SLAC-PUB-10869

FAST TCP:
From Theory to Experiments *

C. Jin, D. 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, S. Singh †

http://netlab.caltech.edu/FAST/

November 1, 2004

Abstract

We describe a variant of TCP, called FAST, that can sustain high throughput and utilization at multi-Gbps over large distance. We present the motivation, review the background theory, summarize key features of FAST TCP, and report our first experimental results.

Keywords: FAST TCP, large bandwidth-delay product

1 Introduction

The congestion control algorithm in the current TCP 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 in the last fifteen years. It is also well-known, however, that as bandwidth-delay product continues to grow, the current TCP implementation will eventually become a performance bottleneck.

In this paper we describe a different congestion control algorithm for TCP, called FAST [1]. FAST TCP has three key differences. First, it is an equation-based algorithm and hence eliminates packet-level oscillations. Second, it uses queueing delay as the primary measure of congestion, which can be more reliably measured by end hosts than loss probability in fast long-distance networks. Third, it has a stable flow dynamics and achieves weighted proportional fairness in equilibrium that does not penalize long flows, as the current congestion control algorithm does. Alternative approaches are described in [2, 3, 4, 5, 6]. The details of the architecture, algorithms, extensive experimental evaluations of FAST TCP, and comparison with other TCP variants can be found in [1, 7].

In this paper, we will highlight the motivation, background theory, implementation and our first major experimental results. The scientific community is singular in its urgent need for efficient high speed data transfer. We explain in Section 2 why this community has been driving the development and deployment of ultrascale networking. The design of FAST TCP builds on an emerging theory that allows us to understand the equilibrium and stability properties of large networks under end-to-end control. It provides a framework to understand issues, clarify ideas and suggest directions, leading to a more robust and better performing design. We summarize this theory in Section 3 and explain FAST

*IEEE Network, to appear.

†G. Buhrmaster and L. Cottrell are with SLAC (Stanford Linear Accelerator Center), Stanford, CA. W. Feng is with LANL (Los Alamos National Lab). O. Martin is with CERN (European Organization for Nuclear Research), Geneva. F. Paganini is with EE Department, UCLA. All other authors are with Caltech, Pasadena, CA.

Work supported in part by the Department of Energy Contract DE-AC02-76SF00515

Invited talk presented at DPF 2004:
Annual Meeting of the Division of Particles and Fields (DPF) of the American Physical Society (APS),
8/26/2004 - 8/31/2004, Riverside, CA

TCP in Section 4. We report the results of our first global experiment in Section 5 and conclude in Section 6.

2 Motivation

One of the key drivers of ultrascale networking is the High Energy and Nuclear Physics (HENP) community, whose explorations at the high energy frontier are breaking new ground in our understanding of the fundamental interactions, structures and symmetries that govern the nature of matter and spacetime in our universe. The largest HENP projects each encompasses 2,000 physicists from 150 universities and laboratories in more than 30 countries. Collaborations on this global scale would not have been attempted if the physicists could not count on excellent network performance. Rapid and reliable data transport, at speeds of 1 to 10 Gbps and 100 Gbps in the future, is a key enabler of the global collaborations in physics and other fields. The ability to analyze and share many terabyte-scale data collections, accessed and transported in minutes, on the fly, rather than over hours or days as is the current practice, is at the heart of the process of search and discovery for new scientific knowledge.

For instance, the CMS (Compact Muon Solenoid) Collaboration, now building next-generation experiments scheduled to begin operation at CERN's (European Organization for Nuclear Research) Large Hadron Collider (LHC) in 2007, along with the other LHC Collaborations, is facing unprecedented challenges in managing, processing and analyzing massive data volumes, rising from the petabyte (10^{15} bytes) to the exabyte (10^{18} bytes) scale over the coming decade. The current generation of experiments now in operation and taking data at SLAC (Stanford Linear Accelerator Center) and Fermilab face similar challenges. SLAC's experiment has already accumulated more than a petabyte of stored data. Effective data sharing will require 10 Gbps of sustained throughput on the major HENP network links within the next 2 to 3 years, rising to terabit/sec within the coming decade.

Continued advances in computing, communication, and storage technologies, combined with the development of national and global Grid systems, hold the promise of providing the required capacities and an effective environment for computing and science. The key challenge we face, and intend to overcome with FAST TCP, is that the current congestion control algorithm of TCP does not scale to this regime.

The currently deployed TCP implementation is an enhanced version of Reno. It is a loss-based approach. It uses AIMD (additive increase multiplicative decrease) where the source transmission rate is increased by one unit (packet per round-trip time) in each round-trip time, and halved on each loss event. While it works well at low speed, AI is too slow and MD too drastic, leading to low utilization, as network scales up in capacity. Moreover, it perpetually pushes the queue to overflow. It also discriminates against flows with large round-trip times. To address these problems, TCP Vegas adopts a delay-based approach where a source implicitly estimates its end-to-end queueing delay. Instead of oscillating and pushing the queue to overflow, TCP Vegas stabilizes its rate at a value that buffers a target number of its own packets in its path in order to keep the path fully utilized. It adjusts its rate by one unit per round-trip time up or down depending on whether the number of its own packets buffered in the path falls short or exceeds the target. FAST TCP is a high-speed version of TCP Vegas, with a fairness property independent of the delay of the flows. We will explain their relationship in more detail in Section 4 below after explaining the background theory.

3 Background theory

There is now a preliminary theory to understand large-scale networks, such as the Internet, under end-to-end control. The theory clarifies how control algorithms and network parameters determine the equilibrium and stability properties of the network, and how these properties affect its performance,

fairness and responsiveness. It is useful both in understanding problems of the current congestion control algorithm as networks scale up in capacity and in designing better algorithms to solve these problems.

Congestion control consists of two components, a source algorithm, implemented in TCP, that adapts sending rate (or window) to congestion information in the source's path, and a link algorithm, implemented in routers, that updates and feeds back local congestion information to sources that traverse the link. Typically, the link algorithm is implicit and the measure of congestion is either loss probability or queueing delay. For example, the current protocol TCP Reno and its variants use loss probability as a congestion measure, and TCP Vegas primarily uses queueing delay as a congestion measure. Both are *implicitly* updated by the queueing process and *implicitly* fed back to sources via end-to-end loss and delay, respectively.

The source-link algorithm pair, referred to here as TCP/AQM (active queue management) algorithms, forms a distributed feedback system, the largest man-made feedback system in deployment. In this system, hundreds of millions of TCP sources and hundreds of thousands of network devices interact with each other, each executing a simple local algorithm, implicitly or explicitly, based on local information. Their interactions result in a collective behavior, whose equilibrium and stability properties we now discuss.

3.1 Equilibrium and performance

We can interpret TCP/AQM as a distributed algorithm over the Internet to solve a global optimization problem [8]; see also [9, 10] for recent surveys. The solution of the optimization problem and that of an associated problem (to be discussed below) determine the equilibrium and performance of the network. Different TCP and AQM algorithms all solve the same prototypical problem. They differ in the objective function of the underlying optimization problem and the iterative procedure to solve it.

Even though historically TCP and AQM algorithms have not been designed as an optimization procedure, this interpretation is valid under fairly general conditions, and useful in understanding network performance, such as throughput, utilization, delay, loss, and fairness. Moreover, the underlying optimization problem has a simple structure, that allows us to efficiently compute these equilibrium properties numerically, even for a large network that is hard to simulate.

Specifically, we can regard each source as having a utility function that measures its "happiness" as a function of its data rate. Consider the problem of maximizing the sum of all source utility functions over their rates, subject to link capacity constraints. This is a standard constrained optimization problem for which many iterative solutions exist. The challenge in our context is to solve for the optimal source rates in a distributed manner using only local information. A key feature we exploit is the duality theory. It says that associated with our (primal) utility maximization problem is a dual minimization problem. Whereas the primal variables over which utility is to be maximized are source rates, the dual variables for the dual problem are congestion measures at the links. Moreover, solving the dual problem is equivalent to solving the primal problem. There is a class of optimization algorithms that iteratively solve for both the primal and the dual problems at once.

TCP/AQM can be interpreted as such a primal-dual algorithm, that is distributed and decentralized, and that solves both the primal and dual problems. TCP iterates on the source rates (a source increases or decreases its window in response to congestion in its path), and AQM iterates on the congestion measures (e.g., loss probability at a link increases or decreases as sources traversing that link increase or decrease their rates). They cooperate to determine iteratively the network operating point that maximizes aggregate utility. When this iterative process converges, the equilibrium source rates are optimal solutions of the primal problem and the equilibrium congestion measures are optimal solutions of the dual problem. The throughput and fairness of the network are thus determined by the TCP algorithm and the associated utility function, whereas utilization, loss and delay are determined by the AQM algorithm.

3.2 Stability

If we think of an equilibrium state as the desired operating point that produces good network performance, then we want to make sure the equilibrium points are stable. This means that when the equilibrium point shifts because of changes in network topology or flow pattern, the network will converge to the new equilibrium point. It seems undesirable to operate a large network in an unstable regime, and unnecessary if we know how to operate it in a stable regime without sacrificing performance.

It has been shown that TCP Reno algorithm can become unstable as delay increases, or more strikingly, as network capacity increases! Moreover the high control gain introduced by TCP is mainly responsible for the instability. The high control gain is a consequence of halving the window size on each loss event. The gain increases rapidly with delay or capacity, making it very difficult for any AQM algorithm to stabilize the current TCP. This underlies the difficulty of tuning parameters in the RED (Random Early Detection) AQM scheme: they can be tuned to improve stability, but only at the cost of a large queue. Most recommendations in the literature aim to avoid a large queue, often leading to violent oscillations and reduced utilization.

Two types of TCP/AQM algorithms are proposed in [11] and [12] that can be proved to maintain stability around the equilibrium point at high capacity and large delay in general networks. While both of these algorithms are decentralized, they are complementary in many ways. The algorithms in [11], called primal algorithms, allow general utility functions, and hence arbitrary fairness in rate allocation, but give up tight control on utilization. The algorithms in [12], called dual algorithms, on the other hand, can achieve very high utilization, but are restricted to a specific class of utility functions, and hence fairness in rate allocation. The main insight from this series of work is that, to maintain stability, sources should scale down their responses by their individual round trip delays (i.e., adjust their rates up or down less aggressively if their round trip delays are large, and vice versa) and links should scale down their responses by their individual capacities (i.e., update their congestion measures less aggressively if their capacities are large, and vice versa).

In the original primal algorithms, only the source adaptation is dynamic and the link adaptation has no dynamics, while in the original dual algorithms, the reverse is true. By adding slow timescale dynamics to the link algorithm, the primal algorithms can be made to achieve both arbitrary fairness and high utilization [13]. The primal approach motivates a TCP implementation tailored for high bandwidth-delay product regime [4].

By adding slow timescale dynamics to the source algorithm, the dual algorithms can also be made to achieve both arbitrary fairness and high utilization [12]. The original link algorithm in [12] assumes explicit feedback so that network queues can be emptied. Its implementation would require modifying routers in the current Internet. However, this link algorithm has the same dynamics mathematically as the dynamics of queueing delay. This observation leads to an algorithm that can maintain linear stability without having to change the current routers [14]. These theoretical results suggest that it is possible to stabilize the Internet, as it continues to scale up in capacity and size, with the current FIFO (first-in-first-out) routers by modifying just the TCP kernel at the *sending hosts*.

4 FAST TCP

The congestion control mechanism of FAST TCP has four components. They are functionally independent so that they can be designed separately and upgraded asynchronously. The *data control* component determines *which* packets to transmit, *window control* determines *how many* packets to transmit, and *burstiness control* determines *when* to transmit these packets. These decisions are made based on information provided by the *estimation* component. Window control regulates packet transmission at the round trip timescale, while burstiness control works at a smaller timescale. In the following, we provide an overview of these components.

4.1 Estimation

This component computes two pieces of feedback information for each data packet sent – a multi-bit queueing delay and an one-bit loss-or-no-loss indication – which are used by the other three components. When a positive acknowledgment is received, FAST calculates the RTT for the corresponding data packet and then uses it to compute the minimum RTT and an exponentially smoothed average RTT. The average RTT and the minimum RTT are used in the window control component. When a negative acknowledgment (signaled by three duplicate acknowledgments or timeout) is received, it generates a loss indication for this data packet to the data control component.

4.2 Window control

Like TCP Vegas, FAST TCP uses queueing delay as its main measure of congestion in its window adjustment algorithm. In a loss-based approach, sources must periodically push buffers to overflow, in the absence of AQM, in order to generate the target loss probability, thus inducing a jittery behavior. Delay information allows the sources to settle into a steady state when the network is static. Queueing delay also has two advantages as a congestion measure. It provides a finer-grained measure of congestion: each measurement of packet loss (whether a packet is lost or not) provides one bit of congestion information, whereas each measurement of queueing delay provides multi-bit information, limited by clock accuracy and measurement noise. Moreover, the dynamics of delay has the right scaling with respect to link capacity that helps maintain stability as networks scale up in capacity [12, 14].

Under normal network conditions, FAST periodically updates the congestion window `w` based on the average RTT (round-trip time) according to:

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

where γ is a constant between 0 and 1, `RTT` is the current average round-trip time, `baseRTT` is the minimum RTT observed so far, and α is a protocol parameter that controls fairness and the number of packets each flow buffered in the network (see below).

Even though windows are adjusted in different ways at the packet level in TCP Vegas and FAST, their flow dynamics are similar mathematically. Indeed, the mathematical model of (1) is (ignoring the `2w` term)

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

where $w_i(t)$ is the window size of flow i in the current update period t, $x_i(t)$ is its current throughput, and $q_i(t)$ is its current queueing delay. TCP Vegas has a flow dynamic

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

where $T_i(t)$ is the current RTT of flow i, and $\mathrm{sign}(z) = -1$ if $z < 0$, 0 if $z = 0$, and 1 if $z > 0$. Hence while TCP Vegas adjusts its window up or down by one packet per RTT depending on whether the number $x_i(t)q_i(t)$ of buffered packets is smaller or greater than its target α_i, the size of window adjustment in FAST TCP depends on the magnitude, as well as the sign, of $\alpha_i - x_i(t)q_i(t)$. In other words, FAST adjusts its window by a large amount, up or down, when the number of buffered packets is far away from its target, and a small amount when it is close. In this sense, FAST is a high-speed version of Vegas.

The equilibrium and fairness properties of FAST TCP in general networks with multiple links and heterogeneous flows are simple to understand. Indeed, the equilibrium throughputs of FAST flows are the unique optimal vector x^* that maximizes $\sum_i \alpha_i \log x_i$ subject to the link constraint that the aggregate flow rate at any link does not exceed the link capacity. Here the sum is taken over all flows i.

Hence FAST maximizes log utility function. This implies in particular that FAST achieves *proportional fairness* which is milder than maxmin fairness in that it does not give absolute priority to small flows.

In addition to determining the fairness properties, the parameter α_i is also equal to the number of flow i's packets that are buffered in the routers in its path in steady state. If there are N flows, the total number of packets buffered in the routers in steady state is $\sum_{i=1}^{N} \alpha_i$. The distribution of these packets in the network, assuming all have enough buffering capacity, is completely determined by the utility maximization problem: while the source rates solve the primal problem, the vector of queueing delays at each link due to these packets is the optimal solution of the associated dual problem (Lagrange multipliers). Hence, it is easy to calculate all the equilibrium flow throughputs and link delays for a general network of FAST flows, if the network is static.

In reality networks are never static. Flows join and depart asynchronously. The solution of the utility maximization problem describes the behavior to which the network as a whole converges when flow pattern or topology shifts the equilibrium to a new point, provided the new equilibrium point is stable. Global stability of FAST TCP in the presence of feedback delay is still an open problem, but several partial results have been proved in [1, 15, 16]. First, FAST TCP is proved to be always locally stable in general networks in the absence of feedback delay [1, 15]. When feedback delay is present, it is locally stable if the *heterogeneity* of flow delays is small [16]. Second, FAST TCP is proved to be globally stable at a single link in the absence of delay [16]. Moreover, it converges exponentially fast to the equilibrium point.

4.3 Data control

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. When there is no loss, new packets are sent in sequence as old packets are acknowledged. This is referred to as *self-clocking* or *ack-clocking*. During loss recovery, a decision must be made on whether to retransmit lost packets, to keep transmitting new packets, or to retransmit older packets that are neither acknowledged nor marked as lost. The data control component makes the decision on how to mix packets from the three candidate pools.

This decision becomes important especially when bandwidth-delay product is large. For example, at a window size of 15,000 packets, a single loss event can lose 7,000 packets or more, e.g, in slow start. They must be retransmitted rapidly, yet in a way that does not exacerbate congestion and lead to more losses or even timeouts. Moreover, packets that are lost may not be detected all at once, which further complicates the decision of what to transmit.

4.4 Burstiness control

The burstiness control component smooths out transmission of packets in a fluid-like manner to track the available bandwidth. It is particularly important in networks with large bandwidth-delay products, where traffic can be extremely bursty due to events both in the network and at the end hosts. For instance, a single acknowledgment can acknowledge several thousand packets, opening up the window in a large burst. Sometimes the sender CPU is occupied for a long period to serve interrupts of incoming packets, allowing outgoing packets to accumulate at device output queue, to be transmitted in a large burst when the CPU becomes available. Extreme burstiness creates long queues and increases the likelihood of massive losses.

Pacing is a common way to solve the burstiness problem at sender. A straightforward implementation of pacing would have the TCP sender schedule successive packet transmissions at a constant time interval, obtained by dividing the congestion window by the current RTT. In practice, this would require a timer with a very high resolution. For example, a host with a 1 Gbps throughput and 1500-byte MTU (Maximum Transmission Unit) sends 83,333 packets per second and requires a scheduling

interval of 12 μs. Considering that the typical kernel task scheduler runs every 10 ms, the overhead of scheduling packet transmissions at 12 μs apart will significantly degrade overall OS performance. We can reduce the overhead by scheduling small bursts of packets instead of individual packets. However, at large congestion window, pacing alone cannot solve the burstiness problem.

We employ two burstiness control mechanisms, one to supplement self-clocking in streaming out individual packets and the other to increase window size smoothly in smaller bursts. *Burstiness reduction* decides how many packets to send, when an ack advances congestion window by a large amount, and attempts to limit the burst size on a smaller timescale than one RTT. *Window pacing* increases 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.

5 Experimental results

We have tested FAST TCP over continental, trans-Atlantic, and trans-Pacific distances of more than 10,000km, employing a variety of commercial products. Its first public demonstration was a series of experiments conducted during the SuperComputing Conference (SC2002) in Baltimore, MD, in November 16–22 2002 by a Caltech-SLAC research team working in partnership with the CERN, DataTAG, StarLight, TeraGrid, Cisco, and Level(3). In this section, we present some of our experiments during and after SC2002.

5.1 Infrastructure

The demonstrations used an OC192 (10 Gbps) link between Starlight (Chicago) and Sunnyvale, the DataTAG 2.5 Gbps link between Starlight and CERN (Geneva), an OC192 link connecting the SC2002 showfloor in Baltimore and the TeraGrid router in StarLight Chicago, and the Abilene backbone of Internet2. The network routers and switches at Starlight and CERN were used together with a Cisco GSR 12406 router at Sunnyvale, and sets of dual Pentium 4 servers each with dual gigabit Ethernet connections at Starlight, Sunnyvale, CERN and the SC2002 show floor provided by Caltech, SLAC and CERN. The network setup is shown in Figure 1.

We have conducted a number of experiments, all using the standard MTU 1500 bytes including TCP and IP headers. In all the experiments reported below, the bottleneck was either the gigabit Ethernet card or the transatlantic OC48 link.

5.2 Throughput and utilization

In this subsection, we report our SC2002 experiments on throughput and utilization. To put these results in perspective, we first present a set of calibration experiments conducted on January 27-28, 2003 after the SC2002 conference using the same testbed shown in Figure 1.

Using default device queue size (`txqueuelen` = 100 packets) at the network interface card, the default Linux TCP (version v2.4.18), without any tuning on the AIMD parameters, routinely achieves an average throughput of 185Mbps, averaged over an hour, with a single TCP flow between Sunnyvale in California and CERN in Geneva, via StarLight in Chicago, a distance of 10,037km with a minimum delay of 180ms round trip. This is out of a possible maximum of 973Mbps to the application, excluding TCP/IP overhead, limited by the gigabit Ethernet card, and represents a utilization of just 19%. If the device queue size is increased 100 times (`txqueuelen` = 10,000 packets), the average throughput increases to 266Mbps and utilization increases to 27%. With two TCP flows sharing the path, one flow between each pair of servers, the aggregate throughputs are 317Mbps with `txqueuelen` = 100 packets and 931Mbps with `txqueuelen` = 10,000 packets, out of a possible maximum of 1,947Mbps.

Under the same experimental conditions, using the default device queue size (`txqueuelen` = 100 packets), FAST TCP achieved an average throughput of 925Mbps and utilization of 95% during SC2002,

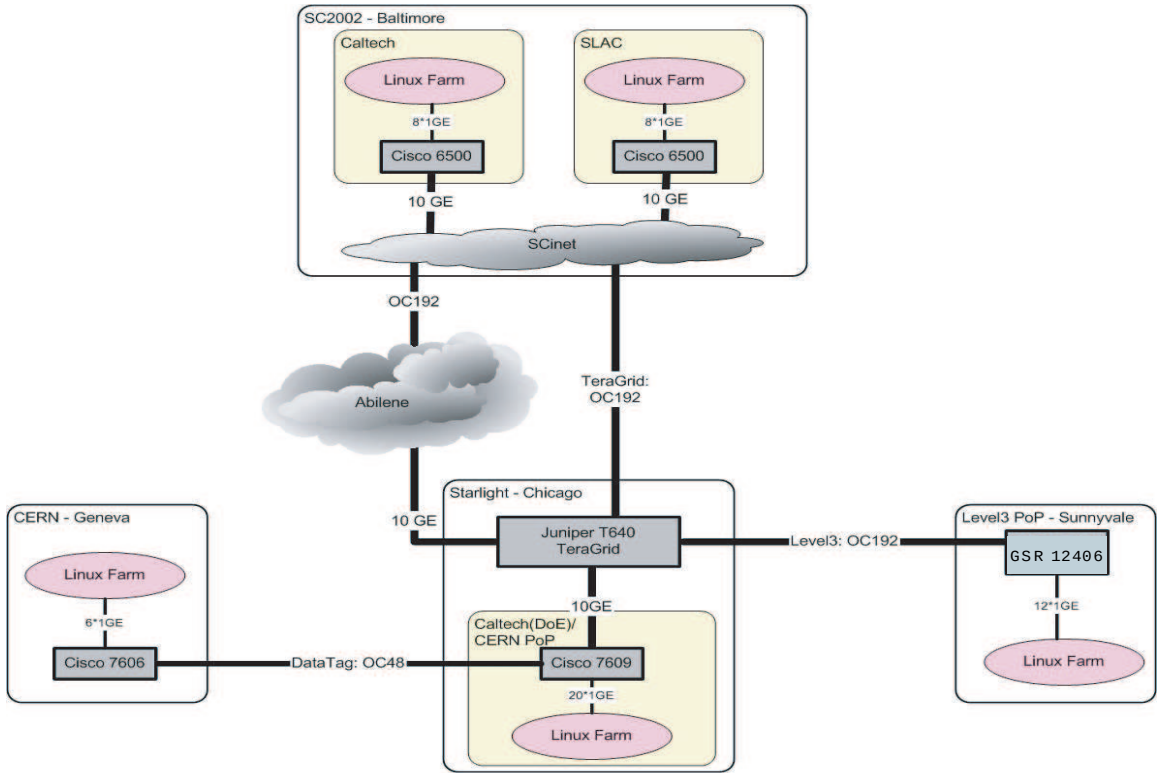

Figure 1: Network setup in SC2002, Baltimore, MD, November 16-22, 2002. Distance between Sunnyvale and Geneva is 10, 037km and that between Sunnyvale and Baltimore is 3,948km.

averaged over an hour. The aggregate throughput with two flows was 1,797Mbps with `txqueuelen` = 100 packets.

The comparison is summarized in the first three rows of Table 1, where results from Linux TCP, using large `txqueuelen`, are shown in parentheses. The throughput in each experiment is the ratio of

#flow	throughput Mbps	utilization	delay ms	distance km	duration s	bmps 10^{15}	transfer GB
1	925 (266)	95% (27%)	180	10,037	3,600	9.28 (2.67)	387 (111)
2	1,797 (931)	92% (48%)	180	10,037	3,600	18.03 (9.35)	753 (390)
7	6,123	90%	85	3,948	21,600	24.17	15,396
9	7,940	90%	85	3,948	4,030	31.35	3,725
10	8,609	88%	85	3,948	21,600	33.99	21,647

Table 1: SC2002 FAST experimental results: average statistics. Statistics in parentheses are for current TCP implementation in Linux v2.4.18 obtained on January 27-28, 2003.

total amount of data transferred and the duration of the transfer. Utilization is the ratio of throughput and bottleneck capacity (gigabit Ethernet card), excluding the (40-byte) overhead of TCP/IP headers. The "bmps" column is the product of throughput and distance of transfer, measured in bit-meter-per-second. It is the combination of high capacity and large distance that causes performance problems, and this is measured by "bmps". Delay is the minimum round trip time. The throughput traces for some of these experiments are shown in Figure 2.

Also shown in Table 1 are aggregate statistics for 7, 9, and 10-flow experiments using FAST with `txqueuelen` = 100 packets. Their throughput traces are shown in Figure 3. In particular, with 10 flows,

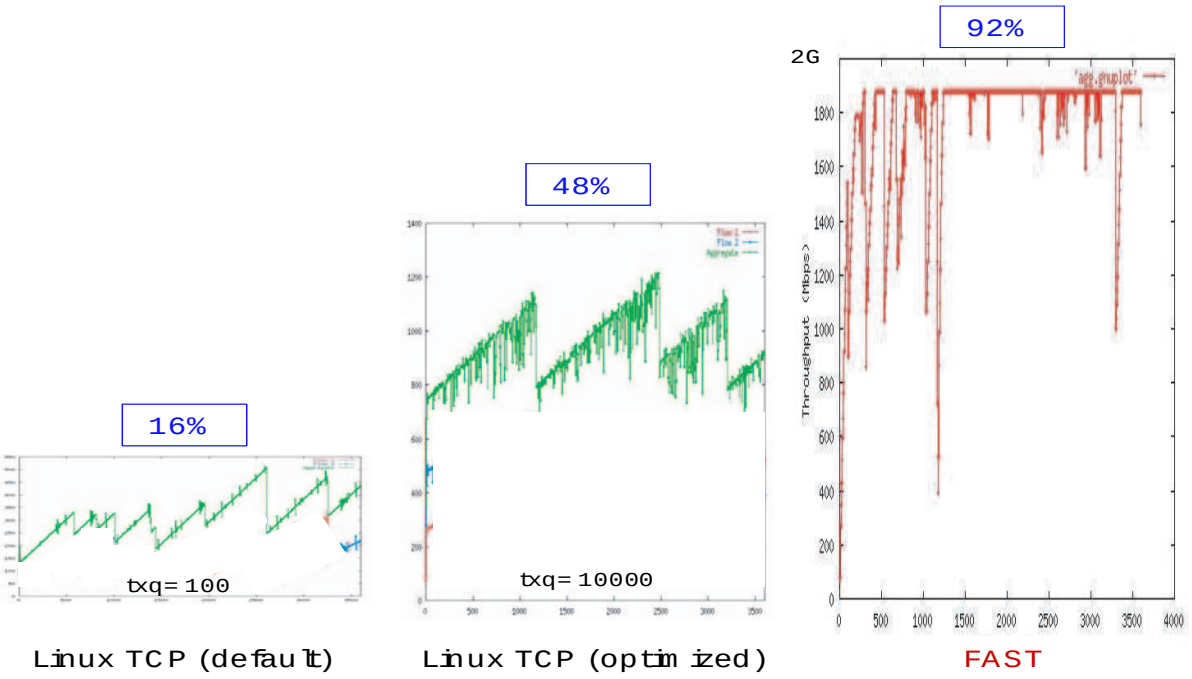

Figure 2: Aggregate throughput traces of 2 flows. From left: Linux (`txqueuelen` = 100), Linux (`txqueuelen` = 10,000), FAST (`txqueuelen` = 100); x-axis is time, y-axis is aggregate throughput, and percentage is utilization.

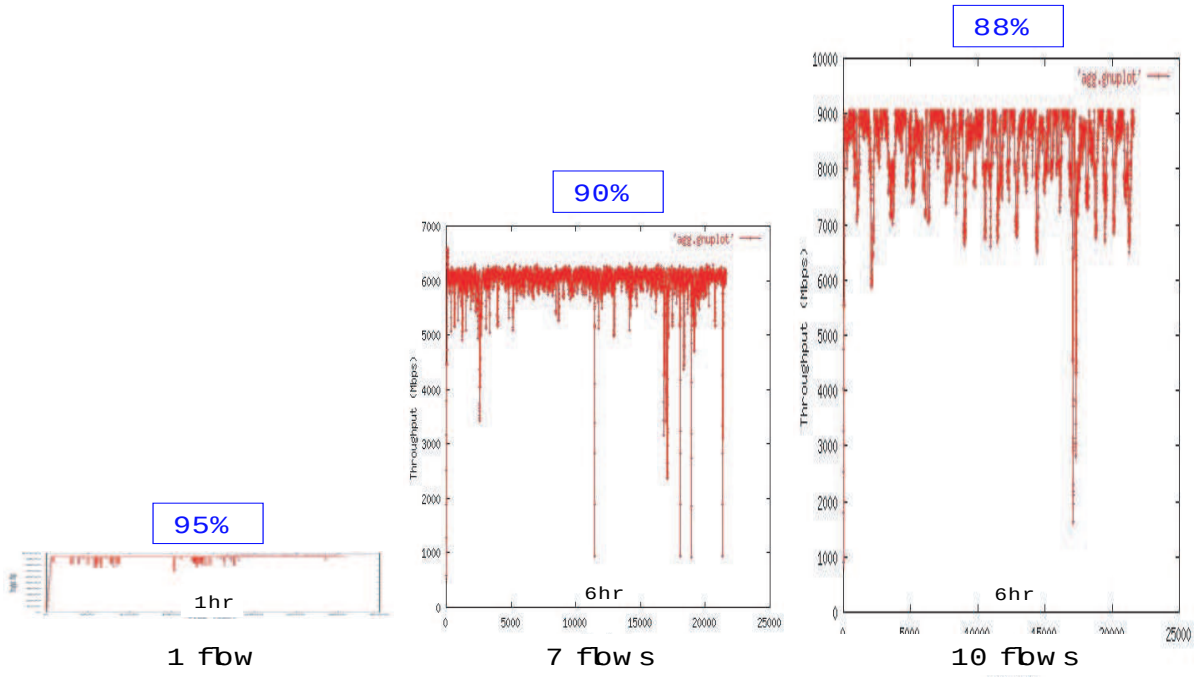

Figure 3: Aggregate throughput traces for FAST experiments in Table 1. From left: 1 flow, 7 flows, 10 flows; x-axis is time, y-axis is aggregate throughput, and percentage is utilization.

FAST TCP achieved an aggregate throughput of 8,609 Mbps and utilization of 88%, averaged over a 6-hour period, over a routed path between Sunnyvale in California and Baltimore in Maryland, using the standard MTU, apparently the largest aggregate throughput accomplished in such a configuration by then as far as we know. These traces, especially those for 9 and 10 flows, display stable reduction in throughput over several intervals of several minutes each, suggesting significant sharing with other conference participants of network bandwidth. We were unable to calibrate our results using current Linux TCP implementation for 7, 9, and 10-flow experiments because the path between StarLight in Chicago and the conference showfloor in Baltimore is not available after SC2002. The path between Sunnyvale and CERN remained available to us until end of February 2003 and allowed us to calibrate the 1 and 2-flow experiments after the conference.

6 Conclusions

We have described the development of FAST TCP, from background theory to actual implementation and its first demonstration. Unlike TCP Reno and its variants, FAST TCP is delay-based. This allows it to achieve high utilization without having to fill the buffer and incur large queueing delay, as loss-based algorithms often do. It achieves proportional fairness and does not penalize flows with large RTTs.

The experiments described in this paper were carried out in relatively simple scenarios. Even though some of the experiments involved multiple flows with heterogeneous delays in the presence of background traffic, the intensity of the background traffic was generally low and our own TCP flows were long-lived. Whether FAST TCP can converge rapidly, yet stably, to a fair allocation, in a dynamic environment where flows of heavy-tailed sizes join and depart in a random fashion, and in the presence of current TCP flows needs a lot more evaluation. Some of these experiments are reported in [1, 7].

Acknowledgments: A global experiment such as the one reported here requires the contribution of a large number of people. We gratefully acknowledge the support of the

- Caltech team, in particular, C. Chapman, C. Hu (Williams/Caltech), J. Pool, J. Wang and Z. Wang (UCLA)
- CERN team, in particular, P. Moroni
- Cisco team, in particular, B. Aiken, V. Doraiswami, M. Potter, R. Sepulveda, M. Turzanski, D. Walsten and S. Yip. Cisco also loaned the GSR 12406 router at Sunnyvale, and additional modules at Starlight, CERN and Sunnyvale.
- DataTAG team, in particular, E. Martelli and J. P. Martin-Flatin.
- LANL team, in particular, G. Hurwitz, E. Weigle, and A. Engelhart
- Level(3) team, in particular, P. Fernes and R. Struble; Level(3) also donated the OC192 link between StarLight in Chicago and Level(3) PoP in Sunnyvale.
- SCinet team, in particular, G. Goddard and J. Patton
- SLAC team, in particular, C. Granieri C. Logg, I. Mei, W. Matthews, R. Mount, J. Navratil and J. Williams
- StarLight team, in particular, T. deFanti and L. Winkler
- TeraGrid team, in particular, L. Winkler

and the funding support of European Commission (Grant IST-2001-32459), US Army Research Office (Grant DAAD19-02-1-0283), US Department of Energy (Grants DE-AC03-76SF00515, DE-FG03-92-ER40701, and W-7405-ENG-36), US National Science Foundation (Grants ANI-0113425 and ANI-0230967).

References

[1] Cheng Jin, David X. Wei, and Steven H. Low. TCP FAST: motivation, architecture, algorithms, performance. In *Proceedings of IEEE Infocom*, March 2004. `http://netlab.caltech.edu`.

[2] 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.

[3] 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.

[4] 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.

[5] Sylvain Ravot. GridDT. The 1st International Workshop on Protocols for Fast Long-Distance Networks, `http://sravot.home.cern.ch/sravot/GridDT/GridDT.htm`, February 2003.

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

[7] Sanjay Hegde, David Lapsley, Bartek Wydrowski, Jan Lindheim, David Wei, Cheng Jin, Steven Low, and Harvey Newman. FAST TCP in high speed networks: An experimental study. In *Proceeding of GridNets*, October 2004.

[8] 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`.

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

[10] 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.

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

[12] 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, to appear*, 2004.

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

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

[15] 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.

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