FAST TCP:

Motivation, Architecture, Algorithms, Performance

Cheng Jin David X. Wei Steven H. Low
Engineering & Applied Science, Caltech
http://netlab.caltech.edu

***Abstract*— We describe FAST TCP, a new TCP congestion control algorithm for high-speed long-latency networks, from design to implementation. We highlight the approach taken by FAST TCP to address the four difficulties, at both packet and flow levels, which the current TCP implementation has at large windows. We describe the architecture and characterize the equilibrium and stability properties of FAST TCP. We present experimental results comparing our first Linux prototype with TCP Reno, HSTCP, and STCP in terms of throughput, fairness, stability, and responsiveness. FAST TCP aims to rapidly stabilize high-speed long-latency networks into steady, efficient and fair operating points, in dynamic sharing environments, and the preliminary results are promising.**

I. Introduction

Congestion control is a distributed algorithm to share network resources among competing users. It is important in situations where the availability of resources and the set of competing users vary over time unpredictably, yet efficient sharing is desired. These constraints, unpredictable supply and demand and efficient operation, necessarily lead to feedback control as the preferred approach, where traffic sources dynamically adapt their rates to congestion in their paths. On the Internet, this is performed by the Transmission Control Protocol (TCP) in source and destination computers involved in data transfers.

The congestion control algorithm in the current TCP, which we refer to as Reno, was developed in 1988 [1] and has gone through several enhancements since, e.g., [2], [3], [4]. It has performed remarkably well and is generally believed to have prevented severe congestion as the Internet scaled up by six orders of magnitude in size, speed, load, and connectivity. It is also well-known, however, that as bandwidth-delay product continues to grow, TCP Reno will eventually become a performance bottleneck itself. The following four difficulties contribute to the poor performance of TCP Reno in networks with large bandwidth-delay products:

1) At the packet level, linear increase by one packet per Round-Trip Time (RTT) is too slow, and multiplicative decrease per loss event is too drastic.
2) At the flow level, maintaining large average congestion windows *requires* an extremely small equilibrium loss probability.
3) At the packet level, oscillation is unavoidable because TCP uses a binary congestion signal (packet loss).
4) At the flow level, the dynamics is unstable, leading to severe oscillations that can only be reduced by the accurate estimation of packet loss probability and a stable design of the flow dynamics.

We explain these difficulties in detail in Section II. In [5], we described HSTCP [6] and STCP [7], two loss-based solutions to these problems. In this paper, we propose a delay-based solution. See [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18] for other proposals.

In Section III, we motivate delay-based approach. Delay-based congestion control has been proposed, e.g., in [19], [20], [8]. Its advantage over loss-based approach is small at low speed, but decisive at high speed, as we will argue below. As pointed out in [21], delay can be a poor or untimely predictor of packet loss, and therefore using a delay-based algorithm to augment the basic AIMD (Additive Increase Multiplicative Decrease) algorithm of TCP Reno is the wrong approach to address the above difficulties at large windows. Instead, a new approach that fully exploits delay as a congestion measure, augmented with loss information, is needed. FAST TCP uses this approach. Using queueing delay as the congestion measure has two advantages.

First, queueing delay can be more accurately estimated than loss probability both because packet losses in networks with large bandwidth-delay product are rare events (probability on the order 10^{-8} or smaller), and because loss samples provide coarser information than queueing delay samples. Indeed, measurements of delay are noisy, just as those of loss probability. Each measurement of packet loss (whether a packet is lost) provides one bit of information for the filtering of noise, whereas each measurement of queueing delay provides multi-bit information. This makes it easier for an equation-based implementation to stabilize a network into a steady state with a target fairness and high utilization. Second, the dynamics of queueing delay seems to have the right scaling with respect to network capacity. This helps maintain stability as a network scales up in capacity [22], [23], [24]. In Section III, we explain how we exploit these advantages to address the four difficulties of TCP Reno.

In Section IV, we lay out an architecture to implement our design. Even though the discussion is in the context of FAST TCP, the architecture can also serve as a general framework to guide the design of other congestion control mechanisms, not necessarily limited to TCP, for high-speed networks. The main components in the architecture can be designed separately and upgraded asynchronously. Unlike the conventional design, FAST TCP can use the same window and burstiness control algorithms regardless of whether a source is in the normal state or the loss recovery state. This leads to a clean separation of components in both functionality and code structure. We then present an overview of some of the algorithms implemented in our current prototype.

In Section V, we present a mathematical model of the window control algorithm. We prove that FAST TCP has the same equilibrium properties as TCP Vegas [25], [26]. In particular, it does not penalize flows with large propagation delays, and it achieves weighted proportional fairness [27]. For the special case of single bottleneck link with heterogeneous flows, we prove that the window control algorithm of FAST is globally stable, in the absence of

feedback delay. Moreover, starting from any initial state, a network converges exponentially to a unique equilibrium.

In Section VI, we present preliminary experimental results to illustrate throughput, fairness, stability, and responsiveness of FAST TCP, in the presence of delay and in heterogeneous and dynamic environments where flows of different delays join and depart asynchronously. We compare the performance of FAST TCP with Reno, HSTCP (HighSpeed TCP [6]), and STCP (Scalable TCP [7]), using their default parameters. In these experiments, FAST TCP achieved the best performance under each criterion, while HSTCP and STCP improved throughput and responsiveness over Reno at the cost of fairness and stability. We conclude in Section VII.

II. Problems at large windows

A congestion control algorithm can be designed at two levels. The *flow*-level (macroscopic) design aims to achieve high utilization, low queueing delay and loss, fairness, and stability. The *packet*-level design implements these flow-level goals within the constraints imposed by end-to-end control. Historically for TCP Reno, packet-level implementation was introduced first. The resulting flow-level properties, such as fairness, stability, and the relationship between equilibrium window and loss probability, were then understood as an afterthought. In contrast, the packet-level designs of HSTCP [6], STCP [7], and FAST TCP are explicitly guided by flow-level goals.

We elaborate in this section on the four difficulties of TCP Reno listed in Section I. It is important to distinguish between packet-level and flow-level difficulties because they must be addressed by different means.

A. Packet and flow level modeling

The congestion avoidance algorithm of TCP Reno and its variants have the form of AIMD [1]. The pseudo code for window adjustment is:

$$\begin{aligned} \texttt{Ack:} \quad & \texttt{w} \longleftarrow \texttt{w} + \frac{1}{\texttt{w}} \\ \texttt{Loss:} \quad & \texttt{w} \longleftarrow \texttt{w} - \frac{1}{2}\texttt{w} \end{aligned}$$

This is a packet-level model, but it induces certain flow-level properties such as throughput, fairness, and stability.

These properties can be understood with a flow-level model of the AIMD algorithm, e.g., [28], [29], [30]. The window $w_i(t)$ of source i increases by 1 packet per RTT,[1] and decreases per unit time by

$$x_i(t)p_i(t) \cdot \frac{1}{2} \cdot \frac{4}{3} w_i(t) \quad \text{packets}$$

where

$$x_i(t) \;:=\; w_i(t)/T_i(t) \quad \text{pkts/sec}$$

$T_i(t)$ is the round-trip time, and $p_i(t)$ is the (delayed) end-to-end loss probability, in period t.[2] Here, $4w_i(t)/3$ is the peak window size that gives the "average" window of $w_i(t)$. Hence, a flow-level model of AIMD is:

$$\dot{w}_i(t) \;=\; \frac{1}{T_i(t)} - \frac{2}{3} x_i(t) p_i(t) w_i(t) \tag{1}$$

Setting $\dot{w}_i(t) = 0$ in (1) yields the well-known $1/\sqrt{p}$ formula for TCP Reno discovered in [31], [32], which relates loss probability to window size in equilibrium:

$$p_i^* \;=\; \frac{3}{2w_i^{*2}} \tag{2}$$

In summary, (1) and (2) describe the flow-level dynamics and the equilibrium, respectively, for TCP Reno. It turns out that different variants of TCP all have the same dynamic structure at the flow level (see [5], [33]). By defining

$$\kappa_i(w_i, T_i) \;=\; \frac{1}{T_i} \quad \text{and} \quad u_i(w_i, T_i) \;=\; \frac{1.5}{w_i^2}$$

and noting that $w_i = x_i T_i$, we can express (1) as:

$$\dot{w}_i(t) \;=\; \kappa(t) \left(1 - \frac{p_i(t)}{u_i(t)}\right) \tag{3}$$

where we have used the shorthand $\kappa_i(t) = \kappa_i(w_i(t), T_i(t))$ and $u_i(t) = u_i(w_i(t), T_i(t))$. Equation 3 can be used to describe all known TCP variants, and different variants differ in their choices of the gain function κ_i and marginal utility function u_i, and whether the congestion measure p_i is loss probability or queueing delay.

Next, we illustrate the equilibrium and dynamics problems of TCP Reno, at both the packet and flow levels, as bandwidth-delay product increases.

B. Equilibrium problem

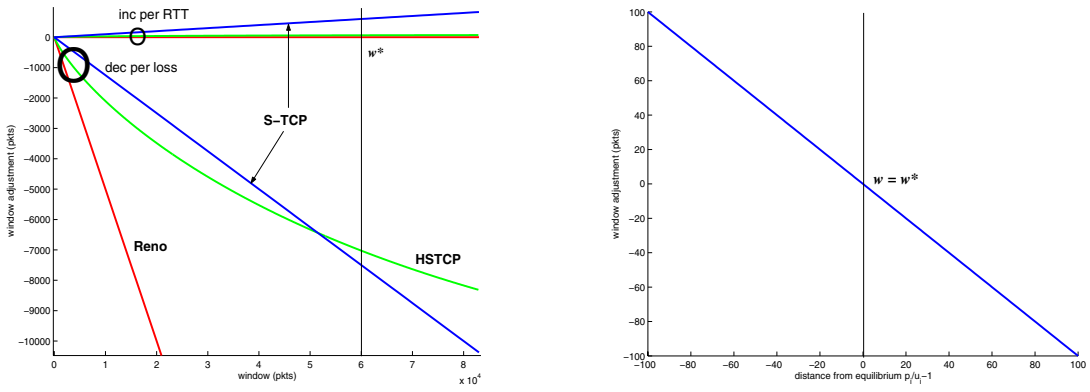
The equilibrium problem at the flow level is expressed in (2): the end-to-end loss probability must be exceedingly small to sustain a large window size, making the equilibrium difficult to maintain in practice, as bandwidth-delay product increases.

Even though equilibrium is a flow-level notion, this problem manifests itself at the packet level, where a source increments its window too slowly and decrements it too drastically. When the peak window is 80,000-packet (corresponding to an "average" window of 60,000 packets), which is necessary to sustain 7.2Gbps using 1,500-byte packets with a RTT of 100ms, it takes 40,000 RTTs, or almost 70 minutes, to recover from a single packet loss. This is illustrated in Figure 1a, where the size of window increment per RTT and decrement per loss, 1 and $0.5w_i$, respectively, are plotted as functions of w_i. The increment function for Reno (and for HSTCP) is almost indistinguishable from the x-axis. Moreover, the gap between the increment and decrement functions grows rapidly as w_i increases. Since the average increment and decrement must be equal in equilibrium, the required loss probability can be exceedingly small at large w_i. This picture is thus simply a visualization of (2).

To address the difficulties of TCP Reno at large window sizes, HSTCP and STCP increase more aggressively and decrease more gently, as discussed in [5], [33].

[1] It should be $(1 - p_i(t))$ packets, where $p_i(t)$ is the end-to-end loss probability. This is roughly 1 when $p_i(t)$ is small.

[2] This model assumes that window is halved on each packet loss. It can be modified to model the case, where window is halved at most once in each RTT. This does not qualitatively change the following discussion.



(a) Reno, HSTCP, and STCP

(b) FAST

Fig. 1. Packet-level implementation: (a) Window increment per RTT and decrement per loss, as functions of the current window. The increment functions for TCP Reno and HSTCP are almost identical at this scale. (b) Window update as a function of *distance from equilibrium* for FAST.

C. Dynamic problems

The causes of the oscillatory behavior of TCP Reno lie in its design at both the packet and flow levels. At the packet level, the choice of binary congestion signal necessarily leads to oscillation, and the parameter setting in Reno worsens the situation as bandwidth-delay product increases. At the flow level, the system dynamics given by (1) is unstable at large bandwidth-delay products [29], [30]. These must be addressed by different means, as we now elaborate.

Figure 2(a) illustrates the operating points chosen by various TCP congestion control algorithms, using the single-link single-flow scenario. It shows queueing delay as a function of window size. Queueing delay starts to build up after point C where window equals bandwidth-propagation-delay product, until point R where the queue overflows. Since Reno oscillates around point R, the peak window size goes beyond point R. The minimum window in steady state is half of the peak window. This is the basis for the rule of thumb that bottleneck buffer should be at least one bandwidth-delay product: the minimum window will then be above point C, and buffer will not empty in steady state operation, yielding full utilization.

In the loss-based approach, full utilization, even if achievable, comes at the cost of severe oscillations and potentially large queueing delay. The DUAL scheme in [20] proposes to oscillate around point D, the midpoint between C and R when the buffer is half-full. DUAL increases congestion window linearly by one packet per RTT, as long as queueing delay is less than half of the maximum value, and decreases multiplicatively by a factor of 1/8, when queueing delay exceeds half of the maximum value. The scheme CARD (Congestion Avoidance using Round-trip Delay) of [19] proposes to oscillate around point C through AIMD with the same parameter $(1, 1/8)$ as DUAL, based on the ratio of round-trip delay and delay gradient, to maximize power. In all these schemes, the congestion signal is used as a binary signal, and hence congestion window *must* oscillate.

Congestion window can be stabilized only if multi-bit feedback is used. This is the approach taken by the equation-based algorithm in [34], where congestion window is adjusted based on the estimated loss probability in an attempt to stabilize around a target value given by (2). Its operating point is T in Figure 2(b), near the overflowing point. This approach eliminates the oscillation due to packet-level AIMD, but two difficulties remain at the flow level.

First, equation-based control requires the explicit estimation of end-to-end loss probability. This is difficult when the loss probability is small. Second, even if loss probability can be perfectly estimated, Reno's flow dynamics, described by equation (1) leads to a feedback system that becomes unstable as feedback delay increases, and more strikingly, as network capacity increases [29], [30]. The instability at the flow level can lead to severe oscillations that can be reduced *only* by stabilizing the flow-level dynamics. We will return to both points in Section III.

III. Delay-based approach

In this section, we motivate delay-based approach to address the four difficulties at large window sizes.

A. Motivation

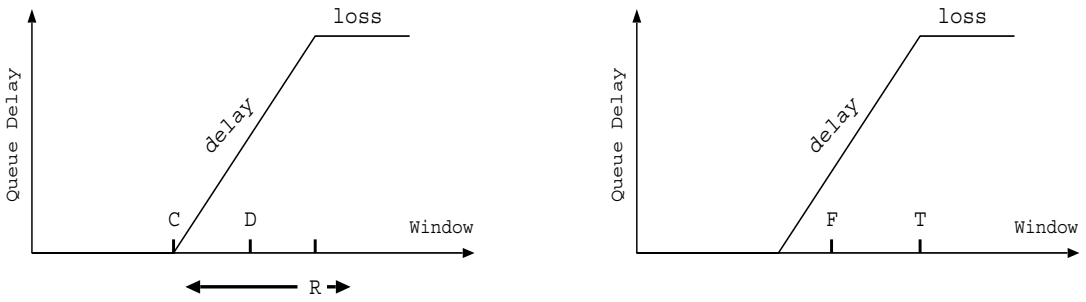
Although improved loss-based protocols such as HSTCP and STCP have been proposed as replacements to TCP Reno, we showed in [5] that they don't address all four problems (Section I) of TCP Reno. To illustrate this, we plot the increment and decrement functions of HSTCP and STCP in Figure 1(a) alongside TCP Reno. Both protocols upper bound TCP Reno: each increases more aggressively and decreases less drastically, so that the gap between the increment and decrement functions is narrowed. This means, in equilibrium, both HSTCP and STCP can tolerate larger loss probabilities than TCP Reno, thus achieving larger equilibrium windows. However, neither solves the dynamics problems at both the packet and the flow levels.

In [5], we show that the congestion windows in Reno, HSTCP and STCP all evolve according to:

$$\dot{w}_i(t) \quad = \quad \kappa_i(t) \cdot \left(1 - \frac{p_i(t)}{u_i(t)}\right) \tag{4}$$

where $\kappa(t) := \kappa_i(w_i(t), T_i(t))$ and $u_i(t) := u_i(w_i(t), T_i(t))$. Moreover, the dynamics of FAST TCP also takes the same form; see below. They differ only in the choice of the gain function $\kappa_i(w_i, T_i)$, the marginal utility function $u_i(w_i, T_i)$, and the end-to-end congestion measure p_i. Hence, at the flow level, there are only three design decisions:

- $\kappa_i(w_i, T_i)$: the choice of the gain function κ_i determines the dynamic properties such as stability and responsiveness, but does not affect the equilibrium properties.



(a) Binary signal: oscillatory

(b) Multi-bit signal: stabilizable

Fig. 2. Operating points of TCP algorithms: R: Reno [1], HSTCP [6], STCP [7]; D: DUAL [20]; C: CARD [19]; T: TFRC [34]; F: Vegas [8], FAST.

- $u_i(w_i, T_i)$: the choice of the marginal utility function u_i mainly determines equilibrium properties such as the equilibrium rate allocation and its fairness.
- p_i: in the absence of explicit feedback, the choice of congestion measure p_i is limited to loss probability or queueing delay. The dynamics of $p_i(t)$ is determined at links.

The design choices in Reno, HSTCP, STCP and FAST are shown in Table I.

	$\kappa_i(w_i, T_i)$	$u_i(w_i, T_i)$	p_i
Reno	$1/T_i$	$1.5/w_i^2$	loss probability
HSTCP	$\frac{0.16b(w_i)w_i^{0.80}}{(2-b(w_i))T_i}$	$0.08/w_i^{1.20}$	loss probability
STCP	aw_i/T_i	ρ/w_i	loss probability
FAST	$\gamma\alpha_i$	α_i/x_i	queueing delay

TABLE I

COMMON DYNAMIC STRUCTURE: w_i IS SOURCE i'S WINDOW SIZE, T_i IS ITS ROUND-TRIP TIME, p_i IS CONGESTION MEASURE, $x_i = w_i/T_i$; $a, b(w_i), \rho, \gamma, \alpha_i$ ARE PROTOCOL PARAMETERS.

These choices produce equilibrium characterizations shown in Table II.

Reno	$x_i = \frac{1}{T_i} \cdot \frac{\alpha_i}{p_i^{0.50}}$
HSTCP	$x_i = \frac{1}{T_i} \cdot \frac{\alpha_i}{p_i^{0.84}}$
STCP	$x_i = \frac{1}{T_i} \cdot \frac{\alpha_i}{p_i}$
FAST	$x_i = \frac{\alpha_i}{p_i}$

TABLE II

COMMON EQUILIBRIUM STRUCTURE.

This common model (4) can be interpreted as follows: the goal at the flow level is to equalize marginal utility $u_i(t)$ with the end-to-end measure of congestion, $p_i(t)$. This interpretation immediately suggests an equation-based packet-level implementation where *both* the direction and size of the window adjustment $\dot{w}_i(t)$ are based on the difference between the ratio $p_i(t)/u_i(t)$ and the target of 1. Unlike the approach taken by Reno, HSTCP, and STCP, this approach eliminates packet-level oscillations due to the binary nature of congestion signal. It however requires the *explicit* estimation of the end-to-end congestion measure $p_i(t)$.

Without explicit feedback, $p_i(t)$ can only be loss probability, as used in TFRC [34], or queueing delay, as used in TCP Vegas [8] and FAST TCP.[3] Queueing delay can be more accurately estimated than loss probability both because packet losses in networks with large bandwidth-delay products are rare events (probability on the order 10^{-8} or smaller), and because loss samples provide coarser information than queueing delay samples. Indeed, each measurement of packet loss (whether a packet is lost) provides one bit of information for the filtering of noise, whereas each measurement of queueing delay provides multi-bit information. This allows an equation-based implementation to stabilize a network into a steady state with a target fairness and high utilization.

At the flow level, the dynamics of the feedback system must be stable in the presence of delay, as the network capacity increases. Here, again, queueing delay has an advantage over loss probability as a congestion measure: the dynamics of queueing delay seems to have the right scaling with respect to network capacity. This helps maintain stability as network capacity grows [22], [23], [24].

B. Implementation strategy

The delay-based approach, with proper flow and packet level designs, can address the four difficulties of Reno at large windows. First, by explicitly estimating how far the current state $p_i(t)/u_i(t)$ is from the equilibrium value of 1, our scheme can drive the system rapidly, yet in a fair and stable manner, toward the equilibrium. The window adjustment is small when the current state is close to equilibrium and large otherwise, *independent of where the equilibrium is*, as illustrated in Figure 1(b). This is in stark contrast to the approach taken by Reno, HSTCP, and STCP, where window adjustment depends on just the current window size and is independent of where the current state is with respect to the target (compare Figures 1(a) and (b)). Like the equation-based scheme in [34], this approach avoids the problem of slow increase and drastic decrease in Reno, as the network scales up.

Second, by choosing a multi-bit congestion measure, this approach eliminates the packet-level oscillation due to binary feedback, avoiding Reno's third problem.

Third, using queueing delay as the congestion measure $p_i(t)$ allows the network to stabilize in the region below the overflowing point, around point F in Figure 2(b), when the buffer size is sufficiently large. Stabilization at this operating point eliminates large queueing delay and unnecessary packet loss. More importantly, it makes room for buffering "mice" traffic. To avoid the second problem in Reno, where the required equilibrium congestion measure (loss probability for Reno, and queueing delay here) is too small to practically

[3]It is debatable whether TCP Vegas is equation-based since the *size* of its window adjustment does not depend on queueing delay. This is not important at low speed but critical at high speed.

estimate, the algorithm must adapt its parameter α_i with capacity to maintain small but sufficient queueing delay.

Finally, to avoid the fourth problem of Reno, the window control algorithm must be stable, in addition to being fair and efficient, at the flow level. The use of queueing delay as a congestion measure facilitates the design as queueing delay naturally scales with capacity [22], [23], [24].

The design of TCP congestion control algorithm can thus be conceptually divided into two levels:

- At the flow level, the goal is to design a class of function pairs, $u_i(w_i, T_i)$ and $\kappa(w_i, T_i)$, so that the feedback system described by (4), together with link dynamics in $p_i(t)$ and the interconnection, has an equilibrium that is fair and efficient, and that the equilibrium is stable, in the presence of feedback delay.
- At the packet level, the design must deal with issues that are ignored by the flow-level model or modeling assumptions that are violated in practice, in order to achieve these flow-level goals. These issues include burstiness control, loss recovery, and parameter estimation.

The implementation then proceeds in three steps:

1) determine various system components;
2) translate the flow-level design into packet-level algorithms;
3) implement the packet-level algorithms in a specific operating system.

The actual process iterates intimately between flow and packet level designs, between theory, implementation, and experiments, and among the three implementation steps.

The emerging theory of large-scale networks under end-to-end control, e.g., [27], [35], [36], [25], [37], [38], [26], [39], [22], [40], [41], [29], [30], [42], [43], [24], [23], [15] (see also, e.g., [44], [45], [46] for recent surveys), forms the foundation of the flow-level design. The theory plays an important role by providing a framework to understand issues, clarify ideas, and suggest directions, leading to a robust and high performance implementation.

We lay out the architecture of FAST TCP next.

IV. Architecture and Algorithms

We separate the congestion control mechanism of TCP into four components in Figure 3. These four components are functionally independent so that they can be designed separately and upgraded asynchronously. In this section, we focus on the two parts that we have implemented in the current prototype (see [33]).

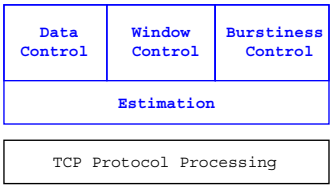


Fig. 3. FAST TCP architecture.

The *data control* component determines *which* packets to transmit, *window control* determines *how many* packets to transmit, and *burstiness control* determines *when* to transmit these packets. These decisions are made based on information provided by the *estimation* component. Window control regulates packet transmission at the RTT timescale, while burstiness control works at a smaller timescale.

In the following subsections, we provide an overview of *estimation* and *window control* and the algorithms implemented in our current prototype. An initial prototype that included the features discussed here was demonstrated in November 2002 at the SuperComputing Conference, and the experimental results were reported in [47].

A. Estimation

This component provides estimations of various input parameters to the other three decision-making components. It computes two pieces of feedback information for each data packet sent. When a positive acknowledgment is received, it calculates the RTT for the corresponding data packet and updates the average queueing delay and the minimum RTT. When a negative acknowledgment (signaled by three duplicate acknowledgments or timeout) is received, it generates a loss indication for this data packet to the other components. The estimation component generates both a multi-bit queueing delay sample and a one-bit loss-or-no-loss sample for each data packet.

The queueing delay is smoothed by taking a moving average with the weight $\eta(t) := \min\{3/w_i(t), 1/4\}$ that depends on the window $w_i(t)$ at time t, as follows. The k-th RTT sample $T_i(k)$ updates the average RTT $\overline{T}_i(k)$ according to:

$$\overline{T}_i(k+1) \quad = \quad (1-\eta(t_k))\overline{T}_i(k) + \eta(t_k)T_i(k)$$

where t_k is the time at which the k-th RTT sample is received. Taking $d_i(k)$ to be the minimum RTT observed so far, the average queueing delay is estimated as:

$$\hat{q}_i(k) \quad = \quad \overline{T}_i(k) - d_i(k)$$

The weight $\eta(t)$ is usually much smaller than the weight (1/8) used in TCP Reno. The average RTT $\overline{T}_i(k)$ attempts to track the average over one congestion window. During each RTT, an entire window worth of RTT samples are received if every packet is acknowledged. Otherwise, if delayed ack is used, the number of queueing delay samples is reduced so $\eta(t)$ should be adjusted accordingly.

B. Window control

The window control component determines congestion window based on congestion information—queueing delay and packet loss, provided by the estimation component. A key decision in our design that departs from traditional TCP design is that the same algorithm is used for congestion window computation independent of the state of the sender. For example, in TCP Reno (without rate halving), congestion window is increased by one packet every RTT when there is no loss, and increased by one for each duplicate ack during loss recovery. In FAST TCP, we would like to use the same algorithm for window computation regardless of the sender state.

Our congestion control mechanism reacts to both queueing delay and packet loss. Under normal network conditions, FAST periodically updates the congestion window based on the average RTT and average queueing delay provided by the estimation component, according to:

$$\mathtt{w} \longleftarrow \min\left\{2\mathtt{w},\ (1-\gamma)\mathtt{w} + \gamma\left(\frac{\mathtt{baseRTT}}{\mathtt{RTT}}\mathtt{w} + \alpha(\mathtt{w},\ \mathtt{qdelay})\right)\right\} \quad (5)$$

where $\gamma \in (0, 1]$, `baseRTT` is the minimum RTT observed so far, and `qdelay` is the end-to-end (average) queueing delay. In our current implementation, congestion window changes over two RTTs: it is updated in one RTT and frozen in the

next. The update is spread out over the first RTT in a way such that congestion window is no more than doubled in each RTT.

In our current prototype, we choose the function $\alpha(\texttt{w, qdelay})$ to be a constant at all times. This produces linear convergence when the qdelay is zero. Alternatively, we can use a constant α only when qdelay is nonzero and an α proportional to window, $\alpha(\texttt{w, qdelay}) = a\texttt{w}$, when qdelay is zero. In this case, when qdelay is zero, FAST performs multiplicative increase and grows exponentially at rate a to a neighborhood of $\texttt{qdelay} > 0$. Then $\alpha(\texttt{w, qdelay})$ switches to a constant α and, as we will see in Theorem 2 below, window converges exponentially to the equilibrium at a different rate that depends on qdelay. The constant α is the number of packets each flow attempts to maintain in the network buffer(s) at equilibrium, similar to TCP Vegas [8].[4]

Although we would like to use the same congestion control function during loss recovery, we have currently disabled this feature because of ambiguities associated with retransmitted packets. Currently when a packet loss is detected, FAST halves its window and enters loss recovery. The goal is to back off packet transmission quickly when severe congestion occurs, in order to bring the system back to a regime where reliable RTT measurements are again available for window adjustment (5) to work effectively. A source does not react to delay until it exits loss recovery.[5]

C. Packet-level implementation

It is important to maintain an abstraction of the implementation as as the code evolves. This abstraction should describe the high-level operations each component performs based on external inputs, and can serve as a road map for future TCP implementations as well as improvements to the existing implementation. Whenever a non-trivial change is required, one should first update this abstraction to ensure that the overall packet-level code would be built on a sound underlying foundation.

Since TCP is an event-based protocol, our control actions should be triggered by the occurrence of various events. Hence, we need to translate our flow-level algorithms into event-based packet-level algorithms. There are four types of events that FAST TCP reacts to: on the reception of an acknowledgment, after the transmission of a packet, at the end of a RTT, and for each packet loss.

For each acknowledgment received, the estimation component computes the average queueing delay, and the burstiness control component determines whether packets can be injected into the network. For each packet transmitted, the estimation component records a time-stamp, and the burstiness control component updates corresponding data structures for bookkeeping. At a constant time interval, which we check on the arrival of each acknowledgment, window control calculates a new window size. At the end of each RTT, burstiness reduction calculates the target throughput using the window and RTT measurements in the last RTT. Window pacing will then schedule to break up a large increment in congestion window into smaller increments over time. During loss recovery, congestion window should be continually updated based on congestion signals from the network. Upon the detection of a packet loss event, a sender determines whether to retransmit each unacknowledged packet right away or hold off until a more appropriate time.

[4]All experiments in Section VI used linear increase, i.e., $\alpha(\texttt{w, qdelay}) = \alpha$ for all qdelay.

[5]In the Linux TCP implementation, congestion window was frequently reduced to one when there were heavy losses. In order to ensure a reasonable recovery time, we impose a minimum window of 16 packets during loss recovery for connections that use large windows. This and other interim measures will be improved in future FAST TCP releases.

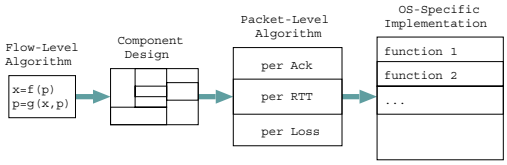


Fig. 4. From flow-level design to implementation.

Figure 4 presents an approach to turn the high-level design of a congestion control algorithm into an implementation. First, an algorithm is designed at the flow-level and analyzed to ensure that it meets the high-level objectives such as fairness and stability. Based on that, one can determine the components necessary to implement congestion control. The flow-level algorithm can then be translated into a packet-level algorithm that consists of a set of event-based tasks. The event-based tasks should be independent of any specific TCP or operating system implementation, but yet detailed enough so the understanding of these tasks enables one to implement FAST in any operating system or protocol stack.

V. Equilibrium and Stability of Window Control Algorithm

In this section, we present a model of the window control algorithm. We show that, in equilibrium, the vectors of source windows and link queueing delays are the unique solutions of a pair of optimization problems (9)–(10). This completely characterizes the network equilibrium properties such as throughput, fairness, and delay. We also analyze the stability of the window control algorithm. We prove in [33] that, for a single link with heterogeneous sources, the window control algorithm (5) is globally stable, assuming zero feedback delay, and converges exponentially to a unique equilibrium. Extensive experiments in Section VI illustrate its stability in the presence of feedback delay.

Given a network that consists of a set of resources with finite capacities c_l, e.g., transmission links, processing units, memory, etc., we refer to them in general as "links" in our model. The network is shared by a set of unicast flows, identified by their sources. Let d_i denote the round-trip propagation delay of source i. Let R be the routing matrix where $R_{li} = 1$ if source i uses link l, and 0 otherwise. Let $p_l(t)$ denote the queueing delay at link l at time t. Let $q_i(t) = \sum_l R_{li} p_l(t)$ be the round-trip queueing delay, or in vector notation, $q(t) = R^T p(t)$. Then the round trip time of source i is $T_i(t) := d_i + q_i(t)$.

Each source i adapts $w_i(t)$ periodically according to: [6]

$$w_i(t+1) \quad = \quad \gamma\left(\frac{d_i w_i(t)}{d_i + q_i(t)} + \alpha_i(w_i(t), q_i(t))\right) \; + \; (1-\gamma) w_i(t) \quad (6)$$

[6]Note that (6) can be rewritten as (when $\alpha_i(w_i, q_i) = \alpha_i$, constant)

$$w_i(t+1) \quad = \quad w_i(t) + \gamma_i(\alpha_i - x_i(t) q_i(t))$$

From [26], TCP Vegas updates its window according to

$$w_i(t+1) \quad = \quad w_i(t) + \frac{1}{T_i(t)} \, \mathrm{sgn}(\alpha_i - x_i(t) q_i(t))$$

where $\mathrm{sgn}(z) = -1$ if $z < 0$, 0 if $z = 0$, and 1 if $z > 0$. Hence FAST can be thought of as a high-speed version of Vegas.

where $\gamma \in (0,1]$, at time t, and $\alpha_i(w_i, q_i)$ is defined by:

$$\alpha_i(w_i, q_i) \quad = \quad \begin{cases} a_i w_i & \text{if } q_i = 0 \\ \alpha_i & \text{otherwise} \end{cases} \tag{7}$$

A key departure of our model from those in the literature is that we assume that a source's *send rate*, defined as $x_i(t) := w_i(t)/T_i(t)$, cannot exceed the *throughput* it receives. This is justified because of self-clocking: one round-trip time after a congestion window is increased, packet transmission will be clocked at the same rate as the throughput the flow receives. See [48] for detailed justification and validation experiments. A consequence of this assumption is that the link queueing delay vector, $p(t)$, is determined implicitly by the instantaneous window size in a *static* manner: given $w_i(t) = w_i$ for all i, the link queueing delays $p_l(t) = p_l \geq 0$ for all l are given by:

$$\sum_i R_{li} \frac{w_i}{d_i + q_i} \qquad \begin{cases} = c_l & \text{if } p_l > 0 \\ \leq c_l & \text{if } p_l = 0 \end{cases} \tag{8}$$

where again $q_i = \sum_l R_{li} p_l$.

The equilibrium values of windows w^* and delays p^* of the network defined by (6)–(8) can be characterized as follows. Consider the utility maximization problem

$$\max_{x \geq 0} \sum_i \alpha_i \log x_i \quad \text{s.t.} \quad Rx \leq c \tag{9}$$

and the following (dual) problem:

$$\min_{p \geq 0} \qquad \sum_l c_l p_l - \sum_i \alpha_i \log \sum_l R_{li} p_l \tag{10}$$

Theorem 1: Suppose R has full row rank. The unique equilibrium point (w^*, p^*) of the network defined by (6)–(8) exists and is such that $x^* = (x_i^* := w_i/(d_i + q_i^*), \forall i)$ is the unique maximizer of (9) and p^* is the unique minimizer of (10). This implies in particular that the equilibrium rate x^* is α_i-weighted proportionally fair.

Theorem 1 implies that FAST TCP has the same equilibrium properties as TCP Vegas [25], [26]. Its throughput is given by

$$x_i \quad = \quad \frac{\alpha_i}{q_i} \tag{11}$$

In particular, it does not penalize sources with large propagation delays d_i. The relation (11) also implies that, in equilibrium, source i maintains α_i packets in the buffers along its path [25], [26]. Hence, the total amount of buffering in the network must be at least $\sum_i \alpha_i$ packets in order to reach the equilibrium.

We now turn to the stability of the algorithm. Global stability in a general network in the presence of feedback delay is an open problem (see [49], [50] for stability analysis for the single-link-single-source case). State-of-the-art results either prove global stability while ignoring feedback delay, or local stability in the presence of feedback delay. Our stability result is restricted to a single link in the absence of delay.

Theorem 2: Suppose there is a single link with capacity c. Then the network defined by (6)–(8) is globally stable, and converges geometrically to the unique equilibrium (w^*, p^*).

The basic idea of the proof is to show that the iteration from $w(t)$ to $w(t+1)$ defined by (6)–(8) is a contraction mapping. Hence $w(t)$ converges geometrically to the unique equilibrium.

Some properties follow from the proof of Theorem 2.

Corollary 3: 1) Starting from any initial point $(w(0), p(0))$, the link is fully utilized, i.e., equality holds in (8), after a finite time.

2) The queue length is lower and upper bounded after a finite amount of time.

VI. Performance

We have conducted some preliminary experiments on our dummynet [51] testbed comparing performance of various new TCP algorithms as well as the Linux TCP implementation. It is important to evaluate them not only in static environments, but also dynamic environments where flows come and go; and not only in terms of end-to-end throughput, but also queue behavior in the network. In this study, we compare performance among TCP connections of the same protocol sharing a single bottleneck link. In summary,

1) FAST TCP achieved the best overall performance in each of the four evaluation criteria: throughput, fairness, responsiveness, and stability.
2) Both HSTCP and STCP improved throughput and responsiveness of Linux TCP, although both showed fairness problems and oscillations with higher frequencies and larger magnitudes.

In the following subsections, we will describe in detail our experimental setup, evaluation criteria, and results.

A. Testbed and kernel instrumentation

We constructed a testbed of a sender and a receiver both running Linux, and an emulated router running FreeBSD. Each testbed machine has dual Xeon 2.66 GHz, 2 GB of main memory, and dual on-board Intel PRO/1000 Gigabit Ethernet interfaces. We have tested these machines to ensure each is able to achieve a peak throughput of 940 Mbps with the standard Linux TCP protocol using `iperf`.

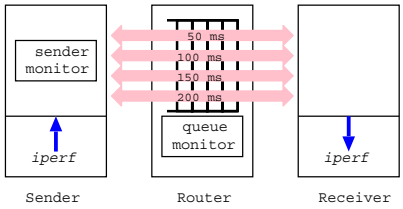


Fig. 5. Testbed and the experimental setup.

Figure 5 shows the setup of the testbed. The testbed router supports paths of various delays and a single bottleneck capacity with a fixed buffer size. It has monitoring capability at the sender and the router. The receiver runs different TCP traffic sinks with different port numbers for connections with different RTTs. We set up and run different experiments from the sender using an automatic script generator to start multiple `iperf` sessions to emulate multiple TCP connections.

Our testbed router ran dummynet [51] under FreeBSD. We configured dummynet to create paths or pipes of different delays, 50, 100, 150, and 200ms, using different destination port numbers on the receiving machine. We then created another pipe to emulate a bottleneck capacity of 800 Mbps and a buffer size of 2,000 packets, shared by all the delay pipes. Due to our need to emulate a high-speed bottleneck capacity, we increased the scheduling granularity of dummynet events. We recompiled the FreeBSD kernel so the task scheduler ran every 1 ms. We also increased the size of the IP layer interrupt

queue (ipintrq) to 3000 to accommodate large bursts of packets.

We instrumented both the sender and the dummynet router to capture relevant information for protocol evaluation. For each connection on the sending machine, the kernel monitor captured the congestion window, the observed baseRTT, and the observed queueing delay. On the dummynet router, the kernel monitor captured the throughput at the dummynet bottleneck, the number of lost packets, and the average queue size every two seconds. We retrieved the measurement data after the completion of each experiment in order to avoid disk I/O that may have interfered with the experiment itself.

We tested four TCP implementations: FAST, HSTCP, STCP, and Reno (Linux implementation). The FAST TCP is based on Linux 2.4.20 kernel, while the rest of the TCP protocols are based on Linux 2.4.19 kernel. We ran tests and did not observe any appreciable difference between the two plain Linux kernels, and the TCP source codes of the two kernels are nearly identical. Linux TCP implementation includes all of the latest RFCs such as New Reno, SACK, D-SACK, and TCP high performance extensions. There are two versions of HSTCP [52], [53]. We present the results of the implementation in [52], but our tests show that the implementation in [53] has comparable performance.

In all of our experiments, the bottleneck capacity is 800 Mbps—roughly 66 packets/ms, and the maximum buffer size is 2000 packets.

We now present our experimental results. We first look at three cases in detail, comparing not only the throughput behavior seen at the source, but also the queue behavior inside the network, by examining trajectories of throughputs, windows, instantaneous queue, cumulative losses, and link utilization. We then summarize the overall performance in a diverse set of experiments in terms of quantitative metrics, defined below, on throughput, fairness, stability, and responsiveness.

B. Case study: static scenario

We present experimental results on aggregate throughput in a simple static environment where, in each experiment, all TCP flows had the same propagation delay and started and terminated at the same times. This set of tests included 20 experiments for different pairing of propagation delays, 50, 100, 150, and 200ms, and the number of identical sources, 1, 2, 4, 8, and 10. We ran this test suite under each of the four TCP protocols. We then constructed a 3-d plot, in Figure 6, for each protocol with the x-axis and y-axis being the number of sources and propagation delay, respectively. The z-axis is the aggregate throughput.

All four protocols performed well when the number of flows was large or the propagation delay was small, i.e., when the window size was small. The performance of FAST TCP remained consistent when these parameters changed. TCP Reno, HSTCP, and STCP had varying degrees of performance degradation as the window size increased, with TCP Reno showing the most significant degradation.

This set of experiments, involving a static environment and identical flows, does not test the fairness, stability and responsiveness of the protocols. We take a close look at these properties next in a dynamic scenario where network equilibrium changes as flows come and go.

C. Case study: dynamic scenario I

In the first dynamic test, the number of flows was small so that throughput per flow, and hence the window size, was large. There were three TCP flows, with propagation delays of 100, 150, and 200ms, that started and terminated at different times, as illustrated in Figures 7(a).

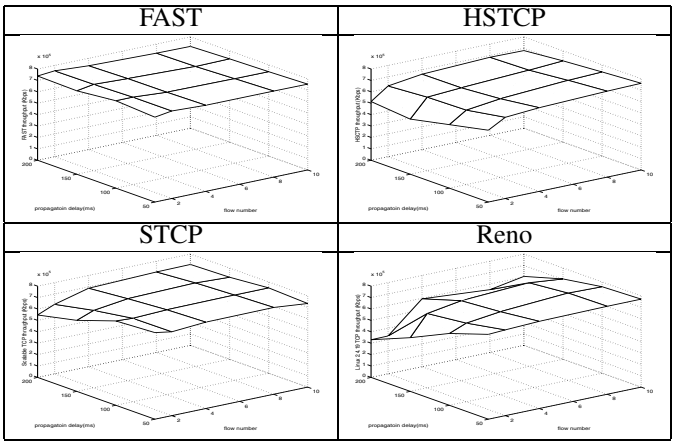


Fig. 6. Static scenario: aggregate throughput.

For each dynamic experiment, we generated two sets of figures, from the sender monitor and the queue monitor. From the sender monitor, we obtained the trajectories of individual connection throughput (in Kbps) and window size (in packets) over time. They are shown in Figure 8.

As new flows joined or old flows left, FAST TCP converged to the new equilibrium rate allocation rapidly and stably. Reno's throughput was also relatively smooth because of the slow (linear) increase before packet losses. The link utilization was low at the end of the experiment when it took 30 minutes for a flow to reclaim the spare capacity due to the departure of another flow. HSTCP and STCP, in an attempt to respond more quickly, went into severe oscillation.

From the queue monitor, we obtained three trajectories: the average queue size (packets), the number of cumulative packet losses (packets), and the utilization of the bottleneck link (in packets/ms), shown in Figure 9 from top to bottom. The queue under FAST TCP was quite small throughout the experiment due to the small number of flows. HSTCP and STCP exhibited strong oscillations that filled the buffer. The link utilizations of FAST TCP and Reno were quite steady, whereas those of HSTCP and STCP showed fluctuations.

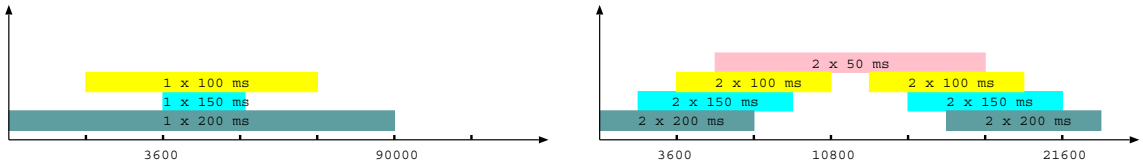
From the throughput trajectories of each protocol, we calculate Jain's fairness index (see Section VI-E for definition) for the rate allocations for each time interval that contains more than one flow (see Figure 7(a)). The fairness indices are shown in Table III. FAST TCP obtained the best fairness,

Time	#Sources	FAST	HSTCP	STCP	Reno
1800 – 3600	2	.967	.927	.573	.684
3600 – 5400	3	.970	.831	.793	.900
5400 – 7200	2	.967	.873	.877	.718

TABLE III
DYNAMIC SCENARIO I: INTRA-PROTOCOL FAIRNESS.

very close to 1, followed by HSTCP, Reno, and then STCP. It confirms that FAST TCP does not penalize against flows with large propagation delays. Even though HSTCP, STCP, and Reno all try to equalize congestion windows among competing connections instead of equalizing rates, this was not achieved as shown in Figure 8. The unfairness is especially severe in the case of STCP, likely due to MIMD as explained in [54].

For FAST TCP, each source tries to maintain the same number of packets in the queue in equilibrium, and thus, in



(a) Dynamic scenario I (3 flows)

(b) Dynamic scenario II (8 flows)

Fig. 7. Dynamic scenario: each colored block represents one or more connections of certain propagation delay. The left and the right edges of each block represent the starting and ending times, respectively, of the flow(s).

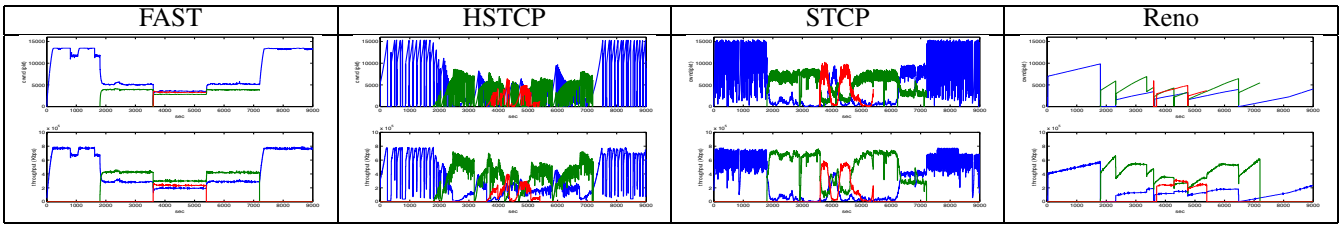


Fig. 8. Dynamic scenario I: throughput and `cwnd` trajectories.

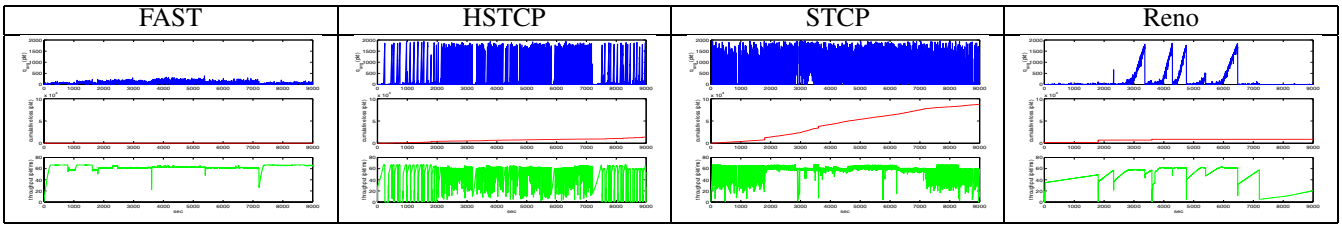


Fig. 9. Dynamic scenario I: dummynet queue sizes, losses, and link utilization.

theory, each competing source should get an equal share of the bottleneck bandwidth. Even though FAST TCP achieved the best fairness index, we did not observe the expected equal sharing of bandwidth (see Figure 8). We found that connections with longer RTTs consistently observed higher queueing delays than those with shorter RTTs. For example, the connection on the path of 100 ms saw an average queueing delay of 6 ms, while the connection on the path of 200 ms saw an average queueing delay of 9 ms. This caused the connection with longer RTTs to maintain fewer packets in the queue in equilibrium, thus getting a smaller share of the bandwidth. We have yet to uncover the source of this problem, but the early conjecture is that when congestion window size is large, it is much harder to break up bursts of packets. With bursty traffic arriving at a queue, each packet would see a delay that includes the transmission times of all preceding packets in the burst. However, if packets were spaced out smoothly, then each packet would have seen a smaller queueing delay at the queue.

D. Case study: dynamic scenario II

This experiment was similar to dynamic scenario I, except that there were a larger number (8) of flows, with different propagation delays, which joined and departed according to the schedule in Figure 7(b). The qualitative behavior in throughput, fairness, stability, and responsiveness for each of the protocols is similar to those in scenario I, and in fact is amplified as the number of flows increases.

Specifically, as the number of competing sources increases in a network, stability becomes worse for the loss-based protocols. As shown in Figures 10 and 11, oscillations in both congestion windows and queue size are more severe for all loss-base protocols. Packet loss is also more severe. The performance of FAST TCP did not degrade in any significant way. Connections sharing the link achieved very similar rates. There was a reasonably stable queue at all times, with little packet loss and high link utilization. Intra-protocol fairness is shown in Table IV, with little change for FAST TCP.

Time	Sources	FAST	HSTCP	STCP	Reno
0 – 1800	2	1.0	.806	.999	.711
1800 – 3600	4	.987	.940	.721	.979
3600 – 5400	6	.976	.808	.631	.978
5400 – 7200	8	.977	.747	.566	.830
7200 – 9000	6	.970	.800	.613	.845
9000 – 10800	4	.989	.906	.636	.885
10800 – 12600	2	.998	.996	.643	.993
12600 – 14400	4	.989	.843	.780	.782
14400 – 16200	6	.944	.769	.613	.880
16200 – 18000	8	.973	.816	.547	.787
18000 – 19800	6	.982	.899	.563	.892
19800 – 21600	4	.995	.948	.668	.896
21600 – 23400	2	1.000	.920	.994	1.000

TABLE IV
FAIRNESS AMONG VARIOUS PROTOCOLS FOR EXPERIMENT II.

E. Overall evaluation

We have conducted several other experiments, with different delays, number of flows, and their arrival and departure patterns. In all these experiments, the bottleneck link capacity was 800Mbps and buffer size 2000 packets. We present in this subsection a summary of protocol performance in terms of some quantitative measures on throughput, fairness, stability, and responsiveness.

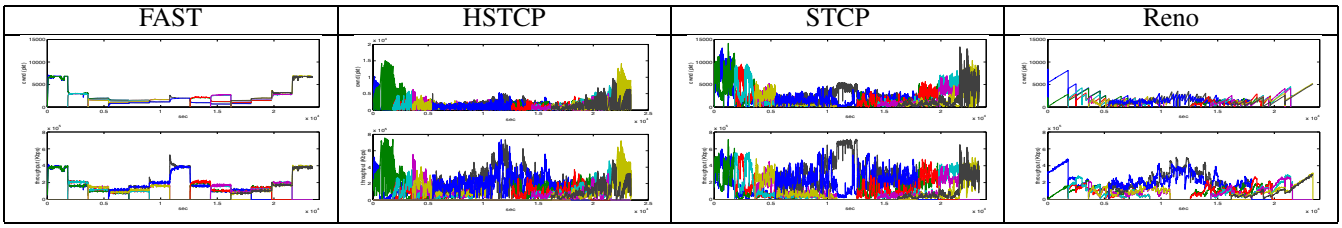


Fig. 10. Dynamic scenario II: throughput and `cwnd` trajectories.

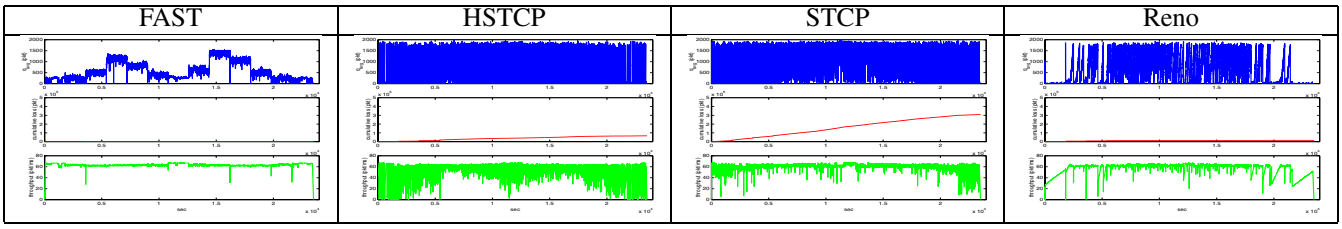


Fig. 11. Dynamic scenario II: dummynet queue sizes, losses, and link utilization.

We use the output of `iperf` for our quantitative evaluation. Each `iperf` session in our experiments produced five-second averages of its throughput. This is the data rate (i.e., goodput) applications such as `iperf` receives, and is slightly less than the bottleneck bandwidth due to IP and Ethernet packet headers.

Let $x_i(k)$ be the average throughput of flow i in the five-second period k. Most tests involved dynamic scenarios where flows joined and departed. For the definitions below, suppose the composition of flows changes in period $k = 1$, remains fixed over period $k = 1, \ldots, m$, and changes again in period $k = m+1$, so that $[1, m]$ is the maximum-length interval over which the same equilibrium holds. Suppose there are n active flows in this interval, indexed by $i = 1, \ldots, n$. Let

$$\overline{x}_i \quad := \quad \frac{1}{m} \sum_{k=1}^{m} x_i(k)$$

be the average throughput of flow i over this interval. We now define our performance metrics for this interval $[1, m]$ using these throughput measurements.

1) **Throughput:** The average aggregate throughput for the interval $[1, m]$ is defined as:[7]

$$E := \sum_{i=1}^{n} \overline{x}_i$$

2) **Intra-protocol fairness:** Jain's fairness index for the interval $[1, m]$ is defined as [55]:

$$F = \frac{\left(\sum_{i=1}^{n} \overline{x}_i\right)^2}{n \sum_{i=1}^{n} \overline{x}_i^2}$$

$F \in (0, 1]$ and $F = 1$ is ideal (equal sharing).

3) **Stability:** The stability index of flow i is the sample standard deviation normalized by the average throughput:

$$S_i \quad := \quad \frac{1}{\overline{x}_i} \sqrt{\frac{1}{m-1} \sum_{k=1}^{m} (x_i(k) - \overline{x}_i)^2}$$

The smaller the stability index, the less oscillation a source experiences. The stability index for interval $[0, m]$ is the average over the n active sources:

$$S \quad := \quad \frac{1}{n} \sum_{i=1}^{n} S_i$$

4) **Responsiveness:** The responsiveness index measures the speed of convergence when network equilibrium changes at $k = 1$, i.e., when flows join or depart. Let $\overline{x}_i(k)$ be the running average by period $k \leq m$:

$$\overline{x}_i(k) \quad := \quad \frac{1}{k} \sum_{t=1}^{k} x_i(k)$$

Then $\overline{x}_i(m) = \overline{x}_i$ is the average over the entire interval $[1, m]$.

Responsiveness index R_1 measures how fast the running average $\overline{x}_i(k)$ of the *slowest* source converges to $\overline{x}_i$:[8]

$$R_1 \quad := \quad \max_i \max \left\{ k : \left| \frac{\overline{x}_i(k) - \overline{x}_i}{\overline{x}_i} \right| > 0.1 \right\}$$

For each TCP protocol, we obtain one set of computed values for each evaluation criterion for all of our experiments. We plot the CDF (cumulative distribution function) of each set of values. These are shown in Figures 12 – 15.

From Figures 12–15, FAST has the best performance among all protocols under each evaluation criterion. More importantly, the variation in each of the distributions is smaller under FAST than under the other protocols, suggesting that FAST had fairly consistent performance in our test scenarios. We also observe that both HSTCP and STCP achieved higher throughput and improved responsiveness compared with TCP Reno. STCP had worse intra-protocol fairness compared with TCP Reno, while HSTCP achieved comparable intra-protocol fairness to Reno (see Figures 13, 8 and 10). Both HSTCP and STCP showed increased oscillations compared with Reno

[7]As mentioned above, this is the throughput (or goodput) seen at the *application* layer, not TCP layer.

[8]The natural definition of responsiveness index as the earliest period after which the throughput $x_i(k)$ (as opposed to the running average $\overline{x}_i(k)$ of the throughput) stays within 10% of its equilibrium value is unsuitable for TCP protocols that do not stabilize into an equilibrium value. Hence we define it in terms of $\overline{x}_i(k)$ which, by definition, always converges to $\overline{x}_i$ by the end of the interval $k = m$. This definition captures the intuitive notion of responsiveness if $x_i(k)$ settles into a periodic limit cycle.

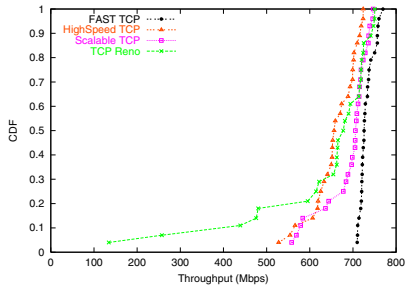


Fig. 12. Overall evaluation: throughput.

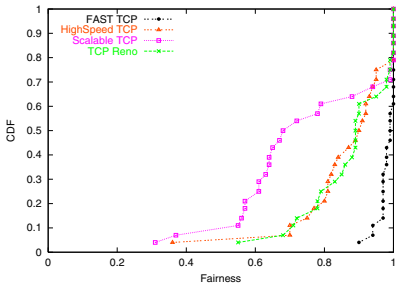


Fig. 13. Overall evaluation: fairness.

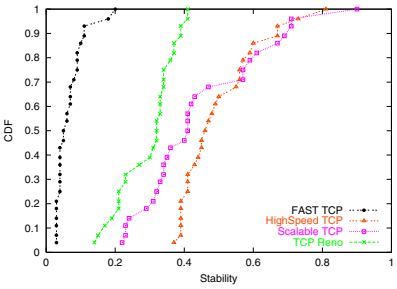


Fig. 14. Overall evaluation: stability.

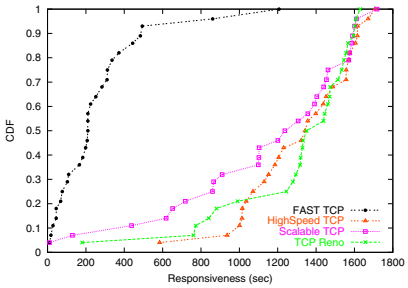


Fig. 15. Overall evaluation: responsiveness index R_1.

(Figures 14, 8 and 9), and the oscillations became worse as the number of sources increased (Figures 10 and 11).

From Figure 15, FAST TCP achieved a much better responsiveness index R_1 (which is based on worst case individual throughput) than the other schemes. We caution however that it can be hard to quantify "responsiveness" for protocols that do not stabilize into an equilibrium point or a periodic limit cycle, and hence the unresponsiveness of Reno, HSTCP, and STCP, as measured by index R_1, should be interpreted with care.

VII. Conclusion

We have described an alternative congestion control algorithm, FAST TCP, that addresses the four main problems of TCP Reno in networks with high capacities and large latencies. FAST TCP has a log utility function and achieves weighted proportional fairness. Its window adjustment is equation-based, under which the network moves rapidly toward equilibrium when the current state is far away and slows down when it approaches the equilibrium. FAST TCP uses queueing delay, in addition to packet loss, as a congestion signal. Queueing delay provides a finer measure of congestion and scales naturally with network capacity.

We have presented experimental results of our first Linux prototype and compared its performance with TCP Reno, HSTCP, and STCP. We have evaluated these algorithms not only in static environments, but also dynamic environments where flows come and go, and not only in terms of end-to-end throughput, but also queue behavior in the network. In these experiments, HSTCP and STCP achieved better throughput and link utilization than Reno, but their congestion windows and network queue lengths had significant oscillations. TCP Reno produced less oscillation, but at the cost of lower link utilization when sources departed. FAST TCP, on the other hand, consistently outperforms these protocols in terms of throughput, fairness, stability, and responsiveness.

Acknowledgments: We gratefully acknowledge the contributions of the FAST project team and our collaborators, at `http://netlab.caltech.edu/FAST/`, in particular, G. Almes, J. Bunn, D. H. Choe, R. L. A. Cottrell, V. Doraiswami, J. C. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, S. Shalunov, S. Singh, J. Wang, Z. Wang, S. Yip. This work is funded by NSF (grants ANI-0113425 and ANI-0230967), Caltech Lee Center for Advanced Networking, ARO (grant DAAD19-02-1-0283), AFOSR (grant F49620-03-1-0119), and Cisco.

References

[1] V. Jacobson, "Congestion avoidance and control," *Proceedings of SIGCOMM'88, ACM*, August 1988, An updated version is available via `ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z`.

[2] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018, `ftp://ftp.isi.edu/in-notes/rfc2018.txt`, October 1996.

[3] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," RFC 1323, `ftp://ftp.isi.edu/in-notes/rfc1323.txt`, May 1992.

[4] Janey Hoe, "Improving the startup behavior of a congestion control scheme for tcp," in *ACM Sigcomm'96*, August 1996, `http://www.acm.org/sigcomm/sigcomm96/program.html`.

[5] Cheng Jin, David X. Wei, and Steven H. Low, "The case for delay-based congestion control," in *Proc. of IEEE Computer Communication Workshop (CCW)*, October 2003.

[6] Sally Floyd, "HighSpeed TCP for large congestion windows," Internet draft draft-floyd-tcp-highspeed-02.txt, work in progress, `http://www.icir.org/floyd/hstcp.html`, February 2003.

[7] Tom Kelly, "Scalable TCP: Improving performance in highspeed wide area networks," Submitted for publication, `http://www-lce.eng.cam.ac.uk/~ctk21/scalable/`, December 2002.

[8] Lawrence S. Brakmo and Larry L. Peterson, "TCP Vegas: end-to-end congestion avoidance on a global Internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–80, October 1995, `http://cs.princeton.edu/nsg/papers/jsac-vegas.ps`.

[9] E. Weigle and W. Feng, "A case for TCP Vegas in high-performance computational grids," in *Proceedings of the 9th International Symposium on High Performance Distributed Computing (HPDC'01)*, August 2001.

[10] W. Feng and S. Vanichpun, "Enabling compatibility between TCP Reno and TCP Vegas," *IEEE Symposium on Applications and the Internet (SAINT 2003)*, January 2003.

[11] C. Casetti, M. Gerla, S. Mascolo, M. Sansadidi, and R. Wang, "TCP Westwood: end-to-end congestion control for wired/wireless networks," *Wireless Networks Journal*, vol. 8, pp. 467–479, 2002.

[12] R. Wang, M. Valla, M. Sanadidi, B. Ng, and M. Gerla, "Using adaptive rate estimation to provide enhanced and robust transport over heterogeneous networks," in *Proc. of IEEE ICNP*, 2002.

[13] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high-bandwidth delay product networks," in *Proc. ACM Sigcomm*, August 2002.

[14] Shudong Jin, Liang Guo, Ibrahim Matta, and Azer Bestavros, "A spectrum of TCP-friendly window-based congestion control algorithms," *IEEE/ACM Transactions on Networking*, vol. 11, no. 3, June 2003.

[15] R. Shorten, D. Leith, J. Foy, and R. Kilduff, "Analysis and design of congestion control in synchronised communication networks," in *Proc. of 12th Yale Workshop on Adaptive and Learning Systems*, May 2003, `www.hamilton.ie/doug_leith.htm`.

[16] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control for fast long-distance networks," in *Proc. of IEEE Infocom*, 2004.

[17] A. Kuzmanovic and E. Knightly, "TCP-LP: a distributed algorithm for low priority data transfer," in *Proc. of IEEE Infocom*, 2003.

[18] H. Bullot and L. Cottrell, "Tcp stacks testbed," `http://www-iepm.slac.stanford.edu/bw/tcp-eval/`.

[19] Raj Jain, "A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks," *ACM Computer Communication Review*, vol. 19, no. 5, pp. 56–71, Oct. 1989.

[20] Z. Wang and J. Crowcroft, "Eliminating periodic packet losses in the 4.3-Tahoe BSD TCP congestion control algorithm," *ACM Computer Communications Review*, April 1992.

[21] Jim Martin, Arne Nilsson, and Injong Rhee, "Delay-based congestion avoidance for TCP," *IEEE/ACM Trans. on Networking*, vol. 11, no. 3, pp. 356–369, June 2003.

[22] Fernando Paganini, John C. Doyle, and Steven H. Low, "Scalable laws for stable network congestion control," in *Proceedings of Conference on Decision and Control*, December 2001, `http://www.ee.ucla.edu/~paganini`.

[23] Hyojeong Choe and Steven H. Low, "Stabilized Vegas," in *Proc. of IEEE Infocom*, April 2003, `http://netlab.caltech.edu`.

[24] Fernando Paganini, Zhikui Wang, Steven H. Low, and John C. Doyle, "A new TCP/AQM for stability and performance in fast networks," in *Proc. of IEEE Infocom*, April 2003, `http://www.ee.ucla.edu/~paganini`.

[25] Jeonghoon Mo and Jean Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp. 556–567, October 2000.

[26] Steven H. Low, Larry Peterson, and Limin Wang, "Understanding Vegas: a duality model," *J. of ACM*, vol. 49, no. 2, pp. 207–235, March 2002, `http://netlab.caltech.edu`.

[27] Frank P. Kelly, Aman Maulloo, and David Tan, "Rate control for communication networks: Shadow prices, proportional fairness and stability," *Journal of Operations Research Society*, vol. 49, no. 3, pp. 237–252, March 1998.

[28] Frank P. Kelly, "Mathematical modelling of the Internet," in *Mathematics Unlimited - 2001 and Beyond*, B. Engquist and W. Schmid, Eds., pp. 685–702. Springer-Verlag, Berlin, 2001.

[29] C.V. Hollot, V. Misra, D. Towsley, and W.B. Gong, "Analysis and design of controllers for AQM routers supporting TCP flows," *IEEE Transactions on Automatic Control*, vol. 47, no. 6, pp. 945–959, 2002.

[30] S. H. Low, F. Paganini, J. Wang, and J. C. Doyle, "Linear stability of TCP/RED and a scalable control," *Computer Networks Journal*, vol. 43, no. 5, pp. 633–647, 2003, `http://netlab.caltech.edu`.

[31] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *ACM Computer Communication Review*, vol. 27, no. 3, July 1997, `http://www.psc.edu/networking/papers/model_ccr97.ps`.

[32] T. V. Lakshman and Upamanyu Madhow, "The performance of TCP/IP for networks with high bandwidth–delay products and random loss," *IEEE/ACM Transactions on Networking*, vol. 5, no. 3, pp. 336–350, June 1997, `http://www.ece.ucsb.edu/Faculty/Madhow/Publications/ton97.ps`.

[33] C. Jin, D. Wei, and S. H. Low, "FAST TCP: motivation, architecture, algorithms, performance," Tech. Rep. CaltechCSTR:2003.010, Caltech, Pasadena CA, 2003, `http://netlab.caltech.edu/FAST`.

[34] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proc. ACM SIGCOMM'00*, September 2000.

[35] Steven H. Low and David E. Lapsley, "Optimization flow control, I: basic algorithm and convergence," *IEEE/ACM Transactions on Networking*, vol. 7, no. 6, pp. 861–874, December 1999, `http://netlab.caltech.edu`.

[36] S. Kunniyur and R. Srikant, "End-to-end congestion control: utility functions, random losses and ECN marks," *IEEE/ACM Transactions on Networking*, 2003.

[37] L. Massoulie and J. Roberts, "Bandwidth sharing: objectives and algorithms," *IEEE/ACM Transactions on Networking*, vol. 10, no. 3, pp. 320–328, June 2002.

[38] H. Yaiche, R. R. Mazumdar, and C. Rosenberg, "A game theoretic framework for bandwidth allocation and pricing in broadband networks," *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, October 2000.

[39] Steven H. Low, "A duality model of TCP and queue management algorithms," *IEEE/ACM Trans. on Networking*, vol. 11, no. 4, pp. 525–536, August 2003, `http://netlab.caltech.edu`.

[40] Glenn Vinnicombe, "On the stability of networks operating TCP-like congestion control," in *Proc. of IFAC World Congress*, 2002.

[41] S. Kunniyur and R. Srikant, "A time-scale decomposition approach to adaptive ECN marking," *IEEE Transactions on Automatic Control*, June 2002.

[42] S. Kunniyur and R. Srikant, "Designing AVQ parameters for a general topology network," in *Proceedings of the Asian Control Conference*, September 2002.

[43] Glenn Vinnicombe, "Robust congestion control for the Internet," submitted for publication, 2002.

[44] Steven H. Low, Fernando Paganini, and John C. Doyle, "Internet congestion control," *IEEE Control Systems Magazine*, vol. 22, no. 1, pp. 28–43, February 2002.

[45] S. H. Low and R. Srikant, "A mathematical framework for designing a low-loss, low-delay internet," 2003.

[46] Frank P. Kelly, "Fairness and stability of end-to-end congestion control," *European Journal of Control*, vol. 9, pp. 159–176, 2003.

[47] C. Jin, D. Wei, S. H. Low, G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle, H. Newman, F. Paganini, S. Ravot, and S. Singh, "FAST Kernel: Background theory and experimental results," in *First International Workshop on Protocols for Fast Long-Distance Networks*, February 2003.

[48] David X. Wei and Steven H. Low, "A model for TCP model with burstiness effect," Submitted for publication, 2003.

[49] Zhikui Wang and Fernando Paganini, "Global stability with time delay in network congestion control," in *Proc. of the IEEE Conference on Decision and Control*, December 2002.

[50] S. Deb and R. Srikant, "Global stability of congestion controllers for the Internet," *IEEE Transactions on Automatic Control*, vol. 48, no. 6, pp. 1055–1060, June 2003.

[51] Luigi Rizzo, "Dummynet," `http://http://info.iet.unipi.it/~luigi/ip_dummynet/`.

[52] Y. Li, "Implementing highspeed tcp," `URL:http://www.hep.ucl.ac.uk/~ytl/tcpip/hstcp/index.html`.

[53] T. Dunigan, "Floyd's tcp slow-start and aimd mods," `URL:http://www.csm.ornl.gov/~dunigan/net100/floyd.html`.

[54] D. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Computer Networks*, vol. 17, pp. 1–14, 1989.

[55] R. Jain, *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation and modeling.*, John Wiley and Sons, Inc., 1991.