

The Case for Delay-based Congestion Control

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

July 17, 2003

Abstract

We argue that, in the absence of explicit feedback, delay-based algorithms become the preferred approach for end-to-end congestion control as networks scale up in capacity. Their advantage is small at low speed but decisive at high speed. The distinction between packet-level and flow-level problems of the current TCP exposes the difficulty of loss-based algorithms at large congestion windows.

1 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 only 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 [10] and has gone through several enhancements since, e.g., [20, 11, 8]. 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* extremely small equilibrium loss probability.
3. At the packet level, oscillation is unavoidable because of the binary nature of the 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.

In this paper, we argue that the solution to these problems requires delay-based algorithms that are scalable to large capacity.

Delay-based congestion control has been proposed, e.g., in [12, 25, 2]. Its advantage over loss-based approach is small at low speed, but decisive at high speed, as we will see below. As pointed out in [19], 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 resolve problems at large windows. Instead, a new approach that fully exploits delay as a congestion measure, augmented with loss information, is needed [13].

2 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 designed 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 [7], STCP [16], and FAST TCP [13] are explicitly guided by flow level goals.

We elaborate in this section on the four difficulties of TCP Reno listed in Section 1. It is important to distinguish between packet-level and flow-level difficulties, because they must be addressed by different means.

2.1 Packet and flow level modeling

The congestion avoidance algorithm of TCP Reno and its variants have the form of AIMD [10]. The pseudo code for window adjustment is:

$$\begin{array}{l} \text{Ack:} \quad w \longleftarrow w + \frac{1}{w} \\ \text{Loss:} \quad w \longleftarrow w - \frac{1}{2}w \end{array}$$

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., [15, 9, 18]. In each RTT, the window $w_i(t)$ of source i increases by 1 packet,¹ and decreases by

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

where

$$x_i(t) := w_i(t)/T_i(t) \quad \text{pkts/sec} \quad (1)$$

$T_i(t)$ 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:

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

Setting $\dot{w}_i(t) = 0$ in (2) yields the well-known $1/\sqrt{p}$ formula for TCP Reno discovered in [21, 17], which relates loss probability to window size in equilibrium:

$$p_i^* = \frac{3}{2w_i^{*2}} \quad (3)$$

In summary, (2) and (3) describe the flow-level dynamics and the equilibrium, respectively, for TCP Reno.

Remarks

1. From (2),

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

i.e., on average, there are $3/2w_i^*$ packet losses per round trip time. In particular, this number decreases in inverse proportion of the equilibrium window size.

2. 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 $w_i = x_iT_i$, we can express (2) as:

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

where we have used the shorthand $\kappa_i(t) = \kappa_i(w_i(t), T_i(t))$ and $u_i(t) = u_i(w_i(t), T_i(t))$. It will turn out that different variants of TCP all have the same dynamic structure (4) at the flow level. They 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.

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

²This 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.

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

2.2 Equilibrium problem

The equilibrium problem at the flow level is expressed in (3): 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 .

To address the difficulties of TCP Reno at large window sizes, HSTCP and STCP increase more aggressively and decrease more gently, as discussed in Section 3 (see [3, 24, 14] for other approaches).

2.3 Dynamic problem

The cause of the oscillatory behavior in TCP Reno lies 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 (2) is unstable at large bandwidth-delay product [9, 18]. These must be addressed by different means, as we now elaborate.

Figure 2(a) illustrates the operating points chosen by various existing TCP congestion control algorithms, using the single-link single-flow case. 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 , and the amount of overshoot depends on the feedback delay. 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 operation, yielding full utilization.

Full utilization, even if achievable, comes at the cost of severe oscillations and potentially large queueing delay. The DUAL scheme in [25] proposes to oscillate around point D , the midpoint between C and R . 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 [12] proposes to oscillate around point C through AIMD with the same parameter $(1, 1/8)$

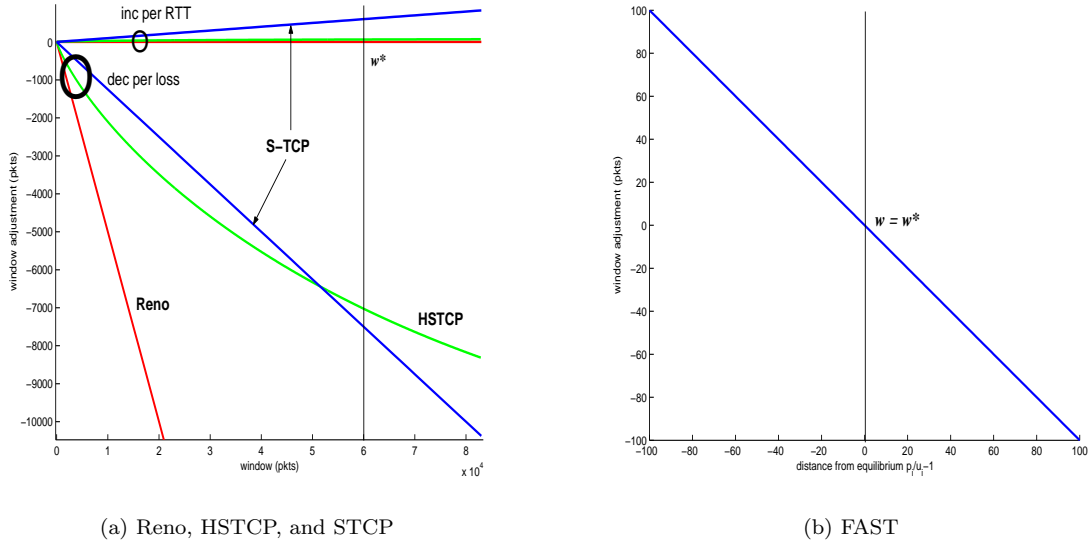


Figure 1: Packet-level implementation: (a) Window increment per RTT and decrement per loss, as functions of the current window size, for TCP Reno, HSTCP, and STCP. The increment functions for TCP Reno and HSTCP are barely distinguishable at this scale. (b) Window update as a function of *distance from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

as DUAL, based on the ratio of round-trip delay and delay gradient, to maximize power. In all these schemes, the congestion signal is binary, 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 [6], where congestion window is adjusted based on the estimated loss probability in an attempt to stabilize around a target value given by (3). 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 on 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 (2) leads to a feedback system that becomes unstable as feedback delay increases, and more strikingly, as network capacity increases [9, 18]. 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 3.3.

3 Loss-based approach

In this section, we first describe HSTCP [7] and STCP [16] at both the packet and flow levels, and then discuss how they address the problems discussed in Section 2.

3.1 HSTCP

The design of HSTCP proceeded almost in the opposite direction to that of TCP Reno [7]: the system equilibrium at the flow level is first designed, and then, the param-

eters of the packet level implementation are determined to implement the flow level equilibrium.

The first design choice decides the relation between window w_i^* and end-to-end loss probability p_i^* in equilibrium for each source i :

$$p_i^* = \frac{0.0789}{w_i^{*1.1976}} \quad (5)$$

The second design choice determines how to achieve the equilibrium defined by (5) through packet level implementation. The (congestion avoidance) algorithm is AIMD, as in TCP Reno, but with parameters $a(w_i)$ and $b(w_i)$ that vary with source i ’s current window w_i . The pseudocode for window adjustment is:

$$\begin{aligned} \text{Ack:} \quad w &\leftarrow w + \frac{a(w)}{w} \\ \text{Loss:} \quad w &\leftarrow w - b(w) \end{aligned}$$

The design of $a(w_i)$ and $b(w_i)$ functions is as follows. From an argument about the single-flow behavior, this algorithm yields an equilibrium where the following holds [5, 7]:

$$\begin{aligned} \frac{a(w_i^*)}{b(w_i^*)} \cdot \left(1 - \frac{b(w_i^*)}{2}\right) &= p_i^* w_i^{*2} \\ &= 0.0789 w_i^{*0.8024} \end{aligned} \quad (6)$$

where the last equality follows from (5). This motivates the design that, when loss probability p_i and the window w_i are *not* in equilibrium, one chooses $a(w_i)$ and $b(w_i)$ to force the relation (6) “instantaneously”:

$$\frac{a(w_i)}{b(w_i)} \cdot \left(1 - \frac{b(w_i)}{2}\right) = 0.0789 w_i^{0.8024} \quad (7)$$

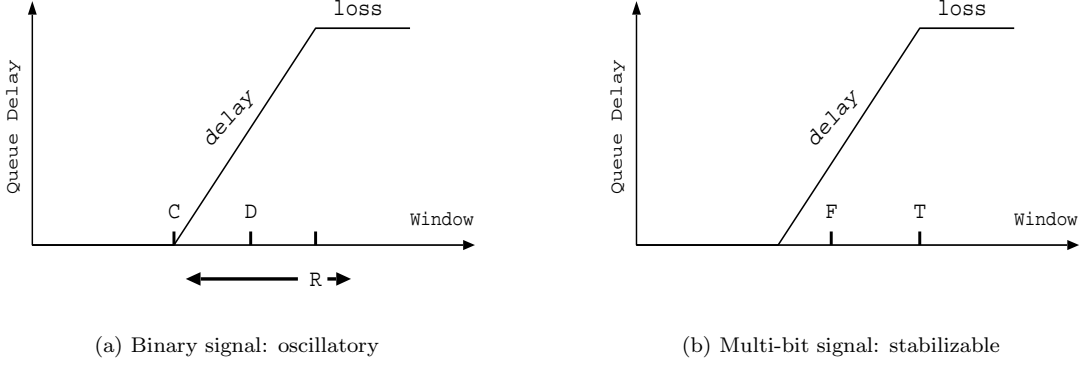


Figure 2: Operating points of TCP algorithms: R : Reno [10], HSTCP [7], STCP [16]; D : DUAL [25]; C : CARD [12]; T : TFRC [6]; F : Vegas [2], FAST [13]

The relation (7) defines a family of $a(w_i)$ and $b(w_i)$ functions. Picking $a(w_i)$ or $b(w_i)$ function uniquely determines the other function. The next design choice made in [7] is to pick a $b(w_i)$, hence also fixing $a(w_i)$.

The choice of $b(w_i)$ in [7] is, for w_i between 38 and 83,333 packets,

$$b(w_i) = -k_1 \log_e w_i + k_2 \quad (8)$$

where $k_1 = -0.0520$ and $k_2 = 0.6892$. This fixes $a(w_i)$ to be, from (7),

$$a(w_i) = 0.1578 w_i^{0.8024} \frac{b(w_i)}{2 - b(w_i)}$$

where $b(w_i)$ is given by (8). For $w_i \leq 38$ packets, $a(w_i) = 1$ and $b(w_i) = 0.5$ and HSTCP reduces to TCP Reno. For w_i (38 to 83,000 packets), $b(w_i)$ varies between $[0.1, 0.5]$.

The flow level model of HSTCP can be modeled using a similar argument to derive (2) for TCP Reno:

$$\begin{aligned} \dot{w}_i(t) &= \frac{a(w_i(t))}{T_i(t)} - \frac{2b(w_i(t))}{2 - b(w_i(t))} x_i(t) p_i(t) w_i(t) \\ &= \frac{2b(w_i(t))}{T_i(t)(2 - b(w_i(t)))} \\ &\quad \cdot \left(\frac{a(w_i(t))}{b(w_i(t))} \left(1 - \frac{b(w_i(t))}{2} \right) - p_i(t) w_i^2(t) \right) \end{aligned}$$

Using (7) to replace the first term in the parentheses, we have

$$\begin{aligned} \dot{w}_i(t) &= \frac{2b(w_i(t))}{T_i(t)(2 - b(w_i(t)))} \\ &\quad \cdot (0.0789 w_i^{0.8024}(t) - p_i(t) w_i^2(t)) \quad (9) \end{aligned}$$

In summary, the model of HSTCP is given by (5), (9) and (8).

Remarks

1. The equilibrium relation (5) implies that, on average, there are

$$p_i^* w_i^* = \frac{0.0789}{w_i^{*0.1976}}$$

many packet losses in a window; in particular, the average number of packet losses per round-trip time decreases with the equilibrium window, but more slowly than for TCP Reno.

2. Algorithm (9) can also be expressed in the form of (4) with the gain and marginal utility functions:

$$\begin{aligned} \kappa_i(w_i, T_i) &= \frac{0.1578 b(w_i) w_i^{0.8024}}{(2 - b(w_i)) T_i} \\ u_i(w_i, T_i) &= \frac{0.0789}{w_i^{1.1976}} \end{aligned}$$

3.2 Scalable TCP

The (congestion avoidance) algorithm of Scalable TCP is MIMD [16]:

$$\begin{aligned} \text{Ack:} & \quad w \longleftarrow w + a \\ \text{Loss:} & \quad w \longleftarrow w - bw \end{aligned}$$

for some constants $0 < a, b < 1$. Note that in each round-trip time without packet loss, the window increases by a multiplicative factor of a . The recommended values in [16] are $a = 0.01$ and $b = 0.125$.

As for HSTCP, the flow model of Scalable TCP is

$$\dot{w}_i = \frac{aw_i(t)}{T_i} - \frac{2b}{2 - b} x_i(t) p_i(t) w_i(t)$$

where $x_i(t) := w_i(t)/T_i$. In equilibrium, we have

$$p_i^* w_i^* = \frac{a}{b} \left(1 - \frac{b}{2} \right) =: \rho \quad (10)$$

This implies that, on average, there are ρ loss events per round-trip time, independent of the equilibrium window size.

We can rewrite (10) in the form of (4) with the gain and marginal utility functions:

$$\begin{aligned} \kappa_i(w_i, T_i) &= \frac{aw_i}{T_i} \\ u_i(w_i, T_i) &= \frac{\rho}{w_i} \end{aligned}$$

3.3 Disadvantages

The increment and decrement functions of HSTCP and STCP are plotted in Figure 1(a). Both upper bound those of Reno: they increment more aggressively and decrement less drastically. At the flow level, this means that, in equilibrium, both HSTCP and STCP can tolerate larger loss probabilities than TCP Reno (compare (5) and (10) with (3)). This alleviates the first two problems listed in Section 1. It does not, however, solve the dynamic problems at the packet and the flow levels.

At the packet level, as mentioned above, a natural way to avoid oscillation is to use multi-bit, as opposed of binary, congestion signal.³ Without explicit feedback, this means adjusting window based either on loss *probability*, as in [6], or on queueing delay, as in [13].⁴ 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, 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 in 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 speed grows [22, 4, 23].

4 Delay-based approach

As shown above, the congestion window in Reno, HSTCP and STCP all evolve according to:

$$\dot{w}_i(t) = \kappa_i(t) \cdot \left(1 - \frac{p_i(t)}{u_i(t)}\right) \quad (11)$$

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 [13] for details. 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 , as shown in the following table:

	$\kappa_i(w_i, T_i)$	$u_i(w_i, T_i)$	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

³Another option is to enlarge the equilibrium point to a set, as TCP Vegas does by using $\alpha < \beta$. This however makes fairness hard to control; see [1].

⁴TCP Vegas is also delay-based, but the *size* of its window adjustment does not depend on queueing delay. This is not important at low speed but critical at high speed.

This common model (11) 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 requires the *explicit* estimation of the end-to-end congestion measure $p_i(t)$. The design decision then reduces to the following two questions:

- What congestion measure $p_i(t)$ to use?
As argued in Section 3.3, in the absence of explicit feedback, queueing delay seems the only viable choice for congestion measure, as network capacity increases.
- How to choose $\kappa_i(w_i, T_i)$ and $u_i(w_i, T_i)$?
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 (11), together with link dynamics and their interconnection, has an equilibrium that is fair and efficient, and that the equilibrium is stable, in the presence of feedback delay.

An example of such a design is described in [13].

This approach, with proper choice of 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, the 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 [6], 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 queue 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 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, 4, 23].

References

- [1] C. Boutremans and J. Y. Le Boudec. A note on the fairness of TCP Vegas. In *Proceedings of International Zurich Seminar on Broadband Communications*, pages 163–170, February 2000.
- [2] 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>.
- [3] 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.
- [4] Hyojeong Choe and Steven H. Low. Stabilized Vegas. In *Proc. of IEEE Infocom*, April 2003. <http://netlab.caltech.edu>.
- [5] S. Floyd, M. Handley, J. Padhye, and J. Widmer. A comparison of equation-based and AIMD congestion control. Preprint, <http://www.aciri.org/tfrc/>, May 2000.
- [6] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proc. ACM SIGCOMM'00*, September 2000.
- [7] 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.
- [8] 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>.
- [9] 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.
- [10] 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>.
- [11] V. Jacobson, R. Braden, and D. Borman. TCP extensions for high performance. RFC 1323, <ftp://ftp.isi.edu/in-notes/rfc1323.txt>, May 1992.
- [12] Raj Jain. A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks. *ACM Computer Communication Review*, 19(5):56–71, Oct. 1989.
- [13] Cheng Jin, David X. Wei, and Steven H. Low. TCP FAST: motivation, architecture, algorithms, performance. submitted for publication, netlab.caltech.edu/, July 2003.
- [14] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high-bandwidth delay product networks. In *Proc. ACM Sigcomm*, August 2002.
- [15] 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.
- [16] 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.
- [17] 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>.
- [18] S. H. Low, F. Paganini, J. Wang, and J. C. Doyle. Linear stability of TCP/RED and a scalable control. *Computer Networks Journal*, to appear, 2003. <http://netlab.caltech.edu>.
- [19] Jim Martin, Arne Nilsson, and Injong Rhee. Delay-based congestion avoidance for TCP. *IEEE/ACM Trans. on Networking*, 11(3):356–369, June 2003.
- [20] 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.
- [21] 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.
- [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] 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>.
- [24] 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.
- [25] 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.