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/

March 30, 2003

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 preliminary experimental results.

Keywords: FAST TCP, large bandwidth-delay product, high speed TCP, multi-Gbps experiment

Contents

*Submitted to IEEE Communications Magazine, Internet Technology Series, April 1, 2003.

†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.

1 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, is that the current congestion control algorithm of TCP does not scale to this regime.

Our goal is to develop the theory and algorithms for ultrascale networking, implement and test them in state-of-the-art testbeds, and deploy them in communities that need it urgently.

2 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. It is useful both in understanding the performance problems of the current congestion control algorithm and in designing better algorithms to solve these problems, while maintaining fairness in resource allocation.

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 a measure of congestion 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 [1, 2]. 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,[1] forms a distributed feedback system, the largest man-made feedback system in deployment. In this system, hundreds of millions 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 infor-

[1]We will henceforth refer it as a "TCP algorithm" even though we really mean the congestion control algorithm in TCP.

mation. Their interactions result in a collective behavior, whose equilibrium and stability properties we now discuss.

2.1 Equilibrium and performance

We can interpret TCP/AQM as a distributed algorithm over the Internet to solve a global optimization problem [3, 2]. The solution of the optimization problem and that of an associated problem 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, 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, to solve the utility maximization problem and the associated dual problem. 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 has converged, 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.

While the role of TCP is to define the utility function and hence the primal problem, the role of AQM is to ensure that the complementary slackness condition of the primal problem is satisfied. Physically, this means that AQM should try to maximize link utilization at every bottleneck link, an implicit goal of most AQM designs.

2.2 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 network is perturbed out of equilibrium by random fluctuations such as arrival or departure of 'mice' traffic (short-duration TCP flows), or when the equilibrium point shifts because of changes in network topology or flow pattern, the network will drift back to the (new) equilibrium point. Stability is important for another reason. We currently do not have a theory to predict the network behavior when it loses stability. It is hence risky 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 the current TCP algorithm can become unstable as delay increases, or more strikingly, as network capacity increases [4, 5]! The analysis in [5] suggests that the high control gain introduced by TCP is mainly responsible for the instability. The gain increases rapidly with delay or capacity, making it very difficult for any AQM algorithm to stabilize the current TCP. This underlies the well-known difficulty of tuning RED parameters: 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 wild oscillations and reduced utilization.

The lack of scalability of TCP due to both equilibrium and stability problems is illustrated in Figure 1 with packet-level simulations using `ns-2` (Network Simulator). The figure shows the link utilization of TCP/RED and that of FAST. The simulation involved a single bottleneck link with 100ms round

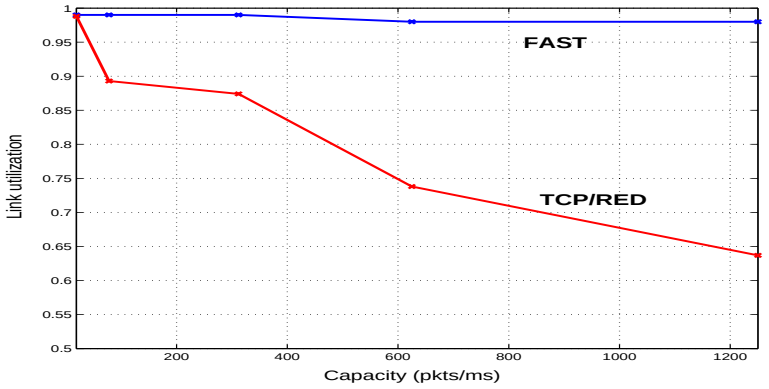


Figure 1: Link utilization of TCP/RED and FAST at bandwidth from 155Mbps to 10Gbps (packet size = 1KB).

trip propagation delay shared by 100 NewReno FTP sources. 'Gentle RED' marking is used at the bottleneck link with large buffer capacity to avoid packet loss. The link capacity varies from 155Mbps to 10Gbps. Packet are all 1000 bytes in size. Simulations show that as link capacity increases, the utilization under TCP/RED drops steadily, in stark contrast to that under FAST.

This has motivated dynamic adjustment of AIMD parameters to alleviate instability, with more aggressive increase and less severe decrease as the window size grows [6, 7].

Two types of TCP/AQM algorithms are proposed in [8] and [9] that can be proved to maintain linear stability at high capacity and large delay in general networks with arbitrary routing and load. While both of these algorithms are decentralized, they are complementary in many ways. The algorithms in [8], called primal algorithms, allow general utility functions, and hence arbitrary fairness in rate allocation, but give up tight control on utilization. The algorithms in [9], 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 and links should scale down their responses by their individual capacities.

By adding slow timescale adaptation to the link algorithm, the primal algorithms can be made to achieve both arbitrary fairness and high utilization [10]. The primal approach motivates a TCP implementation tailored for high bandwidth-delay product regime [11].

By adding slow timescale adaptation to the source algorithm, the dual algorithms can also be made to achieve both arbitrary fairness and high utilization [12]. Moreover, combining this with the insight of [2] leads to a TCP algorithm that can maintain linear stability without having to change the current link algorithm [13]. This approach also allows an incremental deployment strategy where performance steadily improves as ECN (Explicit Congestion Notification) deployment proliferates. These theoretical results suggest that, by modifying just the TCP kernel at the *sending hosts*, we can stabilize the Internet

with the current routers. This provides the motivation for the implementation of FAST TCP.

3 Implementation

The implementation of FAST TCP involves a number of innovations that are crucial to achieve scalability. As the Internet scales up in speed and size, its stability and performance become harder to control. The emerging theory that allows us to understand the equilibrium and stability properties of large networks under end-to-end control forms the foundation of FAST TCP. It plays an important role in its implementation by providing a framework to understand issues, clarify ideas and suggest directions, leading to a more robust and better performing implementation.

Even though theory offers a class of algorithms that can avoid the pitfalls of the current algorithms and maintain stability, their implementation must address several problems that are not captured in the theoretical model and that become severe in the multi-Gbps regime because of the large window size, measured in packets. For instance, the peak window size in our experiments described in the next section was 14,255 packets with 1500-byte MTU (Maximum Transmission Unit). We next briefly summarize four main issues that FAST TCP deals with. Detailed description of the implementation will be reported in a separate paper.

First, we make use of both queueing delay and packet loss as signals of congestion. If only packet loss is used, sources must periodically push buffers to overflow in order to generate the target loss probability, even if the network is static, thus inducing a jittery behavior. The 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 the network scales up in capacity [9, 13, 12].

Second, packet losses are unavoidable, and FAST TCP must deal with massive losses effectively. At large window sizes, thousands of packets can be lost in a single loss event. These packets must be retransmitted rapidly, yet in a way that does not exacerbate congestion and lead to more losses and timeouts. During the recovery of massive losses, round-trip time (RTT) measurements and acknowledgment clocking are lost. Delay information is unavailable for the entire duration of recovery and hence the window size must be controlled based on loss and recovery information.

Third, traffic can be extremely bursty at large window sizes, 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 increases the likelihood of massive losses, which not only wastes bandwidth on retransmission, more importantly, they cause the loss of RTT measurement and acknowledgment clocking. To reduce burstiness and massive losses, FAST TCP employs pacing at the sender.

Finally, after loss recovery, the window size can be far from its equilibrium value and RTT measurements are newly re-acquired and noisy. FAST TCP must converge rapidly to a neighborhood of the equilibrium value and then smoothly home in on the target. In a dedicated environment, this is easy to achieve. The challenge is to converge rapidly, yet stably, to a fair allocation, in a dynamic environment where flows join and depart in a random fashion.

4 Experimental results

FAST TCP was first demonstrated publicly in 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.

We are working to extend these preliminary results and to improve and evaluate the stability, responsiveness, fairness of FAST TCP, and its interaction with the current protocols.

4.1 Infrastructure

The demonstrations used an OC192 (10 Gbps) link donated by Level(3) 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 GSR 12406 router loaned by Cisco at Sunnyvale, additional Cisco modules loaned at Starlight, CERN and 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. In some of our experiments conducted after SC2002, some of the servers at Sunnyvale, Chicago and Geneva were also equipped with Intel's pre-release 10-gigabit Ethernet cards. The network setup is shown in Figure 2.

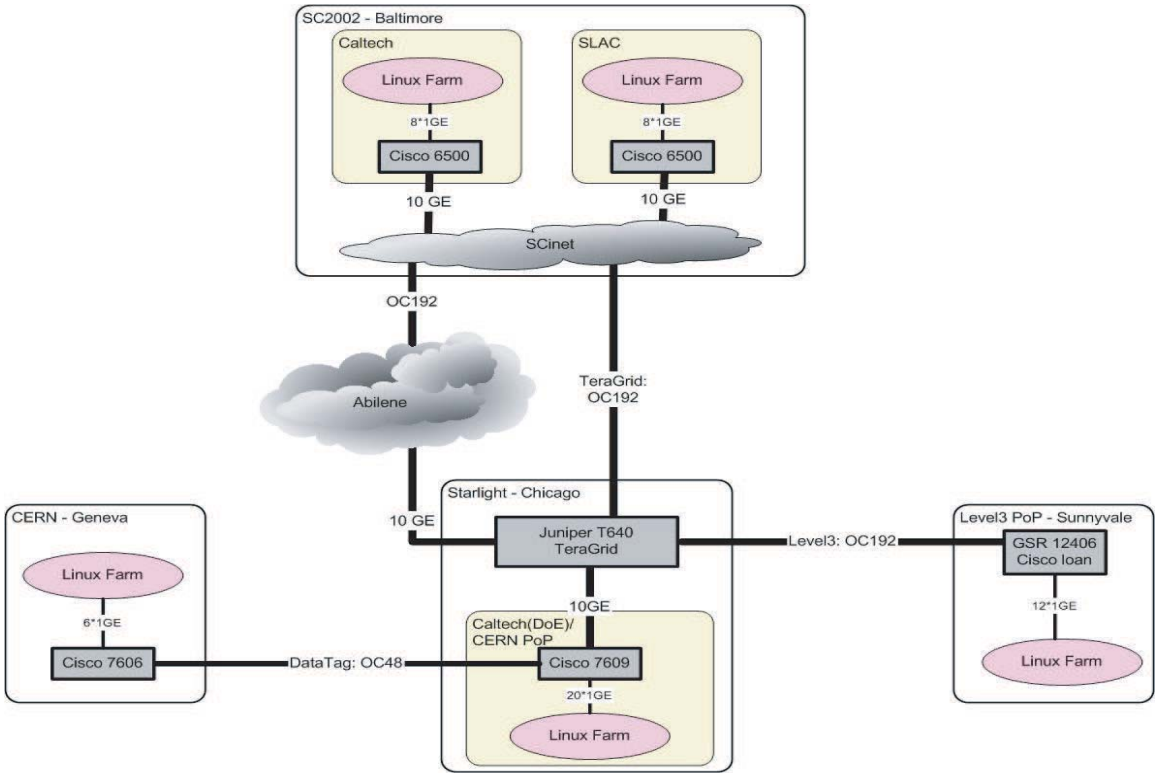


Figure 2: Network setup in SC2002, Baltimore, MD, November 16-22, 2002

We have conducted a number of experiments, all using the standard MTU (Maximum Transmission Unit), 1500 bytes including TCP and IP headers.

4.2 Throughput and utilization

In this subsection, we report our SC2002 experiments on throughput and utilization.

To calibrate, 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. This set of calibration experiments was conducted on January 27-28, 2003 after the SC2002 conference using the same testbed shown in Figure 2.

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, 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 these experiments are shown in Figure 3.

Also shown in Table 1 are aggregate statistics for 7, 9, and 10-flow experiments using FAST with `txqueuelen` = 100 packets.[2] Their throughput traces are shown in Figure 4. In particular, with 10 flows, 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 ever accomplished in such a configuration as far as we know. These traces, especially those for 9 and 10 flows, display stable

[2]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.

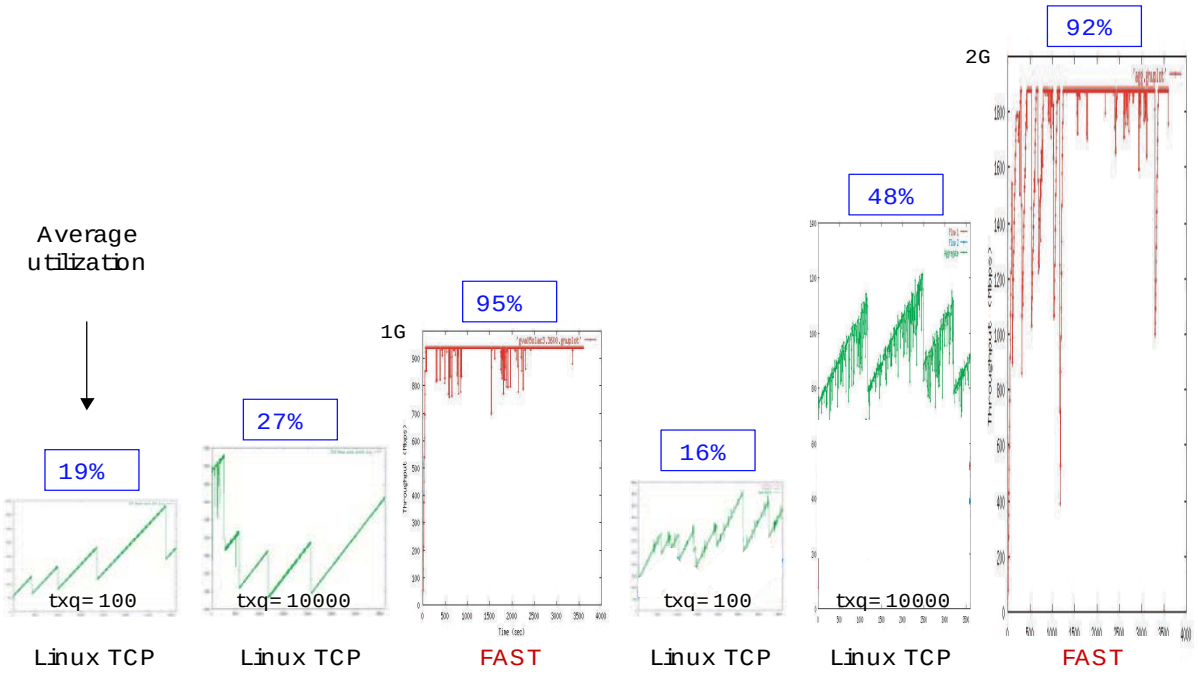


Figure 3: Throughput traces for FAST and Linux TCP. From left: 1 flow (Linux, `txqueuelen` = 100), 1 flow (Linux, `txqueuelen` = 10,000), 1 flow (FAST, `txqueuelen` = 100), 2 flow (Linux, `txqueuelen` = 100), 2 flow (Linux, `txqueuelen` = 10,000), 2 flow (FAST, `txqueuelen` = 100); x-axis is time, y-axis is aggregate throughput, and percentage is utilization.

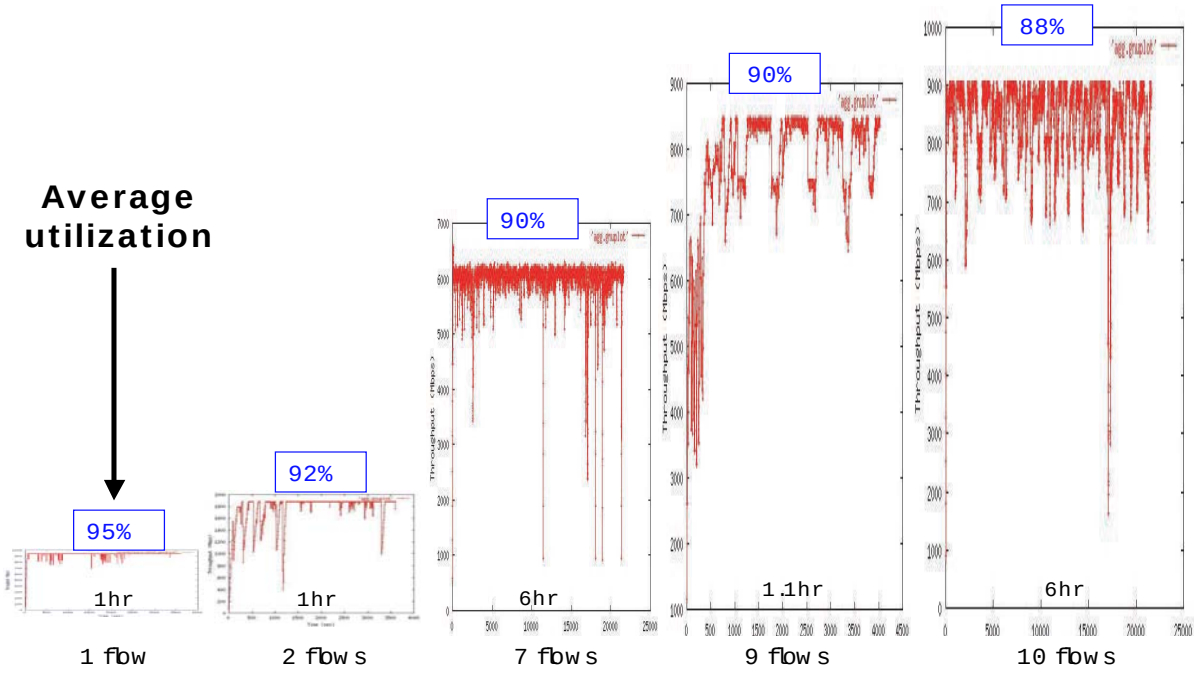


Figure 4: Throughput traces for FAST experiments in Table 1. From left: 1 flow, 2 flows, 7 flows, 9 flows, 10 flows; x-axis is time, y-axis is aggregate throughput, and percentage is utilization.

reduction in throughput over several intervals of several minutes each, suggesting significant sharing with other conference participants of network bandwidth.

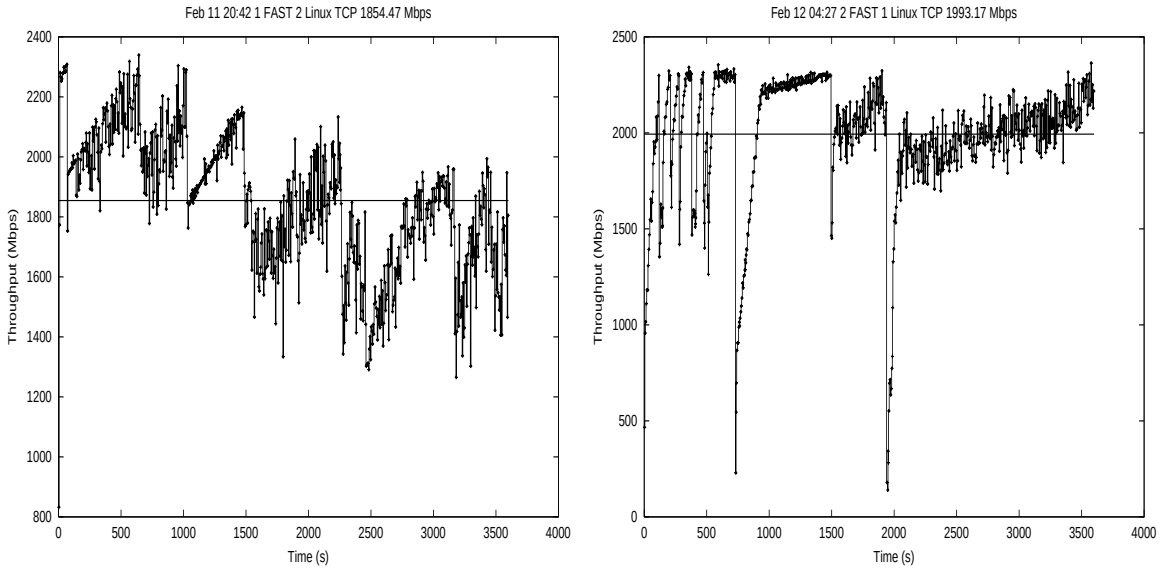
In all the experiments reported above and in the next subsection, the bottleneck is either the gigabit Ethernet card or the transatlantic OC48 link. We conducted some tests in February 2003 on the testbed in Figure 2, using Intel's pre-release experimental 10-gigabit Ethernet card. FAST TCP sustained just 1.3Gbps using the standard MTU from Sunnyvale to Chicago, limited by the CPU power of the sending and receiving systems at the ends of the network path.[3]

4.3 Fairness

Fairness between FAST TCP and Linux TCP is an issue still under study. In this subsection, we present some preliminary experimental results obtained on February 11-12, 2003, from the testbed in Figure 2.

We present two experiments, each involving three concurrent flows, one flow between each pair of servers. One flow ran from CERN in Geneva to Sunnyvale, CA and two from CERN to StarLight in Chicago. In this scenario, each flow ran between a pair of gigabit Ethernet cards, so the bottleneck for the three concurrent flows was the DataTAG transatlantic OC48 link, which had a data rate of 2.4Gbps. These flows are either FAST TCP (with `txqueuelen` = 100 packets) or Linux TCP (with `txqueuelen` = 10,000 packets), all using the standard MTU of 1,500 bytes. All experiments were run for an hour. Throughput traces and average statistics were collected.

In the first experiment, shown in Figure 5(a), the Geneva-Sunnyvale flow is FAST TCP and the two Geneva-Chicago flows are Linux TCP. The single FAST TCP flow attained an average throughput of



(a) 1 FAST and 2 Linux TCP flows

(b) 2 FAST and 1 Linux TCP flows

Figure 5: Aggregate throughput of three flows from Geneva to Chicago and Sunnyvale: 1,854Mbps with 1 FAST and 2 Linux TCP flows, and 1,993Mbps with 2 FAST and 1 Linux TCP flows.

760Mbps, while the two Linux TCP flows attained average throughputs of 648Mbps and 446Mbps.

In the second experiment, shown in Figure 5(b), the Geneva-Sunnyvale flow is Linux TCP and the two Geneva-Chicago flows are FAST TCP. The single Linux TCP flow attained an average throughput

[3]With 9,000-byte MTU, Linux, FAST and Scalable TCP all sustained more than 2.35Gbps on a single flow between Sunnyvale and Geneva, apparently limited by the transatlantic link. HSTCP sustained 1.8Gbps in that experiment. We emphasize that these experiments are preliminary and not yet conclusive.

of 419Mbps, while the two FAST TCP flows attained average throughputs of 841Mbps and 732Mbps.

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

[2] 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`.

[3] Steven H. Low. A duality model of TCP and queue management algorithms. *IEEE/ACM Trans. on Networking, to appear*, October 2003. `http://netlab.caltech.edu`.

[4] Chris Hollot, Vishal Misra, Don Towsley, and Wei-Bo Gong. A control theoretic analysis of RED. In *Proceedings of IEEE Infocom*, April 2001. `http://www-net.cs.umass.edu/papers/papers.html`.

[5] S. H. Low, F. Paganini, J. Wang, S. A. Adlakha, and J. C. Doyle. Dynamics of TCP/RED and a scalable control. In *Proc. of IEEE Infocom*, June 2002. `http://netlab.caltech.edu`.

[6] Sally Floyd. HighSpeed TCP for large congestion windows. Internet draft draft-floyd-tcp-highspeed-01.txt, `http://www.icir.org/floyd/hstcp.html`, 2002.

[7] 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.

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

[9] 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`.

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

[11] 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.

[12] 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`.

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