We address two aspects of this question. First, we use simulation and analysis to evaluate the behavior of several TCP-compatible congestion control mechanisms under dynamic conditions; we focus on persistent packet loss rates, long- and short-term fairness properties, bottleneck link utilization, and smoothness of transmission rates. We find that most of the TCP-compatible algorithms we studied appear to be safe for deployment. While there are examples of algorithms that are TCP-compatible under static conditions but that exhibit unfortunate behavior in dynamic settings, the algorithms that have actually been proposed mostly avoided this problem. However, we find that two algorithms that are compatible under static conditions may not compete equitably under more dynamic conditions, even over long time-scales. In particular, while slowly-responsive TCP-compatible algorithms are safe to deploy in that they do not mistreat TCP, it is also true that they may not always get their equitable share when network conditions change dynamically.

This leads to the second question: Why? That is, what aspects of these algorithms are responsible for their remaining TCP-compatible under dynamic conditions? We find that incorporating the principle of packet conservation (e.g., by self-clocking transmissions as in TCP) is crucial in dynamic settings to ensure safety. The absence of long-term fairness despite static TCP-compatibility is caused by a fundamental trade-off: in return for smoother transmission rates, slowly-responsive algorithms lose throughput to faster ones (like TCP) under dynamic network conditions.

The rest of this paper describes these results. The next section is an overview of TCP-compatible algorithms. We describe the dynamic experiments and scenarios in Section 3, and present and explain our results in detail in Section 4. We conclude with a summary of our findings in Section 5. We discuss the role of timeouts in Appendix A

2. TCP-Compatible Congestion Control Algorithms

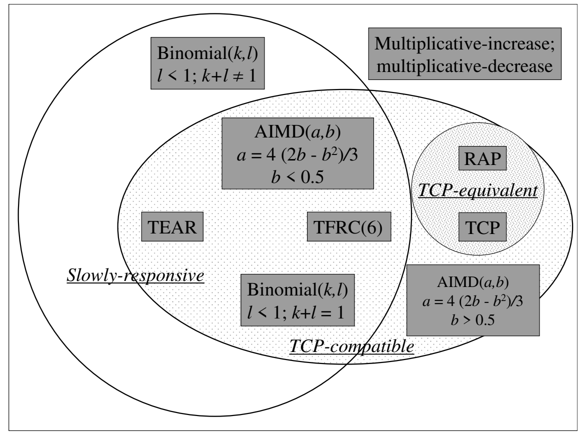

Figure 1: A classification of different end-to-end congestion control algorithms in relation to each other, with specific examples in rectangular boxes. This classification is based on a static notion, and our goal is to understand their behavior under dynamic conditions as well.

In steady-state, a long-running TCP connection uses two congestion control mechanisms: AIMD, which governs the size of the window, and self-clocking, which uses the principle of packet conservation to decide when the window must change and data transmitted. Proposals for end-to-end congestion control may be classified as *TCP-equivalent*, *TCP-compatible*, or *not TCP-compatible*, based on their *steady-state* behavior. Based on their *transient* response to congestion, end-to-end proposals can be classified as TCP-equivalent, *slowly-responsive*, or *responding faster than TCP*.

A congestion control algorithm is *TCP-equivalent* if it uses AIMD to govern its transmission window or rate, with the same increase and decrease parameters as TCP. Examples of TCP-equivalent schemes include various TCP variants, and rate-based schemes like Rejaie *et al.*'s Rate Adaptation Protocol (RAP) [16]. Because of the absence of self-clocking, TCP-equivalent schemes such as RAP can have different transient behavior than TCP, as we discover.

A congestion control mechanism is *TCP-compatible* in the static sense (or simply *TCP-compatible*) if it displays congestion control behavior that, on time scales of several round-trip times (RTTs), obtains roughly the same throughput as a TCP connection in steady-state when the available bandwidth does not change with time. Under conditions of an invariant packet loss rate p, the throughput of a TCP-compatible algorithm obeys the "TCP-friendly" formula giving the sending rate of a TCP sender under the same conditions. In this paper we use the TCP response function derived by Padhye *et al.* [14], observing that even for the static case, deriving a formula that correctly characterizes the sending rate of a particular TCP implementation or model across the entire range of values for p is a non-trivial chore [18]. It is easy to see that all TCP-equivalent schemes are also TCP-compatible, but not vice versa.

Not all TCP-compatible algorithms need to react in the same fashion as TCP on detecting congestion. A congestion control mechanism is said to be *slowly-responsive* (relative to TCP) if its window or rate reduction on a single packet loss or congestion notification is smaller than TCP. This slower response to individual packet drops allows applications using a slowly-responsive congestion control, or *SlowCC*, algorithm to benefit from a smoother sending rate than if they had used TCP's strategy. Examples of such algorithms include equation-based mechanisms such as TFRC (TCP-Friendly Rate Control) [6], AIMD-based mechanisms with different increase/decrease constants from TCP, and binomial congestion control mechanisms. A SlowCC algorithm may or may not be TCP-compatible, and conversely. Figure 1 summarizes the relationship between these different classes.

Two other key components of TCP's congestion control mechanisms that are not reflected in the above categories are the slow-start procedure and the exponential backoff of the retransmit timer. TCP's slow-start procedure is a key congestion control mechanism that is not used in steady-state, but is critical for transient behavior such as the initial start-up. The exponential backoff of the retransmit timer is critical in modeling TCP's behavior in environments with very high packet loss rates, in particular when a flow's average sending rate is less than one packet per round-trip time.

An AIMD-based algorithm is characterized by two parameters, a and b, corresponding to the increase and decrease parameters of the algorithm [8, 20]. After a loss event, the congestion window is decreased from W to $(1-b)W$ packets; in the absence of packet loss, the congestion window is increased from W to $W+a$ packets each RTT. TCP without delayed acknowledgments is an AIMD scheme with $a=1$ and $b=0.5$. For an AIMD scheme to be TCP-compatible, a and b are not independent—rather, $a=4(2b-b^2)/3$. Given an equation such as the one above for deriving a from b, a TCP-compatible AIMD algorithm is completely characterized by the parameter b; values of $b<0.5$ correspond to slowly-responsive AIMD algorithms. We use AIMD(b) to refer to a pure AIMD congestion control mechanism with parameter b, and we use TCP(b) to refer to TCP using AIMD(b) along with the other TCP mechanisms of slow-start, retransmit timeouts, and self-clocking.

Bansal and Balakrishnan consider binomial congestion control algorithms, which are a nonlinear generalization of AIMD [2]. These algorithms are characterized by four parameters, k, l, a, and b. Upon congestion, a binomial algorithm reduces its window (rate) from W to $W-bW^l$, while each RTT without congestion leads to a window (rate) increase from W to $W+a/w^k$. A binomial congestion control algorithm is TCP-compatible if and only if $k+l=1$ and $l\leq 1$, for suitable values of a and b. It is slowly-responsive for suitable values of a and b when $l<1$. The two specific binomial algorithms investigated in [2], IIAD ($k=1, l=0$) and SQRT ($k=l=0.5$), are both TCP-compatible and slowly-responsive. For binomial algorithms, smaller values of l tend to be more slowly-responsive than larger values.

Instead of responding in a fixed fashion to each loss or loss event, Floyd *et al.*'s TFRC responds to the loss event rate as measured over some interval of time [6]. In order to be TCP-compatible, TFRC uses the TCP response function characterizing TCP's sending rate as a function of the loss event rate and round-trip time. We let TFRC(k) denote a variant of TFRC that computes the average loss event rate over the most recent k loss intervals; the default TFRC suggested for deployment corresponds roughly to TFRC(6) [6, 9]. We investigate TFRC(k) for a range of values of k to understand better the limits on the viable parameters for these SlowCC mechanisms, and derive conditions for safe deployment in general.

Rhee *et al.*'s TCP Emulation at Receivers (TEAR) [17] is a receiver-based variant of TCP, where the receiver maintains an exponentially-weighted moving average of the TCP congestion window, and divides this by the estimated round-trip time to obtain a TCP-compatible sending rate. Thus, instead of changing TCP's algorithms for computing the congestion window, TEAR keeps TCP's congestion window algorithms unchanged and then averages the current window. TEAR is TCP-compatible and slowly-responsive under static conditions.

In addition to characterizing the sending rate of these SlowCC algorithms given a steady-state packet loss rate, there has been some exploration of the relative fairness of these mechanisms with TCP in the presence of ON-OFF background traffic and under different levels of statistical multiplexing [6, 19]. In addition, the relative *smoothness* of SlowCC proposals has been explored, with several different metrics used to measure smoothness [8, 19]. There has also been some investigation of the effect of SlowCC proposals on queue dynamics, including the effect on oscillations in the queue size, both with and without active queue management [7]. While there has been a preliminary evaluation of some SlowCC algorithms to measure *aggressiveness* and *responsiveness* [6, 8, 19], we are not aware of any systematic study of the impact of SlowCC mechanisms on competing traffic, or of this impact as a function of the time constants of the SlowCC mechanisms. These are important considerations for widespread deployment.

Our work systematically investigates the dynamics of TFRC, TCP, RAP, and the SQRT binomial algorithm under variable bandwidth conditions, using persistent loss rates, long- and short-term fairness, utilization, and smoothness as metrics.

3. Dynamic Test Scenarios

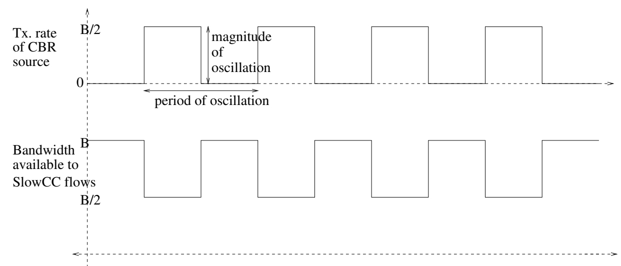

Figure 2: Square-wave oscillating bandwidth used in the simulations.

In this section, we describe the tests we performed to evaluate the behavior of the various TCP-compatible mechanisms in dynamic network conditions. We conducted our experiments with the ns-2 network simulator [12]. Our simulation scripts and results are available for download from `http://nms.lcs.mit.edu/slowcc/`.

The first set of tests in Section 4.1 considers the response of SlowCC mechanisms to a sudden increase in congestion; in the simulations the increase in congestion is triggered either by a competing CBR flow or by a flash crowd of many small TCP flows. We define the *stabilization time* as the time for the packet loss rate to stabilize after the start of a sustained period of high congestion, where the packet loss rate has stabilized if it is within 1.5 times its steady-state value at this level of congestion. The key concern is whether SlowCC mechanisms result in a transient period of high packet drop rates, both for themselves and for competing traffic, after a sudden increase in congestion. We pay particular attention to the cost of increased packet drops as a function of the slowness of the various SlowCC mechanisms. The underlying goal is to evaluate any potential dangers in the deployment of SlowCC mechanisms, since transient periods of high drop-rates could result in degraded network performance and very high response times.

In Section 4.2.1 we study the effect of changing network conditions on the long-term bandwidth fairness of SlowCC congestion control. To create dynamic network conditions, we use an ON/OFF CBR source with equal ON and OFF times, giving the repeating "square-wave" patterns of available bandwidth shown in Figure 2. Other simulation scenarios include "sawtooth" patterns of available bandwidth. These scenarios are motivated by concern for the behavior of SlowCC in dynamic environments with flash crowds and DoS attacks, routing changes, competition from higher-priority ON/OFF traffic, and the like. However, these traffic scenarios are not intended to accurately model reality, but to explore and benchmark the behavior of SlowCC mechanisms in a well-characterized environment. This can be thought of as a "stress test" that explores SlowCC congestion control in an extreme hostile environment. We are particularly interested in the relative bandwidth fairness of SlowCC mechanisms with TCP under these conditions, as a function of the magnitude and frequency of the oscillations in the available bandwidth.

To measure the transient fairness of SlowCC congestion control, in Section *4.2.2* we consider two flows using the same congestion control mechanism but starting with unequal shares of the link bandwidth, and consider the time until the two flows begin sharing the link bandwidth equitably. More formally, we define the *δ-fair convergence time* as the time taken by the two flows to go from a bandwidth allocation of $(B - b_0, b_0)$ to $(\frac{1+\delta}{2}B, \frac{1-\delta}{2}B)$, and we measure the average *δ-fair convergence time*. Here, b_0 is a small amount of bandwidth corresponding to 1 packet per RTT, and we assume $B >> b_0$.

Another concern with SlowCC congestion control mechanisms is that of a temporarily under-utilized link, resulting from the slowness of SlowCC mechanisms in taking advantage of a sudden increase in the available bandwidth. We study link utilization in this scenario in Section *4.2.3* using a new metric, $f(k)$. $f(k)$ is defined as the fraction of bandwidth achieved by a congestion control mechanism in the first k RTTs after the available bandwidth has doubled. In addition, we explore link utilization in a dynamic environment with rapid changes in the available bandwidth in Section *4.2.4*, where we study scenarios with a competing ON/OFF CBR source, as described earlier. Here, we consider link utilization as a function of the magnitude and frequency of the oscillations.

We are also interested in the benefits of SlowCCs, and in Section 4.3 we explore the relative smoothness of SlowCC mechanisms in a range of dynamic environments. The smoothness metric for TFRC has been defined as the largest ratio between the sending rates in two consecutive round-trip times. In Section 4.3 we consider smoothness over longer time intervals, without introducing any new metrics to quantify this.

The *responsiveness* of a congestion control mechanism has been defined as the number of round-trip times of persistent congestion until the sender halves its sending rate, where *persistent congestion* is defined as the loss of one packet per round-trip time [6]. The responsiveness of TCP is 1 round-trip time, and the responsiveness of the currently proposed TFRC schemes tends to vary between 4 and 6 round-trip times, depending on initial conditions [6]. One of the goals of this paper is to rigorously explore the impact of SlowCC congestion control mechanisms with a range of responsiveness measures. Thus, we explore TFRC(k) for k ranging from 1 to 256. Similarly, we explore TCP($1/b$) for b from 1 to 256. We define RAP($1/b$) and SQRT($1/b$) as the TCP-compatible instances of those congestion control mechanisms with multiplicative decrease factor b, and explore those mechanisms for a similar range for b. We note that, just as standard TCP is equivalent to TCP(1/2), standard RAP is equivalent to RAP(1/2). While RAP is TCP-equivalent, this is not true for RAP(b) for values of b other than 1/2.

All of our experiments use a single-bottleneck "dumbbell" topology with RED queue management at the bottleneck. Unless otherwise mentioned, the queue size is set to 2.5 times the bandwidth-delay product, and the min_thresh and max_thresh parameters are set to 0.25 and 1.25 times the bandwidth-delay product, respectively. The round-trip time (RTT) for the connections is approximately 50ms. Each simulation scenario includes data traffic flowing in both directions on the congested link.

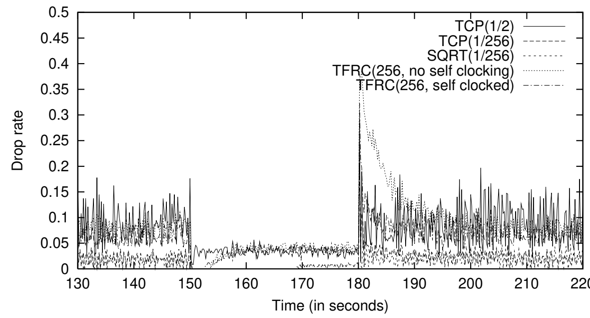

Figure 3: The drop rate for several SlowCC algorithms when a CBR source restarts at $t = 180$s after a 30s idle period.

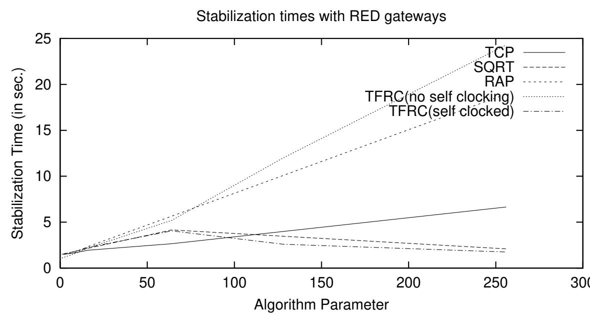

Figure 4: The stabilization time (in seconds, 1 RTT = 50ms) for the various SlowCC algorithms as a function of the algorithm's parameter, γ.

4. Results

In this section, we discuss the results of our experiments. We start by investigating the potential danger of slowly-responsive TCP-compatible algorithms in terms of increased packet loss rates, and discuss ways of reducing this danger. We then discuss two additional potential drawbacks of these algorithms: unfairness with respect to TCP and potentially sub-optimal bottleneck utilization. Finally, we discuss the benefits of these algorithms in terms of smooth transmission rates under dynamic network conditions.

4.1 "The Ugly": Potential Dangers of Slowly-Responsive Algorithms

By definition, SlowCC mechanisms respond slowly to reductions in the available bandwidth. As a consequence, despite being TCP-compatible under static conditions, a SlowCC mechanism could in fact cause high packet loss rates for extended periods of time. This is a grave concern, because persistently high drop rates result in an unnecessary decrease in throughput and an unnecessary increase in response times for the flows traversing the link.

4.1.1 A Competing CBR Source

Our first experiment investigates the performance of different TCP-compatible SlowCC algorithms when confronted with an

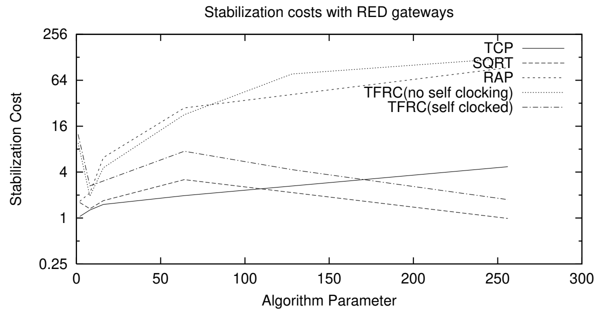

Figure 5: The stabilization cost for the various SlowCC algorithms as a function of the algorithm's parameter, γ.

abrupt reduction in the available bandwidth. We use twenty long-lived SlowCC flows, with changes in bandwidth being orchestrated by an ON/OFF CBR source that starts at $t = 0$s, stops at $t = 150$s, and restarts at $t = 180$s. When it is on, the CBR source uses one-half of the bandwidth of the bottleneck link. The bottleneck uses RED queue management and traffic sources as described in Section 3. During the $t = (0, 150)$s interval, we measure the average packet loss rate in the queue. Because the queue uses FIFO scheduling with RED queue management, all connections see similar loss rates. When the CBR source is idle between $t = (150, 180)$s, the packet drop rate is negligible. When the CBR source starts again at $t = 180$s, the network has a transient spike with a high packet drop rate, with a packet drop rate of roughly 40% for at least one round-trip time, until end-to-end congestion control can begin to take effect. The network then gradually returns to the same steady-state drop rate as during the $t = (0, 150)$s interval. Figure 3 shows the drop rate from several simulations using SlowCC mechanisms with *very* slow response times.

For each SlowCC algorithm, we define the *stabilization time* as the number of RTTs, after a period of high congestion begins, until the network loss rate diminishes to within 1.5 times its steady-state value for this level of congestion. In these simulations the period of high congestion begins at time 180, and the steady-state drop rate for that level of congestion is given by the drop rate over the first 150 seconds. Clearly the stabilization time will be different from one scenario to another; the purpose of the metric is to compare the stabilization times for different transport protocols in the same traffic scenario. We calculate the loss rate as an average over the previous ten RTT periods. Longer stabilization times indicate congestion control mechanisms with longer periods of congestion following a sudden decrease in the available bandwidth.

Figure 4 shows the stabilization time for the different SlowCC mechanisms. For each congestion control mechanism, the x-axis shows the parameter γ, corresponding to TCP($1/\gamma$), RAP($1/\gamma$), SQRT($1/\gamma$), and TFRC(γ). For example, the parameter $\gamma = 256$ corresponds to TCP(1/256) and TFRC(256) respectively. Note that TCP($1/\gamma$), RAP($1/\gamma$), SQRT($1/\gamma$), and TFRC(γ) are not necessarily an equivalent comparison for a specific value of γ. Figure 4 shows that there are extreme cases, notably TFRC(256) without self-clocking, where the stabilization time is hundreds of RTTs (about a half minute). TFRC without self-clocking is the default version of TFRC in ns-2.

While the stabilization time measures the amount of time it takes for the loss rate to return to near the previous value, the *stabilization cost* incorporates not only the time for stabilization, but also the average value of the loss rate during this stabilization period. More formally, we define the stabilization cost to be the product of the stabilization time and the average loss rate (in percentage) during the stabilization interval. The stabilization cost quantifies the true effects of persistent overload; a congestion control mechanism with a stabilization cost of 1 corresponds to an entire round-trip time worth of packets dropped at the congested link during the stabilization period, whether this is from a 100% packet drop rate for one round-trip time, a 50% drop rate for two round-trip times, or something else.

Figure 5 shows the stabilization cost for different SlowCC mechanisms, showing that, for large values of γ, some of them are *two orders of magnitude* worse than the most slowly-responsive TCP($1/\gamma$) or SQRT($1/\gamma$) algorithms we investigated. Note that the vertical axis is on a log-scale. Figure 5 also shows that the stabilization cost is acceptably low for SlowCC mechanisms with the range of parameters that have actually been proposed for use in the Internet.

Does Figure 5 indicate that SlowCC mechanisms with large values of γ, corresponding to very slow response (and stabilization) times, can cause persistent high packet loss rates and are therefore not safe for deployment in the Internet? It turns out that there is in fact a way to improve the stabilization cost of the RAP($1/\gamma$) and TFRC(γ) mechanisms with large values for γ.

To understand the difference between TFRC and RAP on the one hand, and TCP and SQRT on the other, it is worth asking what mechanisms are present in one class and not in the other. RAP($1/\gamma$) and TCP($1/\gamma$) are the closest of these algorithms in terms of the increase/decrease rules, with the main difference between them being the use of a rate variable rather than a window in RAP. The window-based TCP($1/\gamma$), unlike RAP($1/\gamma$), religiously follows the principle of packet conservation, being *self-clocked* by the arrival of acknowledgments from the sender. In contrast, RAP($1/\gamma$) and TFRC(γ) are rate-based; they transmit data based on the rate determined by the increase/decrease algorithm, irrespective of the number of acknowledgements received. Although they do use acknowledgements to update their sending rate, data transmissions themselves are not directly triggered by acknowledgments but instead are sent out based on the determined rate. The consequence of self-clocking is that TCP($1/\gamma$) and SQRT($1/\gamma$) reduce their transmission rates drastically when the available bandwidth drastically decreases, since acknowledgments start arriving only at the rate currently available to the flow at the bottleneck and the sending rate is therefore limited to the bottleneck (acknowledgment) rate from the previous RTT.

To evaluate whether self-clocking is in fact the key differentiator for the behavior of the very slow variants of these SlowCC mechanisms in this scenario, we added stronger self-clocking to the TFRC algorithm. TFRC already limits the sender's sending rate to at most *twice* the rate at which data is received by the receiver in the previous round trip [6]; this is critical to prevent severe over-shooting, and emulates TCP's slow-start phase. To incorporate stronger self-clocking in TFRC, we introduced a `conservative_` option to TFRC in ns-2 that, for the round-trip time following a packet loss, limits the sender's rate to at most the rate at which data is received by the receiver in the previous round trip (i.e., the RTT containing the loss). We call this *TFRC with self-clocking*.

In addition, for TFRC with self-clocking we need to limit the amount by which the sending rate can exceed the receive rate even in the absence of loss. Once the sending rate is reduced due to self-clocking, the absence of losses may cause TFRC to drastically increase its allowed sending rate (because of the memory of good times), once again violating self-clocking. Therefore, when not in slow-start, the `conservative_` option pegs TFRC's maximum

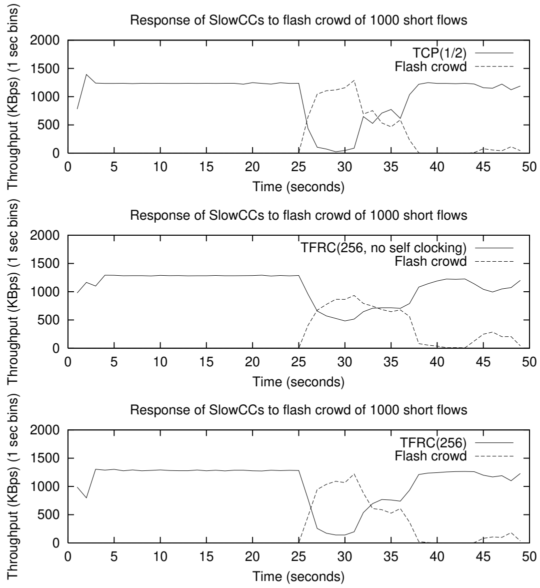

Figure 6: The aggregate throughput for long running SlowCC flows with a flash crowd of short TCP flows at time 25. Note that self-clocking helps TFRC(256) become quite responsive to the flash crowd.

sending rate to at most a constant C times the earlier receive rate.

The pseudo-code for this extension to TFRC is as follows:

```
CALCULATESENDRATE()
    /*
    SEND_RATE is the appropriate sending rate.
    CALC_RATE is what the equation allows.
    RECV_RATE is the reported receive rate.
    C ≥ 1 is a constant, 1.1 in our experiments.
    */
    if (loss is reported) then
        SEND_RATE = min(CALC_RATE, RECV_RATE)
    else if (NOT SLOWSTART) then
        SEND_RATE = min(CALC_RATE, C× RECV_RATE)
```

Thus, after a period of heavy losses in the network, the `conservative_` option causes TFRC's sending rate to immediately reduce to the reported receive rate. The results of TFRC(256) with self-clocking are shown in Figures 4 and 5. The improvement relative to the original TFRC(256) without self-clocking is apparent; the stabilization cost is also small as in TCP. These simulations were done with droptail queue management as well and a similar benefit of self-clocking was seen in those simulations also.

4.1.2 Competing Web Flash Crowd

We also experimented with a more realistic scenario where the dramatic reduction in bandwidth is caused by a flash crowd of small Web transfers rather than a new CBR source. The flash crowd is started at time 25 with a stream of short TCP transfers (10 packets) arriving at a rate of 200 flows/sec for 5 seconds. Figure 6 shows the aggregate throughput achieved by the small TCP connections and the aggregate throughput of the background SlowCC traffic for three different SlowCC traffic types, TCP(1/2), TFRC(256) without self clocking and TFRC(256) with self clocking. From this figure, the benefit of self-clocking in helping SlowCC respond to the flash crowd is clear. Because the flash crowd consists of many short flows in slow-start, the flash crowd grabs bandwidth quite rapidly regardless of whether the background traffic is TCP(1/2) or TFRC(256) (with self clocking).

Our conclusion is that it is possible for certain rate-based TCP-compatible algorithms to cause periods of persistently high loss rates under dynamic conditions. However, systematically applying the principle of packet conservation (e.g., by self-clocking transmissions) overcomes this problem even for the variants of these algorithms configured with very slow response times. Thus, while the possibility of periods of high packet loss rates is a significant concern in some cases, this concern can be eliminated by following the principle of packet conservation.

4.2 "The Bad": Potential Drawbacks of Slowly-Responsive Algorithms

We now turn our attention to two potential drawbacks of TCP-compatible SlowCC algorithms in highly variable environments: (i) unfairness with respect to TCP and each other, and (ii) potentially lower bottleneck link utilization. We study both long- and short-term fairness in dynamic environments.

4.2.1 Long-term Fairness

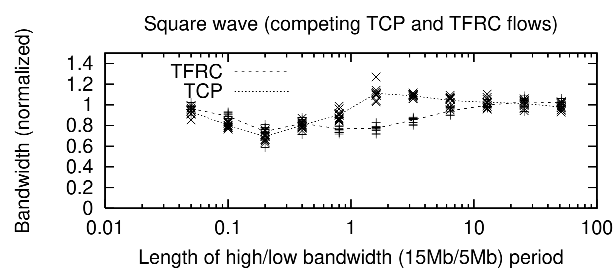

Figure 7: Throughput of TCP and TFRC flows when the available bandwidth changes by a 3:1 factor.

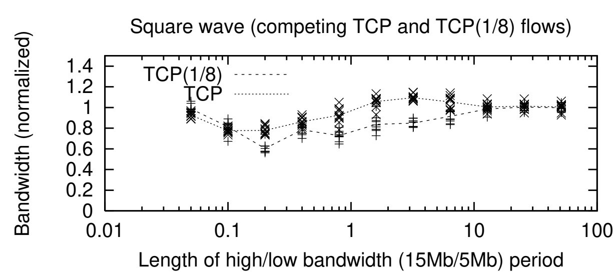

Figure 8: Throughput of TCP and TCP(1/8) flows when the available bandwidth changes by a 3:1 factor.

To investigate long-term fairness in a rapidly-changing environment, we consider a somewhat extreme scenario where the available bandwidth is periodically increased to three times its lower value. In this scenario, ten long-lived flows (five TCP and five TFRC) compete with a "square-wave" CBR source, using the topology described in Section 3. The congested link is 15 Mbps, with only 5 Mbps available to the long-lived flows when the CBR source is active. This gives a 3:1 variation in the bandwidth available to the long-lived flows. During an extended high-bandwidth period in this scenario, we would expect the packet drop rate with

We have experimented with various values of C and used $C = 1.1$ in the results reported here. The value in the NS simulator for TFRC's conservative_ option is $C = 1.5$.

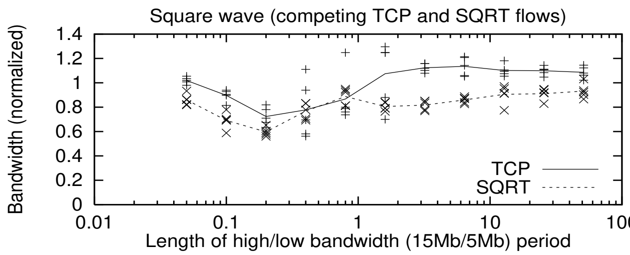

Figure 9: Throughput of TCP and SQRT(1/2) flows when the available bandwidth changes by a 3:1 factor.

ten long-lived flows to be roughly 0.7%, with an average TCP congestion window of 14.4 packets.

Our interest is in the relative fairness between TCP and TFRC as a function of the period of the CBR source. In Figure 7, each column of marks shows the results from a single simulation, with one mark giving the observed throughput for each of the ten flows. The x-axis shows the length in seconds of a combined high- and low-bandwidth period in that simulation, and the y-axis shows the throughput normalized by a single flow's fair share of the available bandwidth. The two lines show the average throughput received by the TCP and the TFRC flows.

As Figure 7 shows, overall link utilization is high when the period of the CBR source is low, while the overall link utilization suffers when the period of the CBR source is 0.2 seconds (4 RTTs). When the period of the CBR source is between one and three seconds, the TCP flows receive more throughput that the TFRC flows, showing that varying network conditions favor TCP over TFRC.

In an effort to find a scenario where TFRC might compete unfairly with TCP, we also ran simulations with a range of patterns for the competing CBR source, include "sawtooth" patterns with the CBR source slowly increased its sending rate and then abruptly entered an OFF period, or reverse sawtooth patterns where the CBR source abruptly entered an ON period and then slowly *decreased* its sending rate down to an OFF period. The results were essentially the same as in Figure 7, with the difference between TCP and TFRC less pronounced. These results demonstrate that there are many dynamic scenarios when TCP receives more bandwidth than competing TFRC flows. However, despite much trying, we could not find any scenarios with varying bandwidths in which TFRC receives more bandwidth than TCP in the long-term. Over short periods of time, immediately after a reduction in the available bandwidth, TFRC flows may get higher throughput than TCP flows, but in the long run, the TCP flows are more than competitive. Figures 8 and 9 show similar results when TCP competes with TCP(1/8) or with SQRT in this dynamic environment. Although not as agile as TCP, these SlowCC mechanisms are reasonably prompt in reducing their sending rate in responses to extreme congestion; however, they are observably slower at increasing their sending rate when the available bandwidth has increased. Our results suggest that there need be no concerns about unfair competition with TCP over long-term durations that would prevent SlowCC from being safely deployed in the current Internet.

We observed similar trends when competing algorithms were subjected to an even more extreme 10:1 oscillation in the available bandwidth—the throughput difference was significantly more prominent in this case. In a nutshell, SlowCC mechanisms lose to TCP under dynamic network conditions in the long run because their response to network conditions is slow; they do not send data fast enough when the bandwidth is actually available. Thus, two mechanisms that are TCP-compatible under static conditions do not necessarily compete equitably, even in the long term, in a more dynamic environment. In return for a smoother sending rate under more static conditions, SlowCC mechanisms pay the price of losing bandwidth, relative to TCP, in more dynamic environments.

4.2.2 Transient Fairness

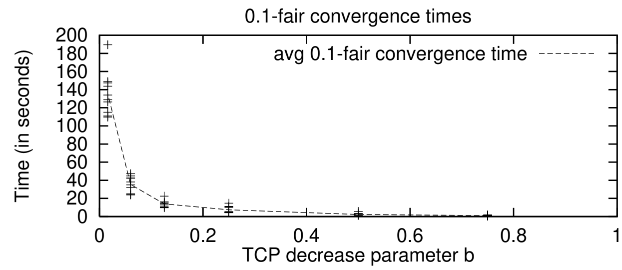

Figure 10: Time (in seconds) for convergence to 0.1-fairness for TCP(b) flows.

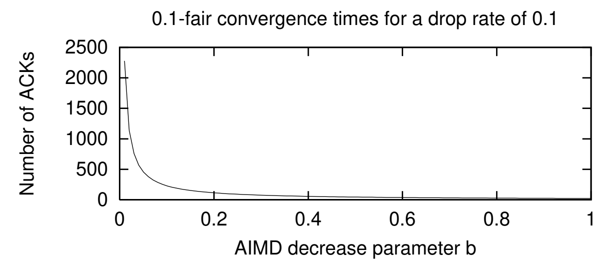

Figure 11: Number of ACKs for convergence to 0.1-fairness for TCP(b) flows.

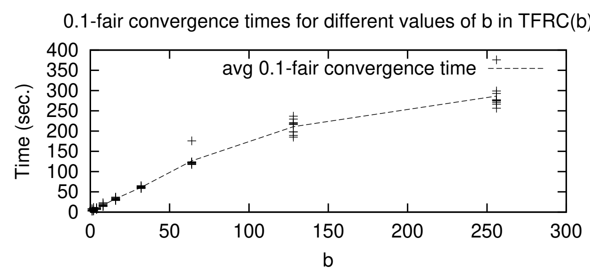

Figure 12: Time (in seconds) for convergence to 0.1-fairness for TFRC(b) flows.

We now consider the effect of SlowCC algorithms on transient fairness under dynamic conditions. We discuss the time for convergence to fairness for two flows using identical SlowCC mechanisms but starting at different sending rates. Transient fairness would be particularly important for short flows, whose entire lifetime might be contained in the transient period of convergence to fairness.

Figure 10 shows the results of simulations with two TCP(b) flows sharing a link of bandwidth B. Let (X_1, X_2) denote the bandwidths of the first and second flows respectively. We measure the *δ-fair convergence time*, defined in Section 3, for $delta = 0.1$. We use a value of the bottleneck bandwidth, $B = 10Mbps$, much bigger than b_0 which is the bandwidth corresponding to 1 packet/RTT (our RTT is 50 ms). Thus, the 0.1-fair convergence time being measured corresponds roughly to the time taken for an initial unfair allocation of $(B, 0)$ to converge to $(0.55B, 0.45B)$. Figure 10 shows the 0.1-fair convergence times for two TCP(b) flows for a range of values of b. If we decreased the link bandwidth, we would expect the convergence times to decrease accordingly.

We use an analytical model with the same framework to estimate the expected δ-fair convergence times for pure AIMD(a, b) flows in an environment with a steady-state packet mark rate p, when $B >> b_0$. (For simplicity of discussion assume that this is an environment with Explicit Congestion Notification (ECN) [15].) Let X_1^i and X_2^i denote the expected values of the congestion windows of the first and second flows after the arrival of the i-th ACK packet, and consider the effect of the $i+1$-th ACK packet. The $i+1$-th ACK belongs to flow 1 with probability $\frac{X_1^i}{X_1^i+X_2^i}$, and to flow 2 with probability $\frac{X_2^i}{X_1^i+X_2^i}$. After the $(i+1)^{th}$ ACK, the expected values of the two congestion windows become

$$X_1^i + \frac{X_1^i}{X_1^i + X_2^i}\left(\frac{a(1-p)}{X_1^i} - bpX_1^i\right)$$

and

$$X_2^i + \frac{X_2^i}{X_1^i + X_2^i}\left(\frac{a(1-p)}{X_2^i} - bpX_2^i\right)$$

respectively. The expected difference in the congestion windows of the two flows changes from

$$\rho_i = |X_1^i - X_2^i|$$

to

$$\rho_{i+1} = \left|X_1^i - X_2^i - bp\left(\frac{(X_1^i)^2}{X_1^i + X_2^i} - \frac{(X_2^i)^2}{X_1^i + X_2^i}\right)\right|$$

$$= \rho_i(1 - bp).$$

Thus the expected number of ACKs needed for a δ-fair allocation, starting from a highly skewed initial allocation, is essentially $log_{(1-bp)}\delta$.

Figure 11 shows the number of ACKs needed for a δ-fair allocation for various values of b for $\delta = 0.1$ and $p = 0.1$; other values of p give almost identically shaped curves. Note that the above analysis applies to TCP only for moderate to low loss probabilities, as it does not include retransmit timeouts or accurately model TCP's behavior when multiple packets are lost from a window of data.

Figure 11 shows that for values of $b >\approx 0.2$ and a drop rate of 10%, 0.1-fair convergence is achieved fairly rapidly, while for smaller values of b convergence takes exponentially longer. This suggests that for transient fairness, AIMD(b) for values of $b >\approx 0.2$ could give acceptable transient fairness, while significantly lower values for b would give unacceptably-long convergence times.

In Figure 12, we plot the 0.1-fair convergence times for TFRC(b) flows for different values of b. As this figure shows, the 0.1-fair convergence time does not increase as rapidly with increased slowness of TFRC flows. This can be explained by the fact that unlike multiplicative decrease in TCP, TFRC relies on a fixed number of loss intervals to adjust its sending rate to the available rate.

4.2.3 Loss in Throughput in a Time of Plenty

The slow increase rate of SlowCC can result in a loss of throughput, as compared to TCP, when there is a sudden increase in the bandwidth available to a flow. The *aggressiveness* of a congestion control mechanism has been defined as the maximum increase in the sending rate in one round-trip time, in packets per second, given the absence of congestion [8]. For TCP(a, b), the aggressiveness is simply the parameter a, while for TFRC the aggressiveness ranges from 0.14 to 0.28 packets/sec, depending on whether a TFRC option called history discounting has been invoked [7].

In this section we consider some of the implications of the low aggressiveness of SlowCC mechanisms in environments with a sudden increase in the available bandwidth. The fundamental underlying tradeoff in SlowCC mechanisms is that, in return for a smooth sending rate during times of steady-state conditions, SlowCC mechanisms are slow to take advantage of a sudden increase in the available bandwidth, relative to TCP. The slower the SlowCC mechanism, the more sluggish it will be in taking advantage of a sudden increase in the available bandwidth. This does not in any way interfere with competing traffic in the network, but we believe this sluggishness will be a compelling reason for applications not to use *extremely* slow congestion control mechanisms.

To make this concrete, we consider scenarios with long-lived flows where the bandwidth available to those flows is suddenly doubled. We define $f(k)$ as the average link utilization (expressed as a fraction) over the first k round-trip times after the bandwidth has doubled. This link utilization is a function not only of the congestion control mechanism and the number of round-trip times k, but also of the round-trip time and the link bandwidth in packets/sec.

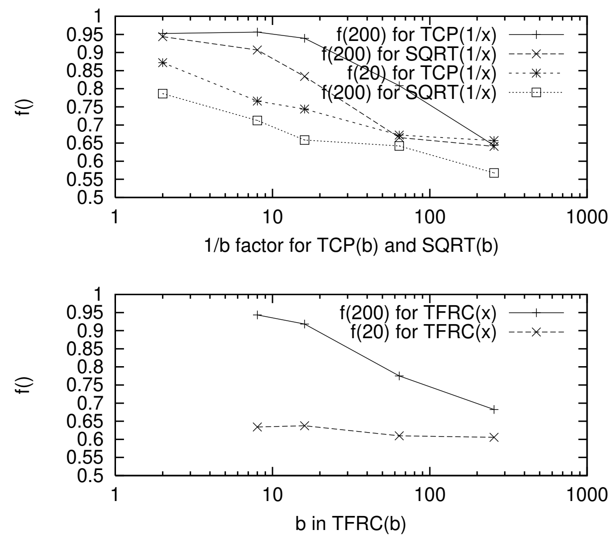

Figure 13: Average link utilization $f(20)$ and $f(200)$ for various SlowCCs, for a link bandwidth of 10 Mbps and a round-trip time of 50 ms.

To evaluate $f(k)$, we use a simulation scenario with ten identical flows, all using the same congestion control mechanism, sharing a bottleneck link of 10 Mbps. At time 500 sec., five flows are stopped, effectively doubling the bandwidth available to the remaining five flows. Figure 13 shows $f(20)$ and $f(200)$, the link utilization in the first 20 and 200 round-trip times, respectively, after the first five flows stopped, for TCP($1/b$), SQRT($1/b$), and TFRC(b) for a range of parameters b. Figure 13 shows that for this scenario, while TCP achieves about 86% utilization after the first 20 round-trip times, TCP(1/8) and TFRC(8) achieve 75% and 65% utilization respectively, showing the cost paid by SlowCC mechanisms in failing to make prompt use of the newly-available bandwidth.

Although slower congestion control mechanisms such as TCP($1/b$) or TFRC(b) for $b > 8$ have not been proposed for deployment, we investigate them to illustrate the extreme sluggishness of such mechanisms in reacting of an increase in the available bandwidth. TCP(1/256) and TFRC(256) both receive only 60% utilization after 20 round-trip times, and after 200 round-trip times have only increased the link utilization to 65-70%. We note that, for TFRC, these simulations use the TFRC implementation in the NS simulator with history discounting (a configurable option, turned on by default) turned off, and that this makes TFRC's performance somewhat worse than it would be otherwise. This allows us to focus solely on the part of TFRC that responds to packet loss rates and sets the transmission rate accordingly.

For a particular congestion control mechanism, $f(k)$ can be derived directly from the aggressiveness metric. Consider TCP(a, b) when the link bandwidth has been increased from λ to 2λ packets/sec, and let the RTT be Rs. After k round-trip times without congestion, TCP(a, b) will have increased its sending rate from λ to $\lambda + ka/R$ packets/sec, for an average sending rate of $\lambda + ka/(2R)$ packets/sec. Therefore, $f(k)$ can be approximated by $1/2 + ka/(4R\lambda)$ for TCP(a,b).

4.2.4 Loss in Throughput in a Time of Oscillations

Section 4.2.1 considered the relative long-term fairness between TCP and SlowCC in an environment with sharp changes in the available bandwidth, and Section 4.2.3 showed the penalty paid by SlowCC in being slow to take advantage of a sudden increase in the available bandwidth. In this section, we consider the overall link utilization in an environment of rapidly-changing available bandwidth when all of the flows use the same congestion control mechanism. We show that in such a dynamic environment, if all of the traffic consisted of long-lived flows using SlowCC, the overall link utilization can be somewhat lower than it would be with long-lived TCP flows in the same environment, depending on the nature of the changes in the available bandwidth. We do not present this as a reason not to deploy SlowCC, but as an exploration of the possible costs of SlowCC in extreme environments.

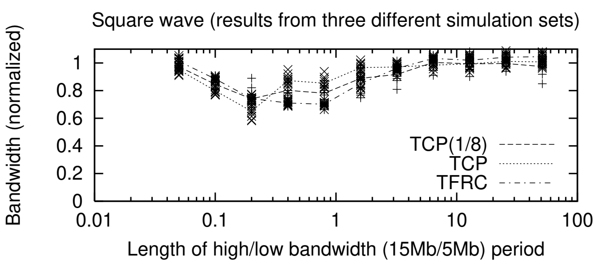

Figure 14: Effect of varying bandwidth on link utilization.

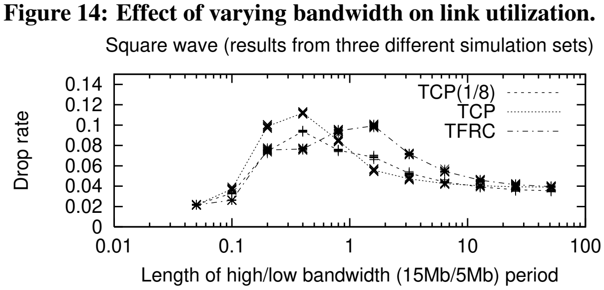

Figure 15: The corresponding packet loss rate.

To study this loss in throughput for SlowCC in an environment with changing network conditions, we use a simulation scenario with ten identical congestion-controlled flows competing with an ON/OFF CBR source. The bandwidth available to the congestion-controlled flows varies from 15 Mbps and 5 Mbps (i.e., a 3:1 ratio) as the CBR flow is OFF and ON respectively. We do not pretend that this is a realistic scenario; however, this simple scenario can provide insight into the dynamics of TCP and of SlowCC in an environment of changing network conditions.

Figure 14 shows the effect of the changing available bandwidth on the overall throughput. Three separate simulation sets were run, using TCP(1/8), TCP, and TFRC(6) respectively. The x-axis shows the length of the ON and the OFF periods for the competing CBR flow in seconds, and the y-axis shows the throughput of the congestion-controlled flows, as a fraction of the average available bandwidth. Each column shows the results of three separate simulations, using TCP(1/8), TCP, and TFRC(6). For each simulation, the graph shows the bandwidth of each flow (as a fraction of its bandwidth share), as well as the average bandwidth. Figure 15 shows the packet drop rate for the simulations in Figure 14.

As we can see from the Figure 14, the period of the competing CBR flow has a significant impact on the overall throughput of the congestion-controlled flows. For example, when the CBR flow has ON and OFF times of 50 ms, throughput is high for TCP(1/8), TCP, and for TFRC(6). This shows that short bursts of competing traffic are not harmful to TCP or to SlowCC, as these short bursts can be effectively accommodated by the active queue management at the congested router. In contrast, when the CBR flow has ON and OFF times of 200 ms, four times the round-trip time, a congestion-controlled flow receives less than 80% of the overall available bandwidth, whether the flow is using TCP(1/8), TFRC, or TCP.

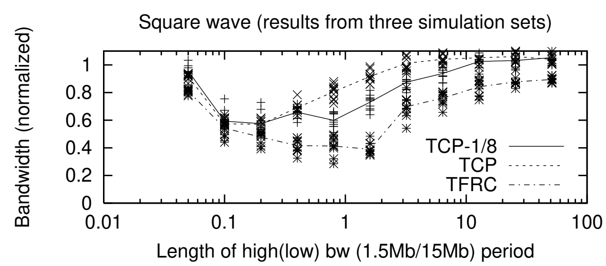

Figure 16: Effect of 10:1 oscillations in network bandwidth on bandwidth utilization of various congestion control algorithms.

Figure 16 shows that in a more extreme environment with repeated 10:1 changes in the available bandwidth, none of the three congestion control mechanisms are particularly successful, but for certain frequencies of change in the underlying bandwidth, TFRC performs particularly badly relative to TCP. This underlies the point that although TCP and SlowCC mechanisms might perform somewhat similarly in a steady-state environment, this is not necessarily the case in more extreme conditions with rapid changes. In particular, an environment with varying load may result in lower throughput (and hence, lower link utilization) with SlowCCs than with TCPs.

4.3 "The Good": Potential Benefits of Slowly-Responsive Algorithms

The main motivation for the development and deployment of SlowCC mechanisms has been that their sending rate is smoother than that of TCP in a steady-state environment with a reasonably smooth packet loss process. The *smoothness metric* is defined as the largest ratio between the sending rates in two consecutive round-trip times. In a steady-state environment with a periodic packet drop rate, TFRC has a perfect smoothness metric of 1, while TCP(b) congestion control has a smoothness metric of $1 - b$; congestion control mechanisms that reduce their window or rate in response to a single drop cannot have perfect smoothness [8]. The focus of this section is to consider the smoothness of SlowCC mechanisms in environments with bursty loss patterns. With such bursty packet loss patterns the relative smoothness of various TCP-compatible congestion controls become considerably more complex.

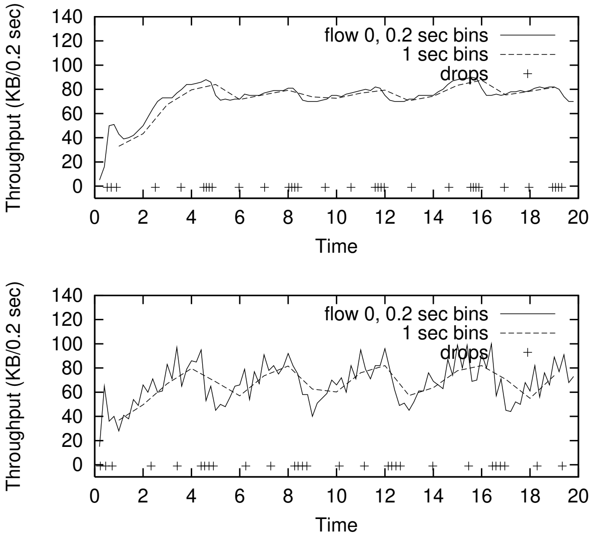

Figure 17: TFRC (top) and TCP(1/8) (bottom) with a mildly bursty loss pattern.

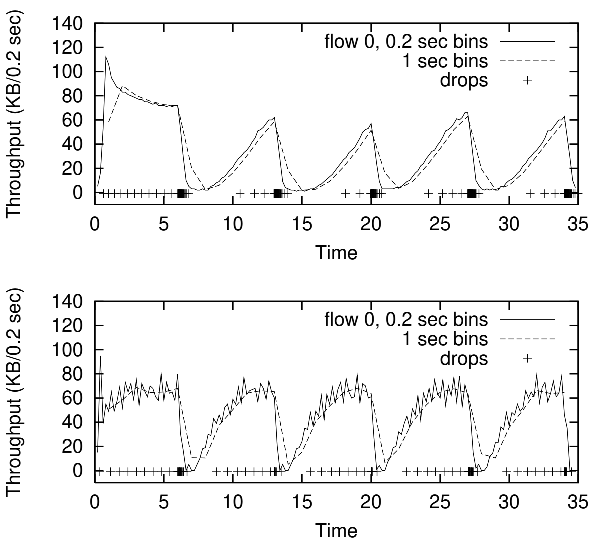

Figure 18: TFRC (top) and TCP(1/8) (bottom) with a more bursty loss pattern.

While the averaging of the loss rate in TFRC gives greater smoothness in steady-state or in mildly bursty conditions, this same averaging of the loss rate results in very poor smoothness from TFRC in other more bursty conditions that exploit TFRC's slowness to forget about past conditions. Figures 17 and 18 show TFRC and TCP(1/8) in two carefully designed scenarios intended to illustrate the best and the worst, respectively, in TFRC's smoothness in the face of bursty packet losses.

The simulations in Figure 17 each show a single flow subjected to a repeating loss pattern of three losses, each after 50 packet arrivals, followed by three more losses, each after 400 packet arrivals. For each graph the solid line shows the sending rate averaged over 0.2-second intervals, and the dashed line shows the sending rate averaged over one-second intervals. At the bottom of each graph is a mark for each packet drop. This loss pattern is designed to fit well with TFRC's mechanism of averaging the loss rate over roughly six successive loss intervals, so that TFRC maintains a steady estimation of the packet loss rate even with this bursty loss pattern. As Figure 17 shows, TFRC in this environment is considerably smoother than TCP(1/8), and at the same time achieves a slightly higher throughput.

In contrast, the more bursty loss pattern in Figure 18 is designed to bring out the worst in TFRC in terms of both smoothness and throughput. These simulations use a repeating loss pattern of a six-second low-congestion phase where every 200th packet is dropped, followed by a one-second heavy-congestion phase where every fourth packet is dropped. The heavy-congestion phase is designed to be just long enough to include six loss intervals, so that TFRC loses all memory of the earlier low-congestion period. In contrast, the low-congestion phase is designed to include only three or four loss intervals, not enough to totally supplant the memory of the heavy-congestion phase. The consequence is that, for this scenario, TFRC performs considerably worse than TCP(1/8), and indeed worse than TCP(1/2), in both smoothness and throughput. Figure 18 explains why TFRC performed badly in Figure 7, competing against TCP for a scenario with oscillating bandwidth with a period of four to eight seconds. In Figure 18, TFRC performs worse relative to TCP than it does in Figure 7. This suggests that the greatest difference between TCP and TFRC throughput occurs not when competing with a square-wave CBR source, but with a CBR source with short ON times and longer OFF times, giving relatively short periods of high congestion.

An equation-based congestion control mechanism other than TFRC, with a different algorithm for estimating the loss rate, would require a different loss pattern to illustrate its worst performance, but we would conjecture that any equation-based mechanism will have a corresponding loss pattern that exploits the weaknesses of the loss estimation algorithm, and for which the equation-based mechanism will perform badly relative to TCP and to TCP(1/8).

Figure 19 shows IIAD and SQRT congestion control with the same mildly bursty loss pattern as in Figure 17. Because IIAD reduces its window additively and increases its window slowly when bandwidth becomes available, it achieves smoothness at the cost of throughput, relative to SQRT.

5. Conclusion

The inappropriateness of TCP's AIMD for certain classes of applications, including some streaming media and multicast, has motivated the development of alternate slowly-responsive congestion control mechanisms governed by the TCP-compatibility requirement. However, this requirement is based on a static notion of throughput under a steady-state loss rate. Internet conditions are dynamic, which leads us to ask if the various proposed SlowCC TCP-compatible algorithms are compatible under dynamic conditions as well. This question is particularly important as there is some justified scepticism in the community about the sufficiency of the static requirement in practice.

We evaluated several recent proposals for SlowCC algorithms, including the equation-based TFRC, AIMD-based mechanisms with different constants from TCP, binomial algorithms, and RAP (a rate-based variant of TCP AIMD). We consider several per-

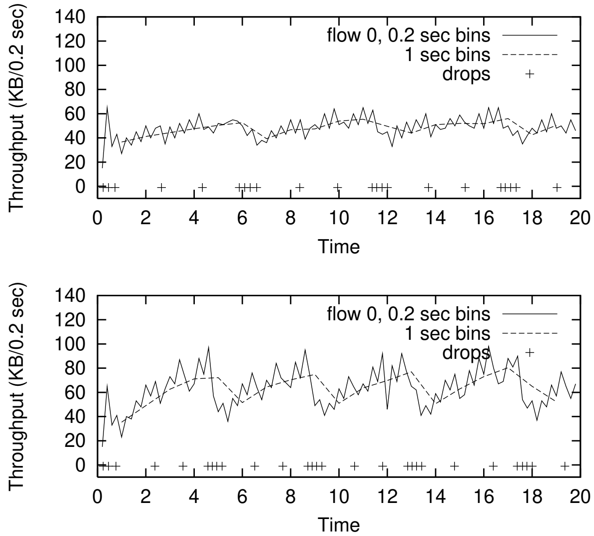

Figure 19: IIAD (top) and SQRT (bottom) with a mildly bursty loss pattern.

formance metrics, including persistent packet loss rates, long- and short-term fairness properties, bottleneck link utilization, and smoothness of transmission rates. We find that most of the TCP-compatible algorithms we studied appear to be safe for deployment; even the more slowly responsive ones can be made to avoid causing the network to go into persistent overload persistent loss rates on sudden bandwidth reductions by incorporating a self-clocking mechanism based on packet conservation. However, we also find that in return for smoother transmission rates, slowly-responsive algorithms lose throughput to faster ones (like TCP) under dynamic network conditions. Fortunately, this does not detract from their deployability because they do not take throughput away from the deployed base of TCP connections. We hope that these findings will help overcome some of the scepticism surrounding the behavior of slowly-responsive congestion control algorithms and move the Internet from an "only-TCP" paradigm to a TCP-compatible paradigm where multiple congestion control algorithms co-exist.

6. References

[1] ALLMAN, M., BALAKRISHNAN, H., AND FLOYD, S. *Enhancing TCP's Loss Recovery Using Limited Transmit*. Internet Engineering Task Force, January 2001. RFC 3042.

[2] BANSAL, D., AND BALAKRISHNAN, H. Binomial Congestion Control Algorithms. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)* (Anchorage, AK, April 2001), pp. 631–640.

[3] CHIU, D.-M., AND JAIN, R. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks and ISDN Systems 17* (1989), 1–14.

[4] FLOYD, S. *Congestion Control Principles*. Internet Engineering Task Force, September 2000. RFC 2914.

[5] FLOYD, S., AND FALL, K. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Trans. on Networking 7*, 4 (Aug. 1999), 458–472.

[6] FLOYD, S., HANDLEY, M., PADHYE, J., AND WIDMER, J. Equation-Based Congestion Control for Unicast Applications. In *SIGCOMM Symposium on Communications Architectures and Protocols* (Stockholm, Sweden, August 2000), pp. 43–56.

[7] FLOYD, S., HANDLEY, M., PADHYE, J., AND WIDMER, J. Equation-Based Congestion Control for Unicast Applications: The Extended Version. Tech. Rep. TR-00-03, International Computer Science Institute, March 2000. Available from `http://www.aciri.org/tfrc/`.

[8] FLOYD, S. AND HANDLEY, M. AND PADHYE, J. A Comparison of Equation-Based and AIMD Congestion Control, May 2000. Available from `http://www.aciri.org/tfrc/`.

[9] HANDLEY, M., PADHYE, J., FLOYD, S., AND WIDMER, J. TCP Friendly Rate Control (TFRC): Protocol Specification, May 2001. `draft-ietf-tsvwg-tfrc-02.txt`, Internet-Draft, work-in-progress.

[10] JACOBSON, V. Congestion Avoidance and Control. In *SIGCOMM Symposium on Communications Architectures and Protocols* (Stanford, CA, Aug. 1988), pp. 314–329. An updated version is available from `ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z`.

[11] MAHDAVI, J., AND FLOYD, S. TCP-Friendly Unicast Rate-Based Flow Control. Available from `http://www.psc.edu/networking/papers/tcp_friendly.html`, January 1997.

[12] ns-2 Network Simulator. `http://www.isi.edu/nsnam/ns/`, 2001.

[13] OTT, T., KEMPERMAN, J., AND MATHIS, M. The Stationary Distribution of Ideal TCP Congestion Avoidance. In *DIMACS Workshop on Performance of Realtime Applications on the Internet* (November 1996).

[14] PADHYE, J., FIROIU, V., TOWSLEY, D., AND KUROSE, J. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *SIGCOMM Symposium on Communications Architectures and Protocols* (Vancouver, Canada, Aug. 1998), pp. 303–314.

[15] RAMAKRISHNAN, K., AND FLOYD, S. *A Proposal to Add Explicit Congestion Notification (ECN) to IP*. Internet Engineering Task Force, Jan 1999. RFC 2481.

[16] REJAIE, R., HANDLEY, M., AND ESTRIN, D. RAP: An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)* (New York, NY, 1999), pp. 1337–1345.

[17] RHEE, I., OZDEMIR, V., AND YI, Y. TEAR: TCP Emulation at Receivers—Flow Control for Multimedia Streaming. Tech. rep., NCSU, April 2000. Available from `http://www.csc.ncsu.edu/faculty/rhee/export/tear_page/`.

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

[19] YANG, Y., KIM, M., AND LAM, S. Transient Behaviors of TCP-friendly Congestion Control Protocols. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)* (Anchorage, AK, April 2001), pp. 1716–1725.

[20] YANG, Y., AND LAM, S. General AIMD Congestion Control. Tech. Rep. TR-2000-09, University of Texas at Austin, May 2000. Available from `http://www.cs.utexas.edu/users/lam/NRL/TechReports/`.

Appendix

A. Modeling the Role of Timeouts

As mentioned earlier, TCP's retransmit timeouts are a key component of TCP congestion control, and the relative fairness of SlowCC congestion control mechanisms relies on their ability to take into account TCP's timeouts as well as the AIMD mechanisms. Here, we show that the exponential backoff of TCP's retransmit timers can in fact be viewed as an extension of the AIMD model to an environment with sending rates less than one packet per RTT.

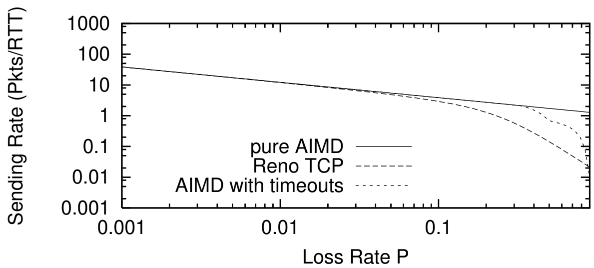

Figure 20: The throughput equation for models with and without timeouts.

The dashed line labeled "Reno TCP" in Figure 20 shows the TCP throughput equation [14] taking into account the role of retransmit timeouts in Reno TCP without delayed acknowledgments. The solid line labeled "pure AIMD" in Figure 20 shows the throughput equation for a pure AIMD scheme without timeouts [5]. The solid line is derived from a simple deterministic model where, for a packet drop-rate p, one in every $1/p$ packets are dropped, causing the sending rate to be halved. This "pure AIMD" model of a sending rate of about $\sqrt{1.5/p}$ packets/RTT does not apply to TCP for a sending rate less than one packet per round-trip time (i.e., for packet drop rates greater than about one-third).

To apply the AIMD model to sending rates less than one packet per RTT, we assume that only complete packets can be sent, and that when the sending rate is less than one packet per RTT, the sender waits for a complete inter-packet interval before sending a new packet. In this deterministic model with the available bandwidth less than one packet/RTT, the sending rate can be determined as follows. Define *stages* such that at state 0 the sending rate is one packet/RTT, and at state i the sending rate is $1/2^i$ packets/RTT, or one packet every 2^i RTTs. At any stage if a packet is acknowledged the sender returns to stage 0, and immediately sends one packet. Otherwise, the sender halves its sending rate, waiting 2^{i+1} RTTs before retransmitting a packet. This halving of the sending rate in response to a packet drop is then equivalent to an exponential backoff of the retransmit timer.

Under this model of transmission, let the steady-state packet drop rate be $p = \frac{n}{n+1}$. Thus, the sender sends $n+1$ packets over $2^{n+1}-1$ round-trip times, with all but the last packet dropped. This gives a steady-state sending rate in this model of:

$$\frac{n+1}{2^{n+1}-1} = \frac{\frac{1}{1-p}}{2^{\frac{1}{1-p}}-1}$$

packets/RTT. For example, for $p = 1/2$, we have $n = 1$, and the sender sends two packets every three round-trip times, for a steady-state sending rate of $2/3$ packets/RTT. This is shown in Figure 20 with the short dashed line labeled "AIMD with timeouts". We note that this analysis is only valid for packet drop rates of 50% or more, while the "pure AIMD" analysis can apply to packet drop rates up to 33%. The "AIMD with timeouts" line gives an upper bound for the analytic behavior of TCP, while the "Reno TCP" line gives a lower bound. The behavior of TCPs with Selective Acknowledgements, Limited Transmit [1], and ECN should fall somewhere between the two lines.