FAST TCP:
Motivation, Architecture, Algorithms, Performance

David X. Wei Cheng Jin Steven H. Low Sanjay Hegde
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 summarize some of the algorithms implemented in our prototype. We characterize the equilibrium and stability properties of FAST TCP. We provide experimental evaluation of our first prototype in terms of throughput, fairness, stability, and responsiveness.**

I. INTRODUCTION AND SUMMARY

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 and fair sharing is desired. These constraints – unpredictable supply and demand and the desire for efficient distributed 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 in this paper, was developed in 1988 [20] and has gone through several enhancements since, e.g., [21], [58], [47], [18], [1], [14]. 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 in congestion window 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, and motivate a delay-based solution. Delay-based congestion control has been proposed, e.g., in [23], [69], [3], [72], [12]. See [5], [68], [28], [27], [56], [75], [36], [4] for other recent proposals.

Using queueing delay as a 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 under TCP Reno (e.g., probability on the order 10^{-7} 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, based on the commonly used ordinary differential equation model of TCP/AQM, the dynamics of queueing delay has the right scaling with respect to network capacity. This helps maintain stability as a network scales up in capacity [51], [8], [53].

In Section III, we lay out an architecture to implement our design, and present an overview of some of the algorithms implemented in our current prototype. 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.

We evaluate FAST TCP both analytically and experimentally. In Section III-B, we present a mathematical model of the window control algorithm. We prove that FAST TCP has the same equilibrium properties as TCP Vegas [50], [44]. In particular, it does not penalize flows with large propagation delays and it achieves weighted proportional fairness [31]. For the special case of single bottleneck link with heterogeneous flows, we prove that the window control algorithm of FAST is locally asymptotically stable, in the absence of feedback delay.

In Section IV, we present both experimental and simulation 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. It is important to evaluate a congestion control algorithm not only in terms of throughput achieved, but also what it does to network queues and how that affects other applications sharing the same queue. We compare the performance of FAST TCP with Reno, HSTCP (HighSpeed TCP [15]), STCP (Scalable TCP [32]), and BIC TCP [75], using their default parameters.

To appear in IEEE/ACM Transactions on Networking, 2007. An abridged version appears in [25] and an expanded version in [24].

In Section V, we summarize open issues and provide references for proposed solutions.

II. Motivations

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 [15], STCP [32], and FAST TCP are explicitly guided by flow-level goals.

A. Packet and flow level modeling

The congestion avoidance algorithm of TCP Reno and its variants have the form of AIMD [20]. The pseudo code for window adjustment is:

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

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., [29], [19], [39], [41]. The window $w_i(t)$ of source i increases by 1 packet per RTT,[1] and decreases per unit time by

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

where $x_i(t) := w_i(t)/T_i(t)$ pkts/sec. $T_i(t)$ is the round-trip time, and $q_i(t)$ 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:

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

Setting $\dot{w}_i(t) = 0$ in (1) yields the well-known $1/\sqrt{q}$ formula for TCP Reno discovered in [48], [37], which relates loss probability to window size in equilibrium:

$$q_i^* \quad = \quad \frac{3}{2 w_i^{*2}} \tag{2}$$

In summary, (1) and (2) describe the flow-level dynamics and equilibrium, respectively, for TCP Reno.

Even though Reno, HSTCP, STCP, and FAST look different at the packet level, they have similar equilibrium and dynamic structures at the flow level; see [24] for detailed derivations. The congestion windows in these algorithms all evolve according to:

$$\dot{w}_i(t) \quad = \quad \kappa_i(t) \cdot \left(1 - \frac{q_i(t)}{u_i(t)} \right) \tag{3}$$

where $\kappa_i(t) := \kappa_i(w_i(t), T_i(t))$ and $u_i(t) := u_i(w_i(t), T_i(t))$. 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 q_i. Within this structure, at the flow level, there are thus 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.
- $u_i(w_i, T_i)$: the choice of the marginal utility function u_i determines equilibrium properties such as the equilibrium rate allocation and its fairness.
- q_i: in the absence of explicit feedback, the choice of congestion measure q_i is limited to loss probability or queueing delay. The dynamics of $q_i(t)$ is determined inside the network.

At the flow level, a goal is to design a class of function pairs, $u_i(w_i, T_i)$ and $\kappa_i(w_i, T_i)$, so that the feedback system described by (3), together with link dynamics of $q_i(t)$ and the interconnection, has an equilibrium that is fair and efficient, and that the equilibrium is stable, in the presence of feedback delay. The design choices in FAST, Reno, HSTCP, and STCP are shown in Table I. These choices produce equilibrium characterizations shown in Table II.

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

TABLE I

Common dynamic structure: w_i is source i’s window size, T_i is its round-trip time, q_i is congestion measure, $x_i = w_i/T_i$; $a, b(w_i), \rho, \gamma, \alpha_i, \tau$ are protocol parameters; see [24].

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

TABLE II

Common equilibrium structure: x_i is source i’s throughput in packets/sec, T_i is equilibrium round-trip time, q_i is end-to-end congestion measure in equilibrium. The parameters are: $\alpha = 1.225$ for Reno, $\alpha = 0.120$ for HSTCP, and $\alpha = 0.075$ for STCP. For FAST, α_i should vary with link capacity.

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

B. Reno’s problems at large window

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

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

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

increases. Indeed, from (2), $q_i^* w_i^* = 1.5/w_i^*$, i.e., the average number of packet losses (or loss events) per window decreases in inverse proportion to the equilibrium window size for Reno. From Table II, this number for HSTCP is $q_i^* w_i^* = 0.0789/w_i^{*\,0.1976}$. Hence it also decreases with the equilibrium window, but more slowly than for TCP Reno. For STCP, this number is $q_i^* w_i^* = a(1-b/2)/b$, which is independent of, and hence scalable with, the equilibrium window size. The recommended values in [32] for the constants are $a = 0.01$ and $b = 0.125$, yielding an average loss of 0.075 per window. Even though equilibrium is a flow-level notion, this problem with Reno manifests itself at the packet level, where a source increases its window too slowly and decreases it too drastically. In contrast, HSTCP and STCP increase more aggressively and decrease less drastically.

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 in congestion windows and bottleneck queues, 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 [19], [39]. These must be addressed by different means.

Congestion window can be stabilized only if multi-bit feedback is used.[3] This is the approach taken by the equation-based algorithm in [13], where congestion window is adjusted based on the estimated loss probability in an attempt to stabilize around a target value given by (2). 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 [19], [39]. The instability at the flow level can lead to severe oscillations that can be reduced *only* by stabilizing the flow level dynamics. We present a delay-based approach to address these problems.

C. Delay-based approach

The common model (3) 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 $q_i(t)$. This interpretation immediately suggests an equation-based packet-level implementation where the window adjustment $\dot{w}_i(t)$ depends on not only the sign, but also the magnitude of the difference between the ratio $q_i(t)/u_i(t)$ 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 $q_i(t)$.

Without explicit feedback, $q_i(t)$ can only be loss probability, as used in TFRC [13], or queueing delay, as used in TCP Vegas [3] and FAST TCP. Queueing delay can be more accurately estimated than loss probability both because loss samples provide coarser information than queueing delay samples, and because packet losses in networks with large bandwidth-delay products tend to be rare events under schemes such as Reno. 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 facilitates 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 have the right scaling with respect to network capacity, according to the commonly used ordinary differential equation model. This helps maintain stability as network capacity grows [51], [8], [53], [52].

It has been found that delay and packet loss can have a weak correlation, e.g., [45], especially when packet losses can be caused by other reasons than buffer overflow. This does not mean that it is futile to use delay as a measure of congestion, but rather, that using delay to predict loss in the hope of helping a loss-based algorithm adjust its window is a wrong approach to address problems at large windows. A different approach that fully exploits delay as a congestion measure, augmented with loss information, is needed.

This motivates the following implementation strategy. First, by explicitly estimating how far the current state $q_i(t)/u_i(t)$ is from the equilibrium value of 1, a delay-based 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*. 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) in [24]). Like the equation-based scheme in [13], 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 $q_i(t)$ allows the network to stabilize in the region below the overflowing point, 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 estimate, the algorithm must adapt its parameter α_i to 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 emerging theory of large-scale networks under end-to-end control, e.g., [31], [42], [35], [50], [46], [76], [44], [41], [6], [51], [63], [34], [19], [39], [33], [64], [53], [8], [52], [73], [11], [56] (see also, e.g., [43], [40], [30], [57] 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.

In the next section, we lay out the architecture of FAST TCP.

[3]See [70] for discussion on congestion signal and decision function.

III. Architecture and Algorithms

A. Architecture

We separate the congestion control mechanism of TCP into four components in Figure 1. These four components are functionally independent so that they can be designed separately and upgraded asynchronously.

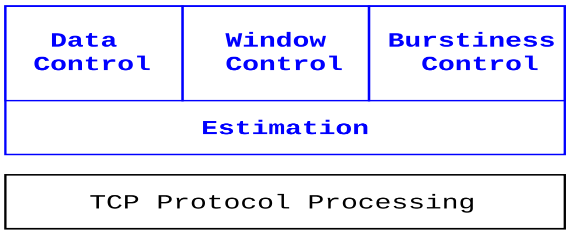

Fig. 1. FAST TCP architecture.

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.

More specifically, the estimation component computes two pieces of feedback information for each data packet sent – a multibit queueing delay and an one-bit loss-or-no-loss indication – which are used by the other three components. Data control selects the next packet to send from three pools of candidates: new packets, packets that are deemed lost (negatively acknowledged), and transmitted packets that are not yet acknowledged. Window control regulates packet transmission at the RTT timescale, while burstiness control works at a smaller timescale. Burstiness control smoothes out transmission of packets in a fluid-like manner to track the available bandwidth. We employ two mechanisms, one to supplement self-clocking in streaming out individual packets and the other to increase window size smoothly in smaller bursts. *Burstiness reduction* limits the number of packets that can be sent when an ack advances congestion window by a large amount. *Window pacing* determines how to increase congestion window over the idle time of a connection to the target determined by the window control component. It reduces burstiness with a reasonable amount of scheduling overhead. For details of these two mechanisms, see [71], [24].

An initial prototype that included some of these features was demonstrated in November 2002 at the SuperComputing Conference, and the experimental results were reported in [26]. In the following, we explain in detail the design of the window control component.

B. Window control algorithm

FAST 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:

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

where $\gamma \in (0,1]$, $\mathsf{baseRTT}$ is the minimum RTT observed so far, and α is a positive protocol parameter that determines the total number of packets queued in routers in equilibrium along the flow's path. The window update period is 20ms in our prototype.

We now provide an analytical evaluation of FAST TCP. We present a model of the window control algorithm for a network of FAST flows. We show that, in equilibrium, the vectors of source windows and link queueing delays are the unique solutions of a pair of optimization problems (6)–(7). This completely characterizes the network equilibrium properties such as throughput, fairness, and delay. We also present a preliminary stability analysis.

We model a network as a set of resources with finite capacities c_l, e.g., transmission links, processing units, memory, etc., to which we refer to 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 its window $w_i(t)$ periodically according to: [4]

$$w_i(t+1) \quad = \quad \gamma\left(\frac{d_i w_i(t)}{d_i + q_i(t)} + \alpha_i\right) + (1-\gamma) w_i(t) \quad (4)$$

where $\gamma \in (0,1]$, at time t.

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: within 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 [66] 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:

$$\sum_i R_{li} \frac{w_i}{d_i + q_i(t)} \quad \begin{cases} = c_l & \text{if } p_l(t) > 0 \\ \leq c_l & \text{if } p_l(t) = 0 \end{cases} \quad (5)$$

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

The next result says that the queueing delay is indeed well defined. All proofs are relegated to the Appendix and [24].

Lemma 1: Suppose the routing matrix R has full row rank. Given $w = (w_i, \forall i)$, there exists a unique queueing delay vector $p = (p_l, \forall l)$ that satisfies (5).

The equilibrium values of windows w^* and delays p^* of the network defined by (4)–(5) can be characterized as follows. Consider the utility maximization problem

$$\max_{x \geq 0} \sum_i \alpha_i \log x_i \quad \text{s.t.} \quad Rx \leq c \quad (6)$$

[4]Note that (4) 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 [44], TCP Vegas updates its window according to

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

where $\operatorname{sgn}(z) = -1$ 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.

and the following (dual) problem:

$$\min_{p\geq 0} \quad \sum_l c_l p_l - \sum_i \alpha_i \log \sum_l R_{li} p_l \tag{7}$$

Theorem 2: Suppose R has full row rank. The unique equilibrium point (w^*, p^*) of the network defined by (4)–(5) exists and is such that $x^* = (x_i^* := w_i/(d_i + q_i^*), \forall i)$ is the unique maximizer of (6) and p^* is the unique minimizer of (7). This implies in particular that the equilibrium rate x^* is α_i-weighted proportionally fair.

Theorem 2 implies that FAST TCP has the same equilibrium properties as TCP Vegas [50], [44]. Its equilibrium throughput is given by

$$x_i \;=\; \frac{\alpha_i}{q_i} \tag{8}$$

In particular, it does not penalize sources with large propagation delays d_i. The relation (8) also implies that, in equilibrium, source i maintains α_i packets in the buffers along its path [50], [44]. 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.

Theorem 3 (Single-link heterogeneous-source): Suppose there is only a single link with capacity c. Then the system defined by (4)–(5) is locally asymptotically stable.

The basic idea of the proof is to show that the mapping from (scaled) $w(t)$ to $w(t+1)$ defined by (4)–(5) has a Jacobian whose spectral radius is strictly less than 1, uniformly in w; see Theorem 6 in the Appendix. Hence $w(t)$ converges locally to the unique equilibrium. The proof technique seems to be different from those in the current literature of TCP congestion control. It also reveals some interesting global properties of FAST TCP at a single link.

Corollary 4: Suppose there is only a single link with capacity c.

1) The equilibrium point (w^*, p^*) is given by

$$w_i^* \;=\; \alpha_i + \frac{\alpha_i c}{\sum_j \alpha_j} d_i, \qquad p \;=\; \frac{1}{c}\sum_i \alpha_i$$

with $x_i^* = c\alpha_i / \sum_j \alpha_j$.

2) Starting from any initial point $(w(0), p(0))$, the link is fully utilized, i.e., equality holds in (5), after a finite time.

3) The queue length is lower and upper bounded after a finite time. If all sources have the same propagation delay, $d_i = d$ for all i, then the system converges in finite time.

The stability result reported here is limited to local asymptotic stability at a single link with heterogeneous sources and feedback delay is ignored. In [65], the local stability result is extended to a multilink network in the absence of feedback delay. With feedback delay, local stability can be maintained for the case of a single link, provided the heterogeneity of the delays is small. This delayed stability result is extended in [66] to a multilink network; furthermore, global stability at a single link is established in the absence of delay using a Lyapunov argument. These results are summarized in [67]. In [9], a condition is derived under which a single-source single-link network is globally aysmptotically stable under FAST.

IV. PERFORMANCE

We have conducted experiments on our dummynet [55] testbed comparing performance of various new TCP algorithms as well as the Linux TCP implementation. For more complex scenarios that are hard to reliably emulate with our dummynet testbed, we report some simulation results on NS-2 [77].

The experiment and simulation results reported aim to zoom in on specific properties of FAST. These scenarios may be incomplete or unrealistic. Experiments in production networks can be found in [17], [26]. Other results not presented in this paper due to space limitation are collected in [70] and [79].

A. Testbed and kernel instrumentation

Our testbed consists of a sender and a receiver both running Linux, that are connected through an emulated router running dummynet under FreeBSD. Each testbed machine has dual Xeon 2.66 GHz CPUs, 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` . 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 multiple `iperf` sinks with different port numbers for connections with different RTTs. We configured dummynet to create paths or pipes of different RTTs, 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. To reduce scheduling granularity, we recompiled the FreeBSD kernel so the task scheduler runs every 1 ms. We also increased the size of the IP layer interrupt queue to 3000 to accommodate large bursts of packets. For each connection on the sending machine, the kernel monitor captures the congestion window, the observed baseRTT, and the observed queueing delay. On the dummynet router, the kernel monitor captures the throughput at the bottleneck, the number of lost packets, and the average queue size every two seconds.

We tested five TCP implementations: FAST, Reno (Linux), HSTCP, STCP, and BIC-TCP using their default parameters for all experiments. The FAST TCP is based on Linux 2.4.20 kernel (α is set to 200 packets), HSTCP, Scalable TCP and Reno are based on Linux 2.4.19 kernel, BIC TCP was based on 2.4.25 kernel. We ran tests and did not observe any appreciable difference among the three plain Linux kernels, and the TCP source codes of the three 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 [38], [10]. We present the results of the implementation in [38], but our tests show that the implementation in [10] has comparable performance.

In all of our experiments, the bottleneck capacity is 800 Mbps—roughly 67 packets/ms, and the maximum buffer size is 2000 packets.

We now present our experimental results. We first look at two 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, 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: 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 Figure 2.

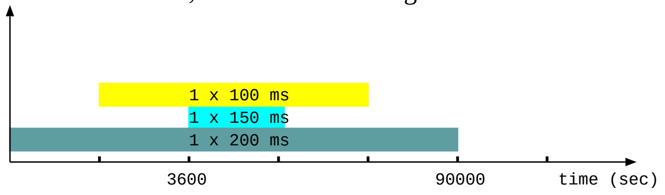

Fig. 2. Dynamic scenario I (3 fbws): active periods.

For each dynamic experiment, we generated two sets of figures. From the sender monitor, we obtained the trajectory of individual connection throughput (in Kbps) over time. They are shown in Figure 3. As new flows joined or old flows left, FAST TCP converged to the new equilibrium rate allocation rapidly and stably (left column). Reno's throughput was also relatively smooth because of the slow (linear) increase between packet losses. It incurred inefficiency towards the end of the experiment when it took 30 minutes for a flow to consume the spare capacity made available by the departure of another flow. HSTCP, STCP, and BIC-TCP responded more quickly but also exhibited significant fluctuation in throughput.

From the queue monitor, we obtained average queue size (packets) shown in Figure 3 on the right column. The queue under FAST TCP was quite small throughout the experiment because the number of flows was small. HSTCP, STCP, and BIC-TCP exhibited strong oscillations that filled the buffer. Since BIC-TCP tried to maintain an aggregate window to be just below the point where overflow occurs, it had the highest average queue length.

From the throughput trajectories of each protocol, we calculate Jain's fairness indices (see Section IV-D for definition) for the rate allocations for each time interval that contains more than one flow (see Figure 2). The fairness indices are shown in Table III. FAST TCP obtained the best intra-protocol fairness, very close to 1, followed by HSTCP, Reno, BIC-TCP, and then STCP. It confirms that FAST TCP does not penalize flows with large propagation delays.

Time (se	#Sources	FAST	Reno	HSTCP	STCP	BIC
1800 – 3600	2	.967	.684	.927	.573	0.683
3600 – 5400	3	.970	.900	.831	.793	0.687
5400 – 7200	2	.967	.718	.873	.877	0.704

TABLE III
DYNAMIC SCENARIO I: INTRA-PROTOCOL FAIRNESS (JAIN'S INDEX).

For FAST TCP, each source tries to maintain the same number of packets in the queue in equilibrium, and thus, in 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 3). Our sender monitor showed that all the flows measured their propagation delays correctly. 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 conjecture that a larger window size (due to longer RTT) produces a more bursty traffic. 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, leading to a larger average queueing delay and a smaller throughput.

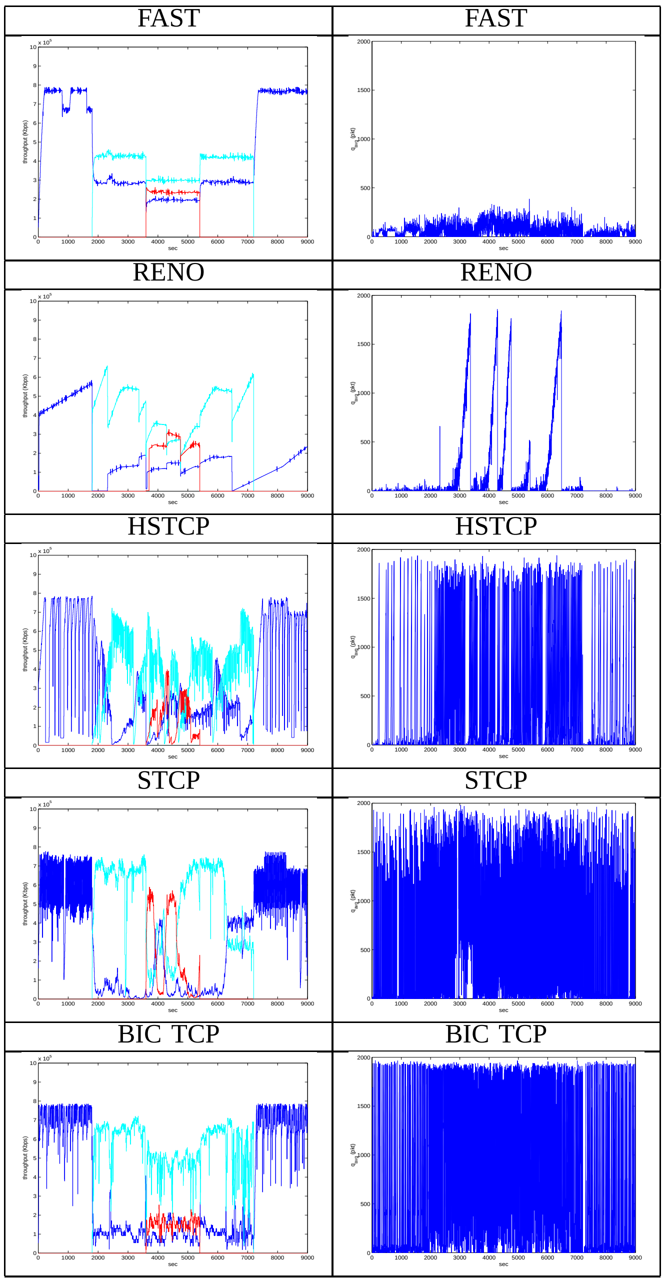

Fig. 3. Dynamic scenario I: Throughput trajectory (left column) and Dummynet queue trajectory (right column).

C. 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 4. The qualitative behavior in throughput, fairness, stability, and responsiveness for each of the protocols is similar in this case as in scenario I, and in fact is amplified as the number of flows increases.

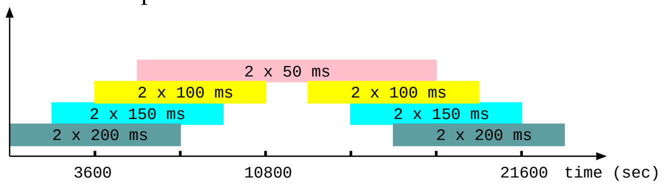

Fig. 4. Dynamic scenario II (8 flows): active periods.

Specifically, as the number of competing sources increases, stability became worse for the loss-based protocols. As shown in Figure 5, oscillations in both throughput and queue size are more severe for loss-base protocols. Packet loss was 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 no significant variation in the fairness of FAST TCP.

Time (sec)	Sources	FAST	Reno	HSTCP	STCP	BIC
0 – 1800	2	1.000	.711	.806	.999	.979
1800 – 3600	4	.987	.979	.940	.721	.971
3600 – 5400	6	.976	.978	.808	.631	.876
5400 – 7200	8	.977	.830	.747	.566	.858
7200 – 9000	6	.970	.845	.800	.613	.856
9000 – 10800	4	.989	.885	.906	.636	.973
10800 – 12600	2	.998	.993	.996	.643	1.000
12600 – 14400	4	.989	.782	.843	.780	.936
14400 – 16200	6	.944	.880	.769	.613	.905
16200 – 18000	8	.973	.787	.816	.547	.779
18000 – 19800	6	.982	.892	.899	.563	.894
19800 – 21600	4	.995	.896	.948	.668	.948
21600 – 23400	2	1.000	1.000	.920	.994	.998

TABLE IV
DYNAMIC SCENARIO II: INTRA-PROTOCOL FAIRNESS (JAIN'S INDEX).

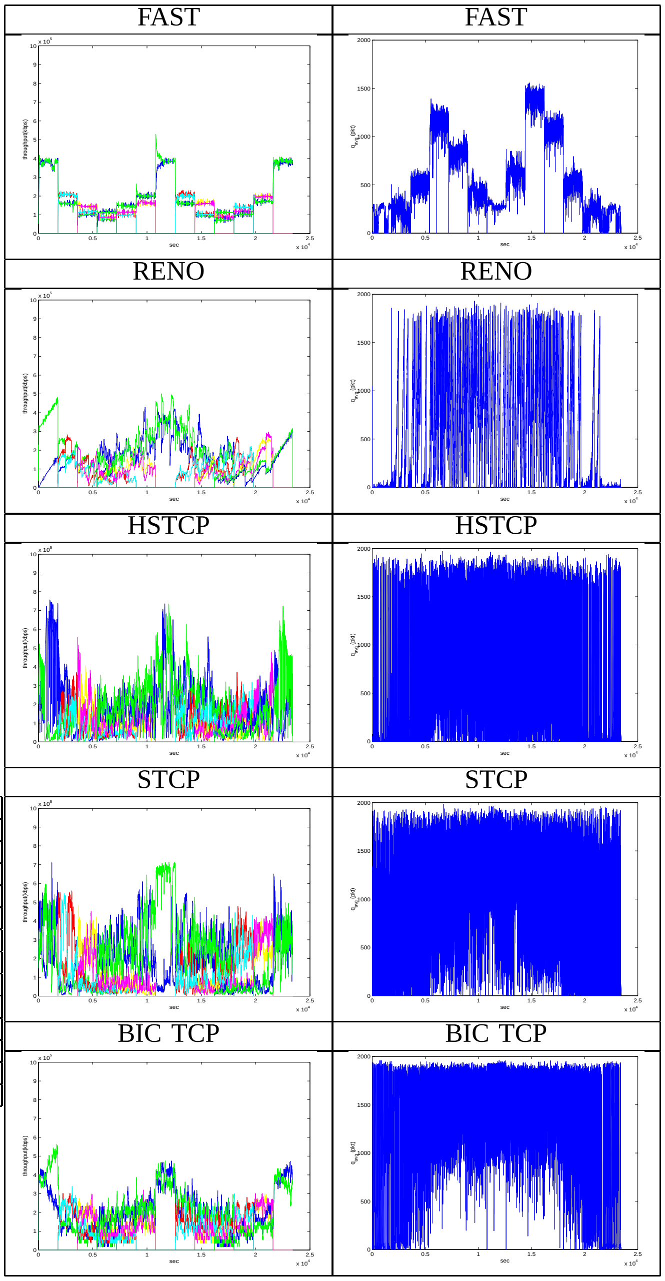

Fig. 5. Dynamic scenario II: Throughput trajectory (left column) and Dummynet queue trajectory (right column).

D. 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 here a summary of protocol performance in terms of some quantitative measures on throughput, fairness, stability, and responsiveness.

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 packet header overheads.

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

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

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 $E := \sum_{i=1}^{n} \overline{x}_i$.
2) **Intra-protocol fairness:** Jain's fairness index for the interval $[1, m]$ is defined as [22] $F := (\sum_{i=1}^{n} \overline{x}_i)^2 / (n \sum_{i=1}^{n} \overline{x}_i^2)$. $F \in (0, 1]$ 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:

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

The smaller the stability index, the less oscillation a source experiences. The stability index for interval $[1, m]$ is the average over the n active sources $S := \sum_{i=1}^{n} S_i/n$.

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 $\overline{x}_i(k) := \sum_{t=1}^{k} x_i(t)/k$ be the running average by period $k \leq m$. 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 $\overline{x}_i$:[5]

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

Responsiveness index R_2 measures how fast the aggregate throughput converges to $\sum_i \overline{x}_i$:

$$R_2 \quad := \quad \max\left\{k : \left|\frac{\sum_i\left(\overline{x}_i(k)-\overline{x}_i\right)}{\sum_i \overline{x}_i}\right| > 0.1\right\}$$

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 6 – 9.

From Figures 6–9, FAST has the best performance among all protocols for three evaluation criteria, fairness, stability and responsiveness index R_1. It has the second best overall throughput. 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 observed 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 both BIC-TCP and HSTCP achieved comparable intra-protocol fairness to Reno. HSTCP, BIC-TCP, and STCP showed increased oscillations compared with Reno (Figures 8, 3), and the oscillations became worse as the number of sources increased (Figure 5).

From Figure 9, 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. Indeed, from Figure 10, all protocols except TCP Reno perform well on the responsiveness index R_2 which is based on *aggregate* throughput. This apparent discrepancy reflects the fact that link utilization traces converge more quickly than individual throughput traces. It also serves as a justification for the link model (5): the *aggregate* input rate to a link converges more rapidly than individual rates, and hence the queue stabilizes quickly to its new level that tracks changes in windows.

[5]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.

E. NS-2 simulations

Our dummynet testbed is limited to experiments with single-bottleneck networks and identical protocol. We conducted NS-2 simulation to study the performance of FAST in more complex environments. The FAST implementation in NS-2 is from CUBIN Lab [78]. We set up the parameters so that only the original FAST algorithm as used in the dummynet experiments reported above was enabled. To eliminate possible simulation artifacts, such as phase effect, we introduced two-way noise traffic in the simulation, where a certain number of Pareto on-off flows with shape parameter 1.5 were introduced in *each* direction.[6] When a noise flow is "on", it transmits at a constant rate of 4Mbps. Each noise flow has an average burst time of 100ms and an average idle time of 100ms. Hence the average length of a flow is 50KB, similar to web traffic. We repeated each scenario 20 times and report both the average rate and the standard deviation (error bars in the figures).

Three sets of simulations were conducted: FAST with different noise levels, FAST with Reno traffic, and FAST on a multilink network. Due to space limitation, we only present a few examples from each set of simulations. See [79] for complete details.

1) FAST with noise traffic: This set of simulations repeated the scenario in Section IV-B, with different levels of noise traffic. The noise traffic was in the form of multiple Pareto on-off flows as described above. We varied the number of noise flows from 0 to 200, corresponding to an aggregate noise traffic of 0% to 50% of the bottleneck capacity. Figures 11 – 13 show the throughput trajectory of three cases: 0%, 10% (40 noise flows) and 30% (120 noise flows). Each point in the figures represents the average rate over a 60 second interval.

The NS-2 simulation with 0% Noise (Figure 11) should be compared with dummynet experiment in Section IV-B. Different from dummynet experiments, the NS-2 simulation was clean, and new flows mistook queueing delay due to existing flows as part of their propagation delays, leading to unfair throughputs. However, when the noise was 10% of the capacity, such unfairness was eliminated. The queue was frequently emptied and new flows observed the correct propagation delays and converged to the correct equilibrium rates, as shown in Figure 12. Figure 13 shows the throughput when the noise was 30% of the capacity. FAST throughputs oscillated, adapting to mice that joined and left frequently. In the period of 720 to 1080 second, the mice traffic generated so much packet loss that the three FAST flows could not keep α packets in the queue and they behaved like an AIMD algorithm. Such AIMD behavior led to discrimination against long RTT flows (flow 1 and flow 3).

[6]We also conducted simulations with exponential on-off traffic. The results are similar.

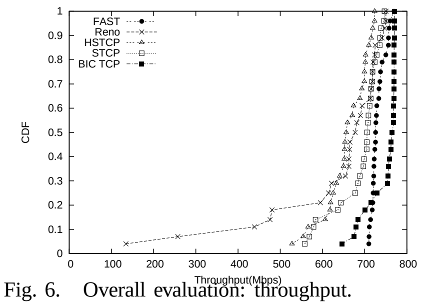

Fig. 6. Overall evaluation: throughput.

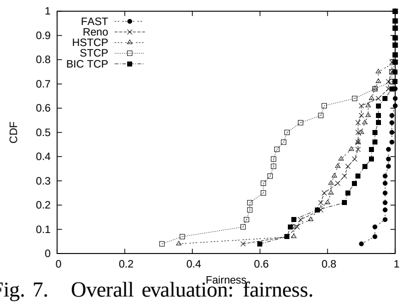

Fig. 7. Overall evaluation: fairness.

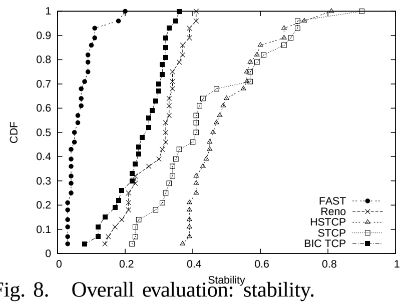

Fig. 8. Overall evaluation: stability.

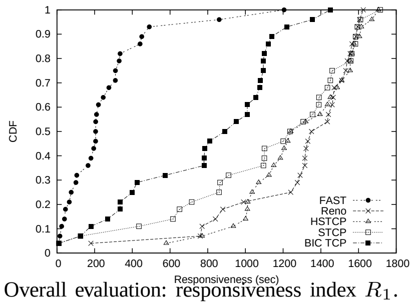

Fig. 9. Overall evaluation: responsiveness index R_1.

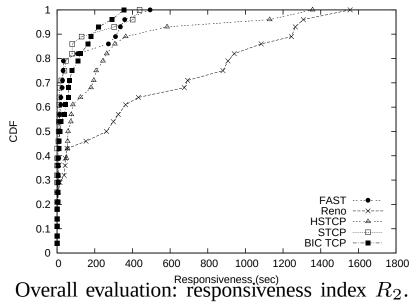

Fig. 10. Overall evaluation: responsiveness index R_2.

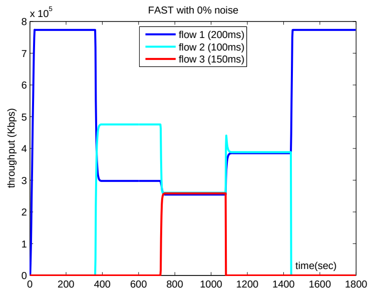

Fig. 11. FAST with 0% mice traffic.

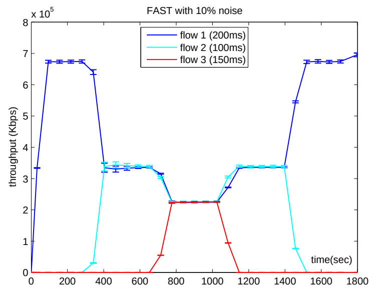

Fig. 12. FAST with 10% mice traffic.

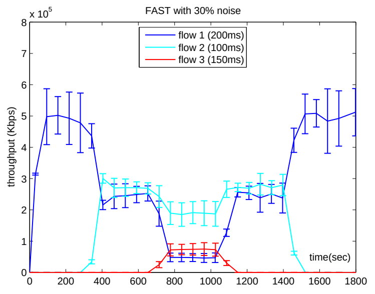

Fig. 13. FAST with 30% mice traffic.

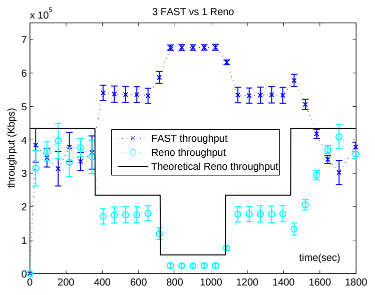

Fig. 14. 3 FAST flows vs 1 Reno flow.

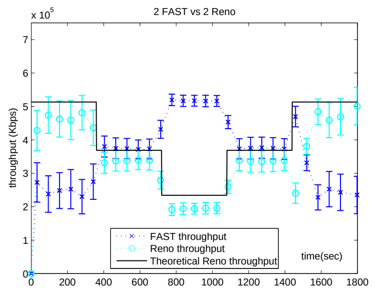

Fig. 15. 2 FAST flows vs 2 Reno flows.

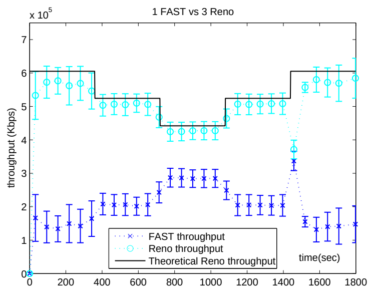

Fig. 16. 1 FAST flow vs 3 Reno flows.

2) *Interaction between FAST and Reno:* In this set of simulations, we used the same set of paths as in Section IV-B, but we reduced the delay on each path to one-fifth of the original value since Reno took a very long time to reach congestion avoidance with the delays in the original setup. On each path, we used 4 parallel flows instead of a single flow. We varied the number Reno flows on each path from zero to four (and the number of FAST flows was hence varied from four to zero) to examine FAST's interaction with Reno. The equilibrium rates of FAST and Reno sharing a single bottleneck link are predictable; see [79] for details. Figures 14 – 16 show the aggregate throughputs of FAST flows and Reno flows when the number of Reno flows on each path is 1, 2, and 3. We also present the theoretic predictions on Reno's throughputs on the same figures for comparison. The aggregate throughputs in simulations match the model predictions reasonably well. Reno's throughput is slightly lower than prediction since the model does not captures Reno's timeout behavior. The simulation results also show

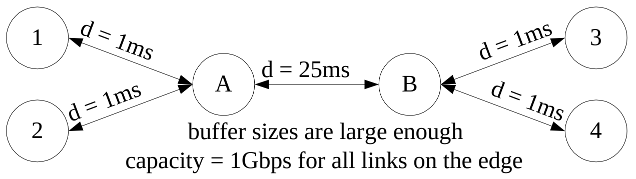
1
2
3
4
A
B
d = 1ms
d = 1ms
d = 25ms
d = 1ms
d = 1ms
buffer sizes are large enough
capacity = 1Gbps for all links on the edge

Fig. 17. Topology with multiple bottlenecks

that FAST may be more aggressive, or less aggressive than Reno, depending on the network setup.

Indeed, the behavior of a *general multilink* network share by heterogeneous protocols that use different congestion signals, such as FAST (which uses delay) and Reno (which uses loss), can be very intricate. See [61], [60] for details and more comments in Section V.

3) *FAST in a network with multiple bottlenecks:* We simulated a network with two LANs connected by a WAN. The topology is shown in Figure 17. Three pairs of flows ran

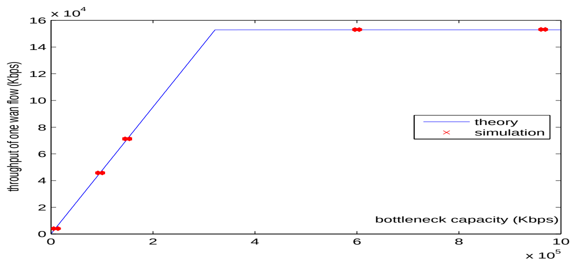

Fig. 18. FAST with multiple bottlenecks

simultaneously on three different paths. The first pair of flows (WAN flows) ran on the path $1 \rightarrow A \rightarrow B \rightarrow 4$. The second pair (LAN flows) ran on $1 \rightarrow A \rightarrow 2$. The third pair (LAN flows) ran on $3 \rightarrow B \rightarrow 4$. All links except $A \rightarrow B$ had a capacity of 1Gbps. The capacity of $A \rightarrow B$ was varied from 10Mbps to 1Gbps. The noise traffic introduced in each link has an average rate of 5% of the link capacity. Link $1 \rightarrow A$ and link $B \rightarrow 4$ were bottlenecks. Link $A \rightarrow B$ also became a bottleneck when its capacity was less than 333Mbps. In all scenarios, FAST converged stably to its equilibrium value, fully utilizing $1 \rightarrow A$ and $B \rightarrow 4$. Figure 18 shows the throughput of the WAN flow $1 \rightarrow A \rightarrow B \rightarrow 4$ with various capacities on $A \rightarrow B$, both measured from simulations and calculated by solving the utility maximization problem in (6). The measured throughputs match the theoretical predictions very well, validating the theoretical model and Theorem 2 on the equilibrium behavior of FAST.

V. Open issues and proposed solutions

FAST is a new delay-based congestion control algorithm. We motivate our design carefully and support it with tentative analysis and experiments. We now summarize some open problems and potential solutions. More practical experience is needed to assess conclusively the seriousness of these problems and the effectiveness of proposed solutions in real networks and applications.

A. *Propagation delay measurement*

Propagation delay (baseRTT) is used in the FAST window control algorithm (4). In a clean network, the queueing delay maintained by existing FAST flows may be mistaken as part of the propagation delay by new flows that join later, as shown in NS-2 simulations in [59]. The effect of this estimation error is equivalent to modifying the underlying utility functions to favor new flows over existing flows; see [44], [59]. Methods to eliminate this error are suggested in [49], [44], [59]. Our experience with high speed networks and dummynet, however, has been that this error is negligible in practice because noise or overshoot in the network is often sufficient to occasionally clear the queue, allowing new flows to observe the true propagation delay.

Propagation delay measurement can also be affected by route change from a shorter path to a longer path during the lifetime of a connection. Though route change at the timescale of TCP connections may be uncommon, mistaking an increase in propagation delay as congestion will reduce the connection's throughput. A solution is proposed in [49] where the propagation delay is estimated by the minimum RTT observed in a certain preceding period, not since the beginning of the connection, so that the estimate tracks route changes.

B. *Queueing delay measurement*

Queueing delay measurement may be affected by the burstiness of the FAST flows themselves, leading to slight unfairness among flows with different RTTs, as shown in IV-B. Such error can be greatly reduced by deploying a burstiness control algorithm in the sender, as shown in [71].

Like Vegas, FAST is affected by queueing delay in reverse path, as shown in [4]. There are a number of ways that have been proposed to eliminate the effect of reverse queueing delay for Vegas without the need for additional supports from receivers, that are applicable to FAST. The method in [36] utilizes the TCP timestamp option that is widely implemented in today's TCP stacks. If the sender and receiver have the same clock resolutions on their timestamps, the difference between the sender timestamp for a data packet, and the receiver timestamp for the corresponding ack is the one-way delay measurement in the forward direction. Thus, the difference between each one-way delay measurement and the minimum one-way delay measurement would be the forward queueing delay. Note that this calculation is correct even when the sender and receiver's clocks are not synchronized. As long as the clock drift is not significant in the life of a TCP flow, the clock offset will be eliminated through subtraction since both the one-way delay measurement and minimum one-way delay measurement has the same offset. If the clock drift is significant, [54], [62] provide techniques to accurately synchronize the clocks without GPS (global positioning system). If the sender and receiver have different clock resolutions, the sender can estimate the receiver clock period by observing the number of ticks of receiver's clock during a fixed time interval. A different method is proposed in [16] that does not directly measure the queueing delay. Instead, they measure that actual throughput in the forward direction, and use this measurement in place of $\frac{W}{d+q}$.

C. *α tuning*

The parameter α in window control equation (4) controls the number of packets that each flow maintains in the bottleneck links. If the number of "elephant" flows sharing a bottleneck link is large and the buffer capacity is small, these flows may not be able to reach their equilibria before they observe packet loss. In this case, the FAST window control algorithm oscillates like a loss-based AIMD algorithm does.[7]

D. *Heterogeneous protocols*

Congestion control algorithms that use the same congestion signal can be interpreted as distributed algorithms for network utility maximization; see e.g., [31], [42], [50], [76], [35], [41]. The underlying utility maximization problem implies e.g. that there exists a unique equilibrium (operating) point for general networks under mild conditions. It turns out that a network with heterogeneous protocols that react to different congestion signals can behave in a much more intricate way. In particular, we prove theoretically in [61] that there are networks that have multiple equilibrium points, and demonstrate experimentally in [60] this phenomenon using TCP Reno and Vegas/FAST. We also prove in [61] conditions

[7]Alternatively, a version of the FAST implementation deals with the problem of insufficient buffering by choosing α among a small set of pre-determined values based on achieved throughput. This can sometimes lead to unfair throughput allocation as reported in some of the literature. This version was used around early 2004, but discontinued since.

on network parameters that guarantee (global) uniqueness of equilibrium points for general networks.

We show in [60] that any desired inter-protocol fairness are *in principle* achievable by an appropriate choice of FAST parameter, and that intra-protocol fairness among flows within each protocol is unaffected by the presence of the other protocol except for a reduction in effective link capacities. How to design practical distributed algorithms that use only local information to achieve a desired inter-protocol fairness is however an open problem.

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. We thank J. Wang for pointing out several errors in an earlier version. 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), DARPA, and Cisco.

VI. Appendix: proofs

A. Proof of Lemma 1

Fix w.[8] Define $U_i(x_i) = w_i \log x_i - d_i x_i$ and consider the following optimization problem:

$$\max_{x \geq 0} \quad \sum_i U_i(x_i) \quad \text{subject to} \quad Rx \leq c \tag{9}$$

Since the objective function is strictly concave and the feasible set is compact, there exists a unique optimal solution x^*. Moreover, since R has full row rank, there exists a unique Lagrange multiplier p^* for the dual problem. See, e.g., [42] for details. We claim that p^* is the unique solution of (5) and, for all i,

$$x_i^* \quad = \quad w_i/(d_i + q_i^*) \tag{10}$$

Now, (10) can be rewritten as, for all i,

$$\sum_l R_{li} p_l^* \quad = \quad q_i^* \quad = \quad \frac{w_i}{x_i^*} - d_i \quad = \quad U_i'(x_i^*)$$

which is the Karush-Kuhn-Tucker condition for (9). Hence (10) holds. Then (5) becomes $\sum_i R_{li} x_i^* \leq c_l$, with equality if $p_l^* > 0$. But this is just the complementary slackness condition for (9). ■

B. Proof of Theorem 2

Clearly unique solution x^* for (6) and unique solution p^* for its dual exist, since the utility functions $\alpha_i \log x_i$ are strictly concave and R is full rank (see e.g. [42]). We need to show that the dual problem of (6) is indeed given by (7). Now the dual objective function is given by [2]

$$\begin{aligned} D(p) \quad := \quad & \sum_i \max_{x_i \geq 0} \left(\alpha_i \log x_i - x_i \sum_l R_{li} p_l \right) \; + \; \sum_l c_l p_l \\ = \quad & \sum_l c_l p_l \; - \; \sum_i \alpha_i \log \sum_i R_{li} p_l \\ & + \; \sum_i \alpha_i (\log \alpha_i - 1) \end{aligned}$$

[8]cf. the proof of a similar result in [50].

Since the last term is independent of p, minimizing $D(p)$ over $p \geq 0$ is the same as minimizing (7) over $p \geq 0$. Hence there exists a unique solution (x^*, p^*) for (6)–(7).

We now show that (x^*, p^*) is the equilibrium point of (4)–(5). In equilibrium, we have $w_i(t+1) = w_i(t) =: w_i$. From (4), the corresponding queueing delays p_l uniquely defined by (5) must be such that the end-to-end queueing delays are strictly positive, i.e., $q_i = \sum_l R_{li} p_l > 0$ for all i even though some p_l can be zero. Then $\alpha_i(w_i, q_i) = \alpha_i$ in equilibrium, and, from (4), we have $q_i = \sum_l R_{li} p_l = \alpha_i / x_i$, where $x_i := w_i/(d_i + q_i)$. But this is the Karush-Kuhn-Tucker condition for (6). Moreover, (5) is the complementary slackness condition. Hence the equilibrium of (4)–(5) coincides with the optimal solution of (6)–(7), i.e. $w = w^*$ and $p = p^*$. ■

C. Proof of Theorem 3

Let N be the number of sources. Let $q(t) = p(t)$ denote the queueing delay at the single link (omitting the subscripts). It is more convenient to work with normalized window

$$y_i(t) \quad := \quad \frac{w_i(t)}{d_i} \tag{11}$$

Let $Y(t) := \sum_i y_i(t)$ be the aggregate normalized window. Then $q(t) > 0$ if and only if $Y(t) > c$.

The window control algorithm (4) can be expressed in terms of updates on $y(t)$:

$$y_i(t+1) \quad = \quad \left(1 - \frac{\gamma q(t)}{d_i + q(t)} \right) y_i(t) + \gamma \hat{\alpha}_i \tag{12}$$

where $\hat{\alpha}_i := \alpha_i / d_i$. Let $\hat{\alpha} := \sum_i \hat{\alpha}_i$.

We first prove that the queue is lower bounded by a positive constant after a finite time.

Theorem 5: 1) For all $t > c/\gamma\hat{\alpha}$, we have $q(t) > 0$.

2) Moreover, given any $\epsilon > 0$ we have

$$\frac{\hat{\alpha}}{c} \cdot \min_i d_i - \epsilon \quad < \quad q(t) \quad < \quad \frac{\hat{\alpha}}{c} \cdot \max_i d_i + \epsilon$$

for all sufficiently large t.

Proof (Theorem 5). For the first claim, we will prove that the queue will be nonzero at some $t > c/\gamma\hat{\alpha}$, and that once it is nonzero, it stays nonzero.

Suppose $q(t) = 0$. Summing (12) over i, we have $Y(t+1) = Y(t) + \gamma\hat{\alpha}$, i.e., $Y(t)$ grows linearly in time by $\gamma\hat{\alpha}$ in each period. Since $Y(0) \geq 0$, $Y(t) > c$ after at most $c/\gamma\hat{\alpha}$ periods. Hence there is some $t > c/\gamma\hat{\alpha}$ such that $q(t) > 0$. We now show that $q(t) > 0$ implies $q(t+1) > 0$.

Since $\gamma < 1$, we have from (12)

$$\begin{aligned} y_i(t+1) \quad & \geq \quad \left(1 - \frac{q(t)}{d_i + q(t)} \right) y_i(t) + \gamma \hat{\alpha}_i \\ & = \quad \frac{d_i(t)}{d_i + q(t)} y_i(t) + \gamma \hat{\alpha}_i \end{aligned}$$

Summing over i gives

$$Y(t+1) \quad \geq \quad \sum_i \frac{d_i(t)}{d_i + q(t)} y_i(t) + \gamma \hat{\alpha}$$

But $q(t) > 0$ if and only if

$$\sum_i \frac{d_i(t)}{d_i + q(t)} y_i(t) \quad = \quad c \tag{13}$$

Hence

$$Y(t+1) \quad \geq \quad c + \gamma\hat{\alpha} \quad > \quad c$$

i.e., $q(t+1) > 0$. This proves the first claim.

For the second claim, we first prove that $Y(t)$ converges to its limit point $Y^* := c+\hat{\alpha}$ geometrically (and monotonically):

$$Y(t) \quad = \quad Y^* \ + \ (Y(0) - Y^*)(1-\gamma)^t \qquad (14)$$

To prove (14), rewrite (12) as

$$y_i(t+1) \quad = \quad (1-\gamma)y_i(t) \ + \ \gamma\left(\frac{d_i y_i(t)}{d_i + q(t)} + \hat{\alpha}_i\right)$$

Summing over i and using (13), we have

$$Y(t+1) \quad = \quad (1-\gamma)Y(t) + \gamma(c+\hat{\alpha})$$

from which (14) follows.

Noting that $d/(d+q)$ is a strictly increasing function of d, we have from (13)

$$\begin{aligned}\frac{\min_i d_i(t)}{\min_i d_i + q(t)} \cdot Y(t) \quad &\leq \quad \sum_i \frac{d_i(t)}{d_i + q(t)} y_i(t) \ = \ c \\ &\leq \quad \frac{\max_i d_i(t)}{\max_i d_i + q(t)} \cdot Y(t)\end{aligned}$$

Hence

$$1 + \frac{q(t)}{\min_i d_i} \quad \geq \quad \frac{Y(t)}{c} \quad \geq \quad 1 + \frac{q(t)}{\max_i d_i} \qquad (15)$$

From (14), we have

$$\frac{Y(t)}{c} \quad = \quad 1 + \frac{\hat{\alpha}}{c} \ + \ \left(\frac{Y(0) - Y^*}{c}\right)(1-\gamma)^t$$

Hence, (15) becomes:

$$\begin{aligned}\frac{q(t)}{\min_i d_i} \quad &\geq \quad \frac{\hat{\alpha}}{c} \ + \ \left[\left(\frac{Y(0)-Y^*}{c}\right)(1-\gamma)^t\right] \\ &\geq \quad \frac{q(t)}{\max_i d_i}\end{aligned}$$

Since $\gamma \in (0,1]$, the absolute value of the term in the square bracket can be made arbitrarily small by taking sufficiently large t. Hence, given any $\epsilon' > 0$,

$$\frac{q(t)}{\min_i d_i} \quad \geq \quad \frac{\hat{\alpha}}{c} - \epsilon'$$

and

$$\frac{q(t)}{\max_i d_i} \quad \leq \quad \frac{\hat{\alpha}}{c} + \epsilon'$$

for all sufficiently large t. This proves the second claim.[9] ■

Hence without loss of generality, we will assume

$$q(t) \quad > \quad \frac{\hat{\alpha}}{2c} \cdot \min_i d_i \qquad \text{for all } t \geq 0$$

This implies that, for all $t \geq 0$, equality holds in (5), or equivalently, (13) holds.

[9]When $\gamma = 1$, then the proof shows that we can set $\epsilon = 0$ in the statement of Theorem 5 after at most $c/\hat{\alpha}$ periods. Moreover, $Y(t) = c + \hat{\alpha}$ for all $t \geq 1$. It also implies that, if $d_i = d$ for all i, then $q(t) = \hat{\alpha}d/c$ for all $t \geq c/\hat{\alpha}$, i.e., the system converges in finite time.

More generally, for all $y \in \Re^N_+$ and $q \in \Re_+$, let

$$G(y,q) \quad := \quad \sum_i \frac{d_i y_i}{d_i + q} - c \quad = \quad 0 \qquad (16)$$

Lemma 1 guarantees that given any $y \in \Re^N_+$, there is a unique $q \in \Re_+$ that satisfies (16).

An important implication of Theorem 5(2) is that we can restrict our space of y to a subset of $\Re^N_+$:

$$\begin{aligned}\mathcal{Y} \quad := \quad \{ \ y \in \Re^N_+ \mid & \text{ the unique } q(y) \text{ defined implicitly} \\ & \text{by (16) is greater than } \hat{\alpha} \cdot \min_i d_i/(2c) \ \}\end{aligned} \qquad (17)$$

The key feature of $\mathcal{Y}$ we will need in Lemma 9 is that, for all $y \in \mathcal{Y}$, $q(y)$ is lower bounded *uniformly* in y. Define $F : \mathcal{Y} \longrightarrow \mathcal{Y}$ by

$$F_i(y) \quad := \quad \left(1 - \frac{\gamma q(y)}{d_i + q(y)}\right) y_i + \gamma\hat{\alpha}_i \qquad (18)$$

where $q(y)$ is implicitly defined by (16). Then the evolution (12) of the normalized window is $y(t+1) = F(y(t))$. Our main result is to show that the iteration F is locally asymptotically stable by proving that the spectral radius of $\partial F/\partial y$ is strictly less than 1 on $\mathcal{Y}$.

Theorem 6: Fix any $\gamma \in (0,1]$. For all $y \in \mathcal{Y}$, the spectral radius of $\partial F/\partial y$ is strictly less than 1.

Theorem 6 implies a neighborhood of the unique fixed point y^* defined by $y^* = F(y^*)$ such that given any initial normalized window $y(0)$ in this neighborhood, $y(t+1) = F(y(t))$ converges to y^*. This implies Theorem 3.

Sketch of proof (Theorem 6). We will show through Lemmas 7–9 that the spectral radius $\rho(\partial F/\partial y)$ is uniformly bounded away from 1, i.e., given $\gamma \in (0,1]$, there exists $\eta' > 0$ such that for all $y \in \mathcal{Y}$,

$$\rho\left(\frac{\partial F}{\partial y}\right) \quad < \quad \eta' \quad < \quad 1 \qquad (19)$$

Let $q(y)$ denote the unique solution of (16). Let

$$\beta_i \quad := \quad \frac{d_i y_i}{(d_i + q(y))^2}\left[\sum_j \frac{d_j y_j}{(d_j + q(y))^2}\right]^{-1} \qquad (20)$$

$$\mu_i \quad := \quad \frac{d_i}{d_i + q(y)} \qquad (21)$$

By Theorem 5(2), we have

$$0 \quad < \quad \beta_i, \mu_i \quad < \quad 1 \qquad \text{and} \qquad \sum_i \beta_i \quad = \quad 1$$

Let M :=diag(μ_i) be the diagonal matrix with μ_i as its nonzero entries. Let $\beta := (\beta_i$, for all $i)^T$ and $\mu := (\beta_i$, for all $i)^T$ be column vectors.

The proof of the following lemma is straightforward and can be found in [24].

Lemma 7: For $\gamma \in (0,1]$,

$$\frac{\partial F}{\partial y} \quad = \quad \gamma\left(M - \beta\mu^T\right) + (1-\gamma)I$$

where I is the $N \times N$ identity matrix.

Let the eigenvalues of $\partial F/\partial y$ be denoted by $\lambda_i(\gamma)$, $i = 1,\ldots,N$, as a function of $\gamma \in (0,1]$. We will show in the next two lemmas that when $\gamma = 1$,

$$0 \ \leq \ \lambda_i(1) \ < \ 1 \qquad \text{for all } i \tag{22}$$

Then Lemma 7 implies that, for all $\gamma \in (0,1]$,

$$0 \ \leq \ \lambda_i(\gamma) \ = \ \gamma\lambda_i(1) + (1-\gamma) \ < \ 1 \qquad \text{for all } i$$

and hence (19) holds for any given $\gamma \in (0,1]$.

The key observation to proving (22) is that we can explicitly characterize all the eigenvalues of $\partial F/\partial y$. These eigenvalues are functions of y even though this is not explicit from the notation. Fix $\gamma = 1$ and fix any y. Suppose the set $\{\mu_1,\ldots,\mu_N\}$ takes $k \leq N$ distinct values. Without loss of generality suppose $\mu_1,\ldots,\mu_{j_1}$ take the value $\tilde{\mu}_1$, $\mu_{j_1+1},\ldots,\mu_{j_2}$ take the value $\tilde{\mu}_2$, ..., $\mu_{j_{k-1}+1},\ldots,\mu_{j_k}$ take the value $\tilde{\mu}_k$, such that $\sum_{i=1}^k j_i = N$. The following lemma characterizes completely the eigenvalues and eigenvectors of the Jacobian, and is proved in [24].

Lemma 8: Suppose $\gamma = 1$ and fix any y. Then

1) $\lambda_1 = 0$ is an eigenvalue of $\partial F/\partial y$ with corresponding eigenvector $v_1 = M^{-1}\beta$.
2) For $i = 1,\ldots,k$, if $j_i > 1$ then $\tilde{\mu}_i$ is an eigenvalue with algebraic and geometric multiplicity $j_i - 1$. There are $N-k$ such distinct eigenvalues.
3) The remaining $k-1$ eigenvalues are the solutions of

$$\sum_{i=1}^{k} \frac{\tilde{\beta}_i}{\tilde{\mu}_i - \lambda} \ = \ 0 \tag{23}$$

counting (algebraic) multiplicity, where $\tilde{\beta}_i := \sum_{j=1}^{j_i} \beta_j$. The eigenvectors corresponding to these eigenvalues λ_i, $i = 2,\ldots,k$, are

$$v_i \ = \ (M - \lambda_i I)^{-1}\beta_i \tag{24}$$

The following lemma proves the assertion and is proved in [24].

Lemma 9: Suppose $\gamma = 1$. Then

$$\rho\left(\frac{\partial F}{\partial y}\right) \ \leq \ \max_i \mu_i \ < \ \max_i \frac{d_i}{d_i + \underline{q}} \ < \ 1$$

where $\underline{q} := \hat{\alpha}\, \min_i d_i/2c > 0$.

This completes the proof of Theorem 6, from which Theorem 3 follows.

D. Proof of Corollary 4

From (12) and (13), the equilibrium windows $w_i^* = y_i^* d_i$ and delay $q^* = p^*$ satisfy

$$\frac{p^* w_i^*}{d_i + p^*} \ = \ \alpha_i \qquad \text{for all } i \tag{25}$$

$$\sum_i \frac{w_i^*}{d_i + p^*} \ = \ c \tag{26}$$

Summing (25) over i and substituting in (26), we have $p^* = \alpha/c$, where $\alpha := \sum_i \alpha_i$. Substituting into (25), we have

$$w_i^* \ = \ \alpha_i\left(1 + \frac{d_i}{p^*}\right) \ = \ \alpha_i + \frac{\alpha_i c}{\alpha} d_i$$

Hence

$$x_i^* \ = \ \frac{w_i^*}{d_i + p^*} \ = \ \frac{\alpha_i}{\alpha}\, c$$

The second and third claims follow from Theorem 5 and footnote 10 at the end of the proof of Theorem 5. ■

References

[1] M. Allman, V. Paxson, and W. Stevens. TCP congestion control. RFC 2581, April 1999.

[2] D. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1995.

[3] 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*, 13(8):1465–80, October 1995. `http://cs.princeton.edu/nsg/papers/jsac-vegas.ps`.

[4] H. Bullot, R. Les Cottrell, and R. Hughes-Jones. Evaluation of advanced TCP stacks on fast long-distance production networks. *Journal of Grid Computing*, 1(4):345–359, August 2004.

[5] 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*, 8:467–479, 2002.

[6] C. S. Chang and Z. Liu. A bandwidth sharing theory for a large number of HTTP-like connections. *IEEE/ACM Trans. on Networking*, 12(5), October 2004.

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

[8] Hyojeong Choe and Steven H. Low. Stabilized Vegas. In Sophie Tarbouriech, Chaouki Abdallah, and John Chiasson, editors, *Advances in Communication Control Networks, Lecture Notes in Control and Information Sciences*. Springer Press, 2004.

[9] J-Y. Choi, K. Koo, J S. Lee and S.H. Low. Global Stability of FAST TCP in Single-Link Single-Source Network. In *Proc. of the IEEE Conference on Decision and Control*, December 2005.

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

[11] X. Fan, M. Arcak, and J.T. Wen. Robustness of network flow control against disturbances and time-delay. *System and Control Letters, to appear*, 2003.

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

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

[14] S. Floyd and T. Henderson. The NewReno modification to TCP's Fast Recovery algorithm. RFC 2582, April 1999.

[15] 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.

[16] C. P. Fu, Soung C. Liew. A Remedy for Performance Degradation of TCPVegas in Asymmetric Networks. *IEEE Communications Letter* Jan., 2003.

[17] S. Hegde, D. Lapsley, J. Lindheim, B. Wydrowski, D. Wei, C. Jin, and S. H. Low. FAST TCP in high-speed networks: An experimental study. In *Proc. of the First International Workshop on Networks for Grid Applications*, October 2004.

[18] 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`.

[19] 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*, 47(6):945–959, 2002.

[20] 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`.

[21] V. Jacobson, R. Braden, and D. Borman. TCP extensions for high performance. RFC 1323, May 1992.

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

[23] Raj Jain. A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks. *ACM Computer Communication Review*, 19(5):56–71, Oct. 1989.

[24] C. Jin, D. X. Wei, and S. H. Low. FAST TCP: motivation, architecture, algorithms, performance. Technical Report CaltechCSTR:2003.010, Caltech, Pasadena CA, December 2003. `http://netlab.caltech.edu/FAST`.

[25] C. Jin, D. X. Wei, and S. H. Low. TCP FAST: motivation, architecture, algorithms, performance. In *Proceedings of IEEE Infocom*, March 2004. `http://netlab.caltech.edu`.

[26] C. Jin, D. X. Wei, S. H. Low, G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, and S. Singh. FAST TCP: from theory to experiments. *IEEE Network*, 2005.

[27] Shudong Jin, Liang Guo, Ibrahim Matta, and Azer Bestavros. A spectrum of TCP-friendly window-based congestion control algorithms. *IEEE/ACM Transactions on Networking*, 11(3), June 2003.

[28] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high-bandwidth delay product networks. In *Proc. ACM Sigcomm*, August 2002. `http://www.ana.lcs.mit.edu/dina/XCP/`.

[29] Frank P. Kelly. Mathematical modelling of the Internet. In B. Engquist and W. Schmid, editors, *Mathematics Unlimited - 2001 and Beyond*, pages 685–702. Springer-Verlag, Berlin, 2001.

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

[31] Frank P. Kelly, Aman Maulloo, and David Tan. Rate control for communication networks: Shadow prices, proportional fairness and stability. *Journal of Operations Research Society*, 49(3):237–252, March 1998.

[32] Tom Kelly. Scalable TCP: Improving performance in highspeed wide area networks. *Computer Communication Review*, 32(2), April 2003. `http://www-lce.eng.cam.ac.uk/~ctk21/scalable/`.

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

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

[35] S. Kunniyur and R. Srikant. End-to-end congestion control: utility functions, random losses and ECN marks. *IEEE/ACM Transactions on Networking*, 11(5):689 – 702, October 2003.

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

[37] 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*, 5(3):336–350, June 1997. `http://www.ece.ucsb.edu/Faculty/Madhow/Publications/ton97.ps`.

[38] Y. Li. Implementing High-Speed TCP. `http://www.hep.ucl.ac.uk/~ytl/tcpip/hstcp/index.html`.

[39] S. H. Low, F. Paganini, J. Wang, and J. C. Doyle. Linear stability of TCP/RED and a scalable control. *Computer Networks Journal*, 43(5):633–647, 2003. `http://netlab.caltech.edu`.

[40] S. H. Low and R. Srikant. A mathematical framework for designing a low-loss, low-delay internet. *Networks and Spatial Economics, special issue on "Crossovers between transportation planning and telecommunications", E. Altman and L. Wynter*, 4:75–101, March 2004.

[41] Steven H. Low. A duality model of TCP and queue management algorithms. *IEEE/ACM Trans. on Networking*, 11(4):525–536, August 2003. `http://netlab.caltech.edu`.

[42] Steven H. Low and David E. Lapsley. Optimization flow control, I: basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, 7(6):861–874, December 1999. `http://netlab.caltech.edu`.

[43] Steven H. Low, Fernando Paganini, and John C. Doyle. Internet congestion control. *IEEE Control Systems Magazine*, 22(1):28–43, February 2002.

[44] Steven H. Low, Larry Peterson, and Limin Wang. Understanding Vegas: a duality model. *J. of ACM*, 49(2):207–235, March 2002. `http://netlab.caltech.edu`.

[45] Jim Martin , Arne Nilsson , Injong Rhee. Delay-based congestion avoidance for TCP. *IEEE/ACM Transactions on Networking*, v.11 n.3, p.356-369, June 2003.

[46] L. Massoulie and J. Roberts. Bandwidth sharing: objectives and algorithms. *IEEE/ACM Transactions on Networking*, 10(3):320–328, June 2002.

[47] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. RFC 2018, October 1996.

[48] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *ACM Computer Communication Review*, 27(3), July 1997. `http://www.psc.edu/networking/papers/model_ccr97.ps`.

[49] J. Mo, R. La, V. Anantharam, and J. Walrand. Analysis and comparison of TCP Reno and Vegas. In *Proceedings of IEEE Infocom*, March 1999.

[50] Jeonghoon Mo and Jean Walrand. Fair end-to-end window-based congestion control. *IEEE/ACM Transactions on Networking*, 8(5):556–567, October 2000.

[51] 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`.

[52] Fernando Paganini, Zhikui Wang, John C. Doyle, and Steven H. Low. Congestion control for high performance, stability and fairness in general networks. *IEEE/ACM Transactions on Networking* , 13(1):43-56, February 2005.

[53] 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`.

[54] Attila Pásztor and Darryl Veitch. Pc based precision timing without GPS. In *Proc. ACM Sigmetrics*, June 2002.

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

[56] 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`.

[57] R. Srikant. *The Mathematics of Internet Congestion Control*. Birkhauser, 2004.

[58] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery algorithms. RFC 2001, January 1997.

[59] L. Tan, C. Yuan and M. Zukerman. FAST TCP: Fairness and Queuing Issues. to appear in *IEEE Communications Letters*.

[60] Ao Tang, Jiantao Wang, Sanjay Hegde, and Steven H. Low. Equilibrium and fairness of networks shared by TCP Reno and FAST. *Telecommunications Systems*, special issue on High Speed Transport Protocols, 2005

[61] Ao Tang, Jiantao Wang, Steven H. Low, and Mung Chiang. Equilibrium of heterogeneous congestion control protocols. In *Proc. of IEEE Infocom*, March 2005.

[62] Darryl Veitch, Satish Babu, and Attila Pásztor. Robust remote synchronisation of a new clock for PCs. Preprint, January 2004.

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

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

[65] Jiantao Wang, Ao Tang, and Steven H. Low. Local stability of FAST TCP. In *Proc. of the IEEE Conference on Decision and Control*, December 2004.

[66] Jiantao Wang, David X. Wei, and Steven H. Low. Modeling and stability of FAST TCP. In *Proc. of IEEE Infocom*, March 2005.

[67] Jiantao Wang, David X. Wei, and Steven H. Low. Modeling and stability of FAST TCP. In P. Agrawal, M. Andrews, P. J. Fleming, G. Yin and L. Zhang, editors, *IMA Volumes in Mathematics and its Applications, Volume 143, Wireless Communications*. Springer Science, 2006.

[68] 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.

[69] 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.

[70] David X. Wei. Congestion Control Algorithms for High Speed Long Distance *TCP* Connections. Master Thesis, California Institute of Technology, Pasadena, Jun 2004. *URL:* http://netlab.caltech.edu/pub/projects/FAST/msthesis-dwei.

[71] David X. Wei, Steven H. Low, and Sanjay Hegde. A burstiness control for TCP. In *Proceedings of the 3rd International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet'2005)*, Feb 2005.

[72] 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.

[73] J.T. Wen and M. Arcak. A unifying passivity framework for network flow control. *IEEE Transaction on Automatic Control, to appear*, 2003.

[74] Bartek Wydrowski. High-resolution one-way delay measurement using RFC1323. Preprint, August 2004.

[75] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary increase congestion control for fast long distance networks. In *IEEE Proc. of INFOCOM*, March 2004.

[76] 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*, 8(5), October 2000.

[77] The Network Simulator - ns-2. *URL:* http://www.isi.edu/nsnam/ns/.

[78] CUBIN Lab. FAST TCP simulator module for NS-2. *URL:* http://www.cubinlab.ee.mu.oz.au/ns2fasttcp/.

[79] NetLab, Caltech NS-2 Simulation results of FAST. *URL:* http://netlab.caltech.edu/pub/projects/FAST/ns2-test.