SLAC-PUB-10402
April 2004
Revised
May 2004

Evaluation of Advanced TCP Stacks on Fast Long-Distance Production Networks

Hadrien Bullot
School of Computer & Communication Sciences
Swiss Federal Institute of Technology, Lausanne
hadrien@slac.stanford.edu

R. Les Cottrell
Stanford Linear Accelerator Center
Stanford University, Menlo Park
cottrell@slac.stanford.edu

Richard Hughes-Jones
Department of Physics and Astronomy
The University of Manchester, Manchester
R.Hughes-Jones@man.ac.uk

Abstract[1]

With the growing needs of data intensive science, such as High Energy Physics, and the need to share data between multiple remote computer and data centers worldwide, the necessity for high network performance to replicate large volumes (TBytes) of data between remote sites in Europe, Japan and the U.S. is imperative. Currently, most production bulk-data replication on the network utilizes multiple parallel standard (Reno based) TCP streams. Optimizing the window sizes and number of parallel stream is time consuming, complex, and varies (in some cases hour by hour) depending on network configurations and loads. We therefore evaluated new advanced TCP stacks that do not require multiple parallel streams while giving good performances on high speed long-distance network paths. In this paper, we report measurements made on real production networks with various TCP implementations on paths with different Round Trip Times (RTT) using both optimal and sub-optimal window sizes.

We compared the New Reno TCP with the following stacks: HS-TCP, Fast TCP, S-TCP, HSTCP-LP, H-TCP and Bic-TCP. The analysis will compare and report on the stacks in terms of achievable throughput, impact on RTT, intra- and inter-protocol fairness, stability, as well as the impact of reverse traffic.

We also report on some tentative results from tests made on unloaded 10 Gbps paths during SuperComputing 2003.

1 Introduction

With the huge amounts of data gathered in fields such as High Energy and Nuclear Physics (HENP), Astronomy, Bioinformatics, Earth Sciences, and Fusion, scientists are facing unprecedented challenges in managing, processing, analyzing and transferring the data between major sites like major research sites in Europe and North America that are separated by long distances. Fortunately, the rapid evolution of high-speed networks is enabling the development of data-grids and super-computing that, in turn, enable sharing vast amounts of data and computing power. Tools built on TCP, such as bbcp [11], bbftp [4] and GridFTP [1] are increasingly being used by applications that need to move large amounts of data.

The standard TCP (Transmission Control Protocol) has performed remarkably well and is generally known for having prevented severe congestion as the Internet scaled up. It is well-known that the current version of TCP - which relies on the Reno congestion avoidance algorithm to measure the capacity of a network - is not appropriate for high speed long-distance networks. The need to acknowledge packets sets a limit for the throughput for Reno TCP to be a function[2] of $1/RTT$ where RTT is the Round Trip Time. For example, with 1500-Byte packets and a 100 ms RTT, it would require an average congestion window of 83,333 segments and a packet drop rate of at most one congestion event every 5,000,000,000 packets to

[1] This work was supported in part by the Director, Office of Science, Office of Advanced Scientific Computing Research, Mathematical, Information, and Computational Sciences Division under the U.S. Department of Energy; and part by the UK e-Science Initiative though the Particle Physics & Astronomy Research Council.The SLAC work is under Contract No. DE-AC03-76SF00515.

[2] The macroscopic behavior of the TCP congestion avoidance algorithm by Mathis, Semke, Mahdavi & Ott in *Computer Communication Review, 27(3), July 1997*

Work supported by the Department of Energy Contract DE-AC03-76SF00515

achieve a steady-state throughput of 10 Gbps (or equivalently, at most one congestion event every 100 minutes)[8]. This loss rate is typically below what is possible today with optical fibers.

Today the major approach, on production networks, to improve the performance of TCP is that of adjusting the TCP window size to the bandwidth (or more accurately the bitrate) * delay (RTT) product (BDP) of the network path, and using parallel TCP streams.
In this paper, we provide an independent (of the TCP stack developers) analysis of the performance and the fairness of various new TCP stacks. We ran tests in 3 network configurations: short distance, middle distance and long distance. With these different network conditions, our goal is to find a protocol that is easy to configure, that provides optimum throughput, that is network friendly to other users, and that is stable to changes in available bitrates. We tested 7 different TCP stacks (see section 2 for a brief description of each): P-TCP, S-TCP, Fast TCP, HS-TCP, HSTCP-LP, H-TCP and Bic-TCP. The main aim of this paper is to compare and validate how well the various TCP stacks work in real high-speed production networks.

Section 2 describes the specifications of each advanced protocol we tested. Section 3 explains how we made the measurements. Section 4 shows how each protocol: affects the RTT and CPU loads, and behaves with respect to the txqueuelen (the number of packets queued up by the IP layer for the Network Interface Card (NIC)). This section also shows: how much throughput each protocol can achieve; how stable is each protocol in the face of “stiff” sinusoidally varying UDP traffic; and the stability of each protocol. Section 5 moves on to consider the effects of cross-traffic on each protocol. We consider both cross-traffic from the same protocol (intra-protocol) and a different protocol (inter-protocol). We also look at the effects of the reverse traffic on the protocols. Section 6 reports on some tentative results from tests made during SuperComputing 20003 (SC03). Section 7 talks about possible future measurements and section 8 provides the conclusion.

2 The advanced stacks

We selected the following TCP stacks according to two criteria in order to achieve high throughput on long distance:

> **Software change** Since most data-intensive science sites are end-users of networks - with no control over the routers or infrastructure of the wide area network - we required that any changes needed would only apply to the end-hosts. Thus, for standard production networks, protocols like XCP [15] (router assisted protocol) or Jumbo Frame (e.g. MTU=9000) are excluded. Furthermore, since our sites are major generators and distributors of data, we wanted a solution that only required changes to the sender end of a transfer. Consequently we eliminated protocols like Dynamic Right Sizing [5], which required a modification on the receiver’s side.
>
> **TCP improvement** Given the existing software infrastructure based on file transfer applications such as bbftp, bbcp and GridFTP that are based on TCP, and TCP’s success in scaling up to the Gbps range [6], we restricted our evaluations to implementations of the TCP protocol. Rate based protocols like SABUL [9] and Tsunami [21] or storage based protocols such as iSCSI or Fibre Channel over IP and circuit oriented solutions are currently out of scope.

We call *advanced stacks* the set of protocols presented below, except the first (TCP Reno). All of these stacks are improvements of TCP Reno apart from Fast TCP that is an evolution from TCP Vegas. All the stacks only require to be used on the sender’s side. Further all the advanced stacks run on GNU/Linux.

2.1 Reno TCP

TCP’s congestion management is composed of two major algorithms: the slow-start and congestion avoidance algorithms which allow TCP to increase the data transmission rate without overwhelming the network. Standard TCP cannot inject more than *cwnd* (congestion window) segments of unacknowledged data into the network. TCP Reno’s congestion avoidance mechanism is referred to as AIMD (Additive Increase Multiplicative Decrease). In the congestion avoidance phase TCP Reno increases *cwnd* by one packet per packet of data acknowledged and halves *cwnd* for every window of data containing a packet drop. Hence the following equations:

Slow-Start

$$\mathrm{ACK}: \mathrm{new}_{\mathrm{cwnd}} = \mathrm{old}_{\mathrm{cwnd}} + \mathrm{c} \qquad (1)$$

Congestion Avoidance

$$\mathrm{ACK}: \mathrm{new}_{\mathrm{cwnd}} = \mathrm{old}_{\mathrm{cwnd}} + \frac{\mathrm{a}}{\mathrm{old}_{\mathrm{cwnd}}} \qquad (2)$$

$$\mathrm{DROP}: \mathrm{new}_{\mathrm{cwnd}} = \mathrm{old}_{\mathrm{cwnd}} - b * \mathrm{old}_{\mathrm{cwnd}} \qquad (3)$$

Where a = 1, b = 0.5, c = 1.

2.2 P-TCP

After tests with varying maximum window sizes and numbers of streams, from our site to many sites, we observed that using the TCP Reno protocol with 16 streams and an appropriate window size (typically the number of streams * window size ~ BDP) was a reasonable compromise for medium and long network distance paths. Since today physicists are typically using TCP Reno with multiple parallel streams to achieve high throughputs, we use this number of streams as a base for the comparisons with other protocols. However:

- It may be over-aggressive and unfair
- The optimum number of parallel streams can vary significantly with changes (e.g., routes) or utilization of the networks.

To be effective for high performance throughput, the best new advanced protocols, while using a single stream, need to provide similar performance to P-TCP (parallel TCP Reno) and in addition, they should have better fairness than P-TCP.

For this implementation, we used the latest GNU/Linux kernel available (2.4.22) which includes SACK [RFC 2018] and New Reno [RFC 2582]. This implementation still has the AIMD mechanism shown in (2) and (3).

2.3 S-TCP

Scalable TCP changes the traditional TCP Reno congestion control algorithm: instead of using Additive Increase, the increase is exponential and the Multiplicative Decrease factor b is set to 0.125 to reduce the loss of throughput following a congestion event. It was described by Tom Kelly in [16].

2.4 Fast TCP

The Fast TCP protocol is the only protocol which is based on Vegas TCP instead of Reno TCP. It uses both queuing delay and packet loss as congestion measures. It was introduced by Steven Low and his group at Caltech in [14] and demonstrated during SC2002 [13]. It reduces massive losses using pacing at sender and converges rapidly to an equilibrium value.

2.5 HS-TCP

The HighSpeed TCP was introduced by Sally Floyd in [7] and [8] as a modification of TCP's congestion control mechanism to improve the performance of TCP in fast, long delay networks. This modification is designed to behave like Reno for small values of *cwnd*, but above a chosen value of *cwnd* a more aggressive response function is used. When *cwnd* is large (greater than 38 packets equivalent to a packet loss rate of 1 in 1000), the modification uses a table to indicate by how much the congestion window should be increased when an ACK is received, and it releases less network bandwidth than 1/2 *cwnd* on packet loss. We were aware of two versions of High-Speed TCP: Li [18] and Dunigan [3]. Apart from the SC03 measurements, we chose to test the stack developed by Tom Dunigan which was included in the Web100[3] patch.

2.6 HSTCP-LP

The aim of this modification, which is based on TCP-LP [17], is to utilize only the excess network bandwidth left unused by other flows. By giving a strict higher priority to all non-HSTCP-LP cross-traffic flows, the modification enables a simple two-class prioritization without any support from the network. HSTCP-LP was implemented by merging together HS-TCP and TCP-LP.

2.7 H-TCP

This modification has a similar approach to High-Speed TCP since H-TCP switches to the advanced mode after it has reached a threshold. Instead of using a table like HS-TCP, H-TCP uses an heterogeneous AIMD algorithm described in [24].

2.8 Bic-TCP

In [26], the authors introduce a new protocol whose objective is to correct the RTT unfairness of Scalable TCP and HS-TCP. The protocol uses an additive increase and a binary search increase. When the congestion window is large, additive increase with a large increment ensures linear RTT fairness as well as good scalability. Under small congestion windows, binary search increase is designed to provide TCP friendliness.

3 Measurements

Each test was run for 20 minutes from our site to three different networks: Caltech for short-distance (minimum RTT of 10 ms), University of Florida (UFL) for middle distance (minimum RTT of 70 ms) and University of Manchester for long-distance (minimum RTT of 170 ms). We duplicated some tests to DataTAG[4] Chicago (minimum RTT of 70 ms) and DataTAG CERN (minimum RTT of 170 ms) in order to see if our tests were coherent. We ran all the tests once. Some tests were duplicated in order to see if we can get the same result again.

[3] http://www.web100.org

[4] Research & Technological Development for a Transatlantic Grid: http://datatag.web.cern.ch/datatag/

These duplicated tests corroborated our initial findings. The tests were run for about 20 minutes, and this helped us determine if the data were coherent.

The throughputs on these production links go from 400 Mbps to 600 Mbps which was the maximum we could reach because of the OC12/POS (622 Mbps) links to ESnet and CENIC at our site. The route for Caltech uses CENIC from our site to Caltech and the bottleneck capacity for most of the tests was 622 Mbps. The route used for UFL was CENIC and Abilene and the bottleneck capacity was 467 Mbps at UFL. The route to CERN was via ESnet and Starlight and the bottleneck capacity was 622 Mbps at our site. The route used for University of Manchester is ESnet then GEANT and JANET.

At the sender side, we used three machines:

- **Machine 1** runs `ping`.
- **Machine 2** runs Advanced TCP.
- **Machine 3** runs Advanced TCP for cross-traffic or UDP traffic.

Machines 2 and 3 had 3.06 GHz dual-processor Xeons with 1 GB of memory, a 533 MHz front side bus and an Intel Gigabit Ethernet (GE) interface.

Due to difficulties concerning the availability of hosts at the receiving sites, we usually used only two servers on the receiver's side (Machines 1 and 2 at the sender side send data to the same machine at the receiver side).

After various tests, we decided to run ping and iperf in separate machines. With this configuration we had no packet loss for ping during the tests. We used a modified version of iperf[5] in order to test the advanced protocol in a heterogeneous environment. The `ping` measurements provide the RTT which provide information on how the TCP protocol stack implementations affect the RTT and how they respond to different RTTs. Following an idea described by Hacker [10], we modified iperf to be able to send UDP traffic with a sinusoidal variation of the throughput. We used this to see how well each advanced TCP stack was able to adjust to the varying "stiff" UDP traffic. The amplitude of the UDP stream varied from 5% to 20% of the bandwidth with periods of 60 seconds and 30 seconds. Both the amplitude and period could be specified.

We ran iperf (TCP and UDP flows) with a report interval of 5 seconds. This provided the incremental throughputs for each 5 second interval of the measurement. For the ICMP traffic the interval that was used by the traditional ping program, is of the same order as the RTT in order to gain some granularity in the results. The tests were run mostly during the weekend and the night in order to reduce the impact on other traffic.

On the sender's side, we used the different kernels patched for the advanced TCP stacks. The different kernels are based on vanilla GNU/Linux 2.4.19 through GNU/Linux 2.4.22. The TCP source code of the vanilla kernels is nearly identical. On the receiver's side we used a standard Linux kernel no patches for TCP.

For each test we computed different values: throughput average and standard deviation, RTT average and standard deviation, stability and fairness index. The stability index helps us find out how the advanced stack evolves in a network with rapidly varying available bandwidth.

With iperf, we can specify the maximum sender and receiver window sizes the congestion window can reach. For our measurements we set the maximum sender and receiver window sizes equal. When quoting the maximum window sizes for P-TCP we refer to the window size for each stream. The optimal window sizes according the bandwidth delay product are about 500 KBytes for the short distance path, about 3.5 MBytes for the medium distance path and about 10 MBytes for the long distance path. We used 3 main window sizes for each path in order to try and bracket the optimum in each case: for the short-distance we used 256 KBytes, 512K Bytes and 1024 KBytes; for the middle distance we used 1 MBytes, 4 MBytes and 8 MBytes; and for the long-distance we used 4MByte, 8 MByte and 12 MByte maximum windows. In this paper, we refer to these three different window sizes for each distance as: size 1, 2 and 3.

4 Results

In this section, we present the essential points and the analysis of our results. The data are available on our website[6].

4.1 RTT

All advanced TCP stacks are "fair" with respect to the RTT (i.e. do not dramatically increase RTT) except for P-TCP Reno. On the short distance, the RTT of P-TCP Reno increases from 10 ms to 200 ms. On the medium and long distances, the variation is much less noticeable and the difference in the average RTTs between the stacks is typically less than 10ms.

For the other advanced stacks the RTT remains the same except with the biggest window size we noticed, in general, a small increase of the RTT.

[5] http://dast.nlanr.net/Projects/Iperf/

[6] http://www-iepm.slac.stanford.edu/bw/tcp-eval/

4.2 CPU load

We ran our tests with the **time** command in order to see how each protocol used the CPU resource of the machine on the sender's side. We calculated the MHz/Mbps rating by:

$$\text{MHz/Mbps} = \frac{\text{(CPU Utilization * CPU MHz)}}{\text{Average Throughput}}$$

The MHz/Mbps utilization averaged over all stacks, for all distances and all windows was 0.93 ± 0.08 MHz/Mbps. The MHz/Mbps averaged over all distances and window sizes varied from 0.8± 0.35 for S-TCP to 1.0 ± 0.2 for Fast. We observed no significant difference in sender side CPU load between the various protocols.

	TCP Reno	P-TCP	S-TCP	Fast TCP	HS-TCP	Bic-TCP	H TCP	HSTCP-LP
Caltech 256KB	238±15	395±33	226±14	233±13	225±17	238±16	233±25	236±18
Caltech 512KB	361±44	412±18	378±41	409±27	307±31	372±35	338±48	374±51
Caltech 1MB	374±53	434±17	429±58	413±58	284±37	382±41	373±34	381±51
UFL 1MB	129±26	451±32	109±18	136±12	136±15	134±13	140±14	141±18
UFL 4MB	294±110	428±71	300±108	339±101	431±91	387±52	348±76	382±120
UFL 8MB	274±115	441±52	281±117	348±96	387±95	404±34	351±56	356±118
Manchester 4MB	97±38	268±94	170±20	163±33	171±15	165±26	172±13	87±61
Manchester 8MB	78±41	232±74	320±65	282±113	330±52	277±92	323±64	118±111
Manchester 12MB	182±66	212±83	459±71	262±195	368±161	416±100	439±129	94±113
Avg. thru Size 1	154	371	178	177	177	179	185	155
Avg. thru Size 2	244	357	384	343	356	345	336	292
Avg. thru Size 3	277	362	422	341	346	367	388	277
Avg. thru size 2 & 3	261	360	403	342	351	356	362	294
Std. dev. size 2 & 3	113	107	49	53	54	49	41	125

Table 1: `Iperf` TCP throughputs for various TCP stacks for different window sizes, averaged over the three different network path lengths.

4.3 txqueuelen

In the GNU/Linux 2.4 kernel, the txqueuelen enables us to regulate the size of the queue between the kernel and the Ethernet layer. It is well-known that the size of the txqueuelen for the NIC can change the throughput but we have to use some optimal tuning. Some previous tests [19] were made by Li. Although use of a large txqueuelen can result in a large increase of the throughput with TCP flows and a decrease of sendstall, Li observed an increase of duplicate ACKs.

Scalable TCP by default used a txqueuelen of 2000 but all the others use 100. Thus, we tested the various protocols with txqueuelen sizes of 100, 2000 and 10000 in order to see how this parameter could change the throughput. In general, the advanced TCP stacks perform better with a txqueuelen of 100 except for S-TCP which performs better with 2000. With the largest txqueuelen, we observe more instability in the throughput.

4.4 Throughput

Table 1 and Figure 1 show the `iperf` TCP throughputs averaged over all the 5 seconds intervals for each 1200 second measurement (henceforth referred to as the 1200 second average) together with the standard deviations, for the various stacks, network distances and window sizes. Also shown are the "averages of the 1200 second averages" for the three network distances for each window size. Since the smallest window sizes were unable to achieve the optimal throughputs, we also provide the averages of the 1200 second averages for sizes 2 and 3.

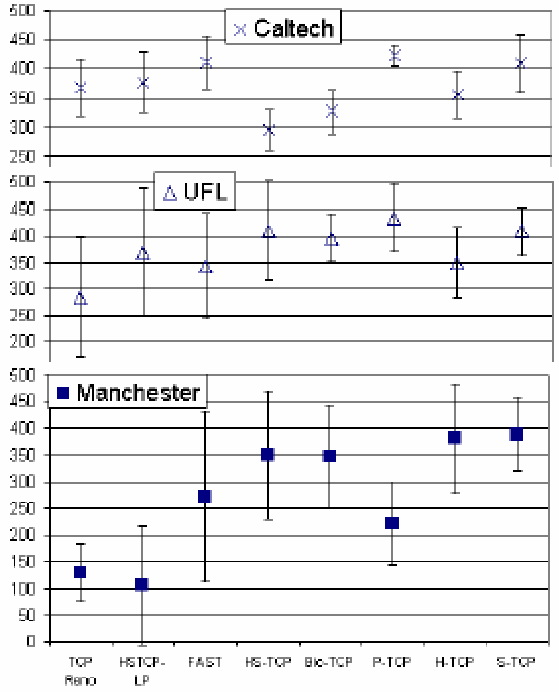

Figure 1: Average of the 1200 second averages for maximum window sizes 2 and 3 shown for three network distances and various TCP stacks. The y axis is the throughput achieved in Mbps.

- With the smallest maximum window sizes (size 1) we were unable to achieve optimal throughputs except when using P-TCP.
- Depending on the paths, we could achieve throughputs varying from 300 to 500 Mbps.
- There are more differences in the protocol achievable throughputs for the longer distances.
- For the long distance (Manchester), the BDP predicts an optimum window size closer to 12 MBytes than 8 Mbytes. As a result S-TCP, H-TCP, Bic-TCP and HS-TCP perform best for the Manchester path with the 12 MByte maximum window size.
- The top throughput performer for window sizes 2 and 3 was Scalable-TCP, followed by (roughly equal) Bic-TCP, Fast TCP, H-TCP, P-TCP and HS-TCP, with HSTCP-LP and Reno single stream bringing up the rear.
- The poor performance of Reno single stream is to be expected due to its AIMD congestion avoidance behavior.
- Since HSTCP-LP deliberately backs off early to provide a lower priority, it is not unexpected that it will perform less well than other more aggressive protocols.
- P-TCP performs well on short and medium distances, but not as well on the long-distance path, possibly since the windows*streams product was >> the BDP.

We note that the standard deviations of these averages are sufficiently large that the ordering should only be regarded as a general guideline.

4.5 Sinusoidal UDP

The throughput of a protocol is not sufficient to describe its performance. Thus, we analyzed how the protocol behaves when competing with a UDP stream varying in a sinusoidal manner. The purpose of this stream is to emulate the variable behavior of the background cross-traffic. Our results show that in general, all protocols converge quickly to follow the changes in the available bandwidth and maintain a roughly constant aggregate throughput - especially for Bic-TCP. Fast TCP, and P-TCP to a lesser extent have, some stability problems on long-distance and become unstable with the largest window size. Figure 2 shows an example of the variation of Bic-TCP in the presence of sinusoidal UDP traffic measured from our site to UFL with an 8 MByte window.

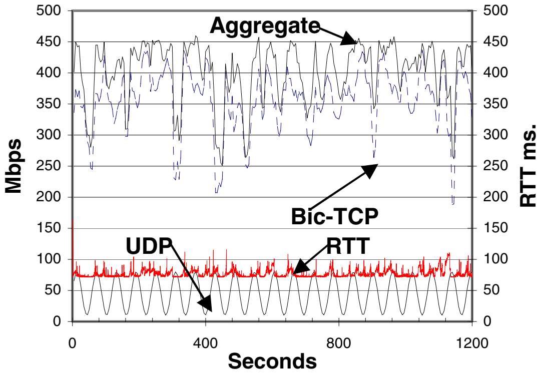

Figure 2: Bic-TCP with sinusoidal UDP traffic.

4.6 Stability

Following [14], we compute the stability index as the standard deviation normalized by the average throughput Index (i.e. standard deviation / average throughput). If we have few oscillations in the throughput, we will have a stability index close to zero.

Figure 3 shows the stability index for each of the stacks for each of the distances averaged over window sizes 2 and 3. Without the UDP cross-traffic, all stacks have better stability indices (factor of 1.5 to 4 times better) with the smallest window sizes (average stability index over all stacks and distances for size 1 = 0.09±0.02, for size 2 =

0.2±0.1 and size 3 = 0.24±0.1). S-TCP has the best stability (index ~ 0.1) for the optimal and larger than optimal window sizes, this is followed closely by H-TCP, Bic-TCP and HS-TCP. Single stream Reno and HSTCP-LP have poorer stabilities (> 0.3).

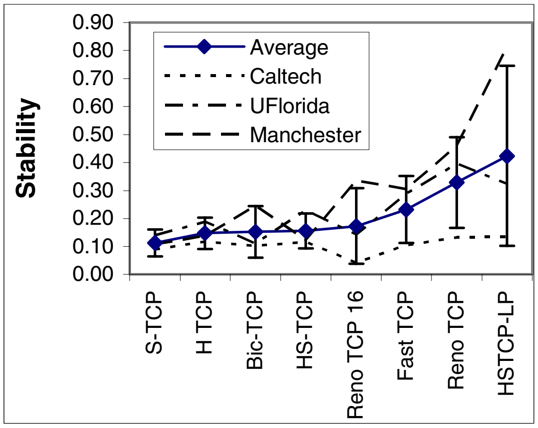

Figure 3: Stability index for the 3 different network paths, averaged over the optimal and largest window sizes. Also shown are the averages and standard deviations over the two window sizes and paths.

With the sinusoidal UDP traffic, better stability is achieved once again with the smallest window sizes (stability index averaged over all stacks and distances for size 1 = 0.13±0.06, size 2= 0.21±0.08, size 3= 0.25±0.01). For the other window sizes (see Figure 4) there is little difference (0.01) between the two UDP-frequency stabilities for a given stack. The throughputs with the UDP cross-traffic are generally larger (15%) than those without the UDP cross-traffic. Bic-TCP closely followed by the two more aggressive protocols, P-TCP and Scalable-TCP, have the best stability indices (< 0.2). H-TCP and HS-TCP have stability indices typically > 0.2 and Fast TCP and HSTCP-LP have stability indices > 0.3

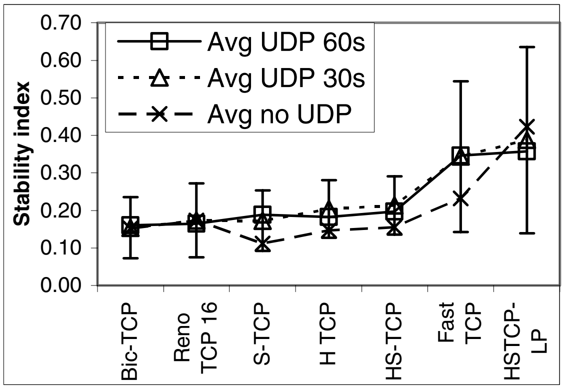

Figure 4: Stability as a function of TCP stack and UDP cross traffic frequency. The data is averaged over window sizes 2 and 3 and network paths.

5 Cross-traffic

5.1 Intra-protocol fairness

The cross-traffic tests are important and help us to understand how fair a protocol is. At our research centers, we wanted to know not only the fairness of each advanced protocol against TCP Reno, but also how fairly the protocols behave towards each other. It is important to see how the different protocols compete with one another since the protocol that our research centers will adopt shortly must coexist harmoniously with existing protocols and with advanced protocols chosen by other sites. Of course, we cannot avoid a future protocol being unfair only with our chosen one. In this paper we consider a fair share per link metric. If there are *n* flows through a bottleneck link, each flow will take 1/*n* of the capacity of the bottleneck link. We measure the average bandwidth x_i of each source *i* during the test then we compute the fairness index as described in [2] by Chiu and Jain :

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

A fairness index of 1 corresponds to a perfect allocation of the throughput between all protocols.

There are other definitions of the concept of fairness. For example, in [25] the authors describe and extend the concept of "F_a fairness". However, we chose to use the definition of Chiu and Jain which is the one most quoted in the networking literature concerning a simple model of a single bottleneck.

The intra-protocol fairness is the fairness between two flows of the same protocol. Each flow is sent from a different sending host to a different receiving host at the same time.

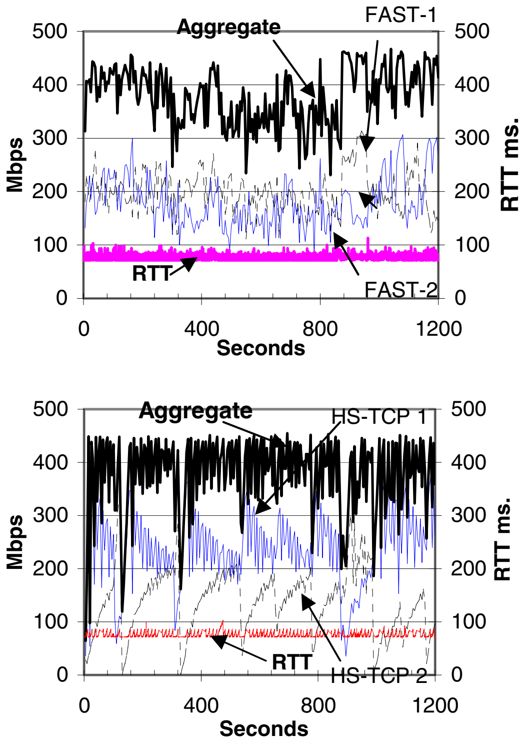

Figure 5: Comparison of Intra-protocol fairness measurements from our site to UFL

Table 2 shows the Intra-protocol friendliness measured from our site to Caltech, UFL and Manchester for the 3 different window sizes. Also shown are the averages and standard deviations.

Dest	Window	P-TCP	S-TCP	Fast	HS-TCP	Bic-TCP	H-TCP	HS TCP-LP
Calt	256K	0.99	1.00	1.00	1.00	1.00	1.00	1.00
Calt	512K	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Calt	1MB	1.00	1.00	1.00	1.00	0.99	1.00	1.00
U. Fl.	1MB	1.00	1.00	1.00	1.00	1.00	1.00	1.00
U. Fl.	4MB	1.00	1.00	1.00	0.99	1.00	1.00	1.00
U. Fl.	8MB	1.00	1.00	1.00	0.94	1.00	1.00	1.00
Man	4MB	1.00	1.00	1.00	1.00	0.98	0.98	0.99
Man	8MB	0.98	1.00	0.97	1.00	1.00	0.90	0.96
Man	12MB	0.92	0.97	1.00	0.97	0.97	0.79	0.86
	Avg	0.99	0.99	1.00	0.99	0.99	0.96	0.98
	Std	0.03	0.01	0.01	0.02	0.01	0.07	0.05

Table 2: Intra-protocol Fairness

In general, all the protocols have a good intra-fairness (83% of the measurements had $F \geq 0.98$). Poorer fairness was observed for larger distances and to a lesser extent for larger windows. Figure 5 shows examples of Intra-protocol measurements between our site and UFL for FAST vs. FAST ($F \sim 0.99$) and HS-TCP vs. HS-TCP ($F \sim 0.94$) from our site to UFL with window sizes of 8 MBytes. The two time series (one with a solid line, the other with a dotted line) in the middle for each plot are the individual throughputs for the two HS-TCP (lower plot) and FAST (upper plot) protocols. We observe that in this example the two HS-TCP flows will switch with one another instead of maintaining a constant share of the bandwidth. The first flow will decrease after a certain time and leave the available bandwidth to the second flow. As a result, we observe a large instability in these HS-TCP flows. This effect was present but less noticeable on the Manchester path for window sizes 2 and 3. We did not notice this HS-TCP behavior on the short distance path or window size 1.

5.2 Inter-protocol fairness

For the inter-protocol fairness we sent two different flows on the link from two different machines. The aim of this experiment was to see how each protocol behaves with a competing protocol. We hoped that the protocol would neither be too aggressive nor too gentle (non-aggressive) towards the other protocols. The fairness computation described earlier does not tell us how aggressive or gentle the protocol is, only that it is not taking/getting a fair share of the achievable throughput. Hence we introduce the following formula, which defines the asymmetry between two throughputs:

$$A = \frac{\bar{x}_1 - \bar{x}_2}{\bar{x}_1 + \bar{x}_2}$$

where x_1 and x_2 are the throughput averages of streams 1 and 2 in the cross-traffic.

Table 3 shows the asymmetries of the cross-traffic between different stacks. A value near one indicates that the protocol is too aggressive towards the competing protocol. A value near minus one indicates a too gentle protocol. The optimal is to have a value near 0 that indicates that the protocol is fair against the other protocols.

	P-TCP	S-TCP	Fast	HS-TCP	Bic-TCP	H-TCP	HS TCP-LP
Caltech	0.16	0.24	-0.1	-0.28	0.01	-0.02	-0.47
UFL	0.78	0	-0.01	-0.06	0.15	-0.12	0
Manchester	0.19	-0.08	0.04	-0.38	-0.03	0.25	-0.56
Avg	0.37	0.05	-0.02	-0.24	0.04	0.04	-0.34

Table 3: Average asymmetry of each protocol vs. all others

Our results show that Bic-TCP, Fast TCP, S-TCP and H-TCP have small absolute values of the fairness asymmetry. It is normal for HSTCP-LP to be too gentle (and have a large negative value of the asymmetry) since it uses only the remaining bandwidth and is deliberately non-intrusive - thus we removed it from our calculation of the average asymmetry of the other protocols for the middle-distance and long-distance. On the short-distance, we can see that all advanced TCP stacks other than P-TCP compete like a single stream of Reno but since P-TCP is very aggressive (as expected), we do not include it in the average asymmetry of the other protocols for the short-distance. Only Bic-TCP is sufficiently aggressive to compete with P-TCP in this case, but it appears too aggressive for the other protocols. Our results show that S-TCP, which is very aggressive in short-distance, becomes quite gentle in the long-distance. On the other hand, H-TCP, which is gentle in the short and middle distances, becomes aggressive in long-distance. HS-TCP, as expected, is too gentle in our tests.

5.3 Reverse-traffic

Reverse-traffic causes queuing on the reverse path. This in turn can result in the ACKs being lost or coming back in bursts (compressed ACKs [30]). Normally, the router, the path and the Ethernet card are full-duplex and should not be affected by the reverse-traffic but in actuality the reverse-traffic affects the forward traffic implicitly by modifying the ACK behavior. Therefore, we tested the protocols by sending TCP traffic from our site to UFL using an advanced stack and from UFL to our site using P-TCP with 16 streams. Table 4 shows the results of the throughputs in Mbps measured with 8 MByte windows where the first 10 minutes of the measurement had the reverse traffic and the remaining 10 minutes had no reverse traffic. Typical standard deviations are about 10-20% of the average throughputs. It is seen that Fast TCP - which is based on TCP Vegas that uses RTT for congestion detection - is more heavily affected by heavy reverse-traffic that affects (usually increases) the reverse path delays and hence the RTTs. The net effect is that, for the tested version of Fast TCP, throughput is typically about 4 times less than the other stacks, apart from HS-TCP. HS-TCP never reaches the limit at which the AIMD behavior changes from Reno to HS.

	Bic-TCP	Fast	HS-TCP	HSTCP-LP	H-TCP	P-TCP	S-TCP
With rev. traffic	230 ± 40	20 ± 10	110 ± 50	220 ± 60	220 ± 40	200 ± 60	280 ± 50
Without rev. traffic	400 ± 40	260 ± 50	380 ± 50	380 ± 30	380 ± 40	380 ± 60	400 ± 20

Table 4: `Iperf` TCP throughputs in Mbps from our site to UFL with and without reverse traffic

6 10Gbps path tests

During SuperComputing 2003[7], we made some tentative TCP performance measurements on 10 Gbps links between hosts at our booth at the Phoenix convention center and a host at the Palo Alto Internet eXchange (PAIX), a host at StarLight in Chicago and a host at NIKHEF in Amsterdam. Due to the limited amount of time we had access to these links (<3 days) and the emphasis on demonstrating the maximum throughput for the SC03 Bandwidth Challenge, these measurement are necessarily incomplete, however some of the results are felt to be worth reporting.

6.1 Setup

All the hosts at Phoenix and PAIX were Dell 2650s with dual Xeon CPUs, a 533 MHz front side bus, and an Intel PRO/10GbE LR Network Interface Card (NIC) plugged into the 133 MHz 64 bit PCI-X bus slot. There were 3 hosts at our booth at SC03, two with 3.06 GHz CPUs, and the third with 2.04 GHz CPUs. The host at PAIX had dual Xeon 3.06 GHz CPUs and a 10 Gbps Ethernet connection (LAN PHY at level 2) to a Cisco GSR router in LA. From the GSR router the signal was transmitted via an OC192/POS circuit to a Juniper 640 router managed by SCInet at the Phoenix Convention Center. From the Juniper the path went via a Force 10 E1200 router to the Cisco 6509 in our booth using a 10 Gbps Ethernet link.

The StarLight/Chicago path from the booth went from the Cisco 6509 via a second 10 Gbps Ethernet link to the Force 10 router and on through a second Juniper router connected to the Abilene core at 10 Gbps, and thence via Abilene routers at Los Angeles, Sunnyvale, Kansas City, and Indianapolis to Chicago. The host at StarLight was an HP Integrity rx2600 (64 bit Itanium) system with dual 1.5 GHz CPUs and 4 GByte RAM.

The Amsterdam path followed the same path to StarLight and then had one extra hop over the SURFnet 10 Gbps link to Amsterdam. The host at Amsterdam was an HP Itanium/F10 at NIKHEF.

[7] SC2003: http://www.sc-conference.org/sc2003/

6.2 Methodology

We set up the sending hosts at SC03 with the Caltech Fast TCP stack, and the DataTAG altAIMD stack [28]. The latter allowed dynamic (without reboot) selection of the standard Linux TCP stack (New Reno with Fast re-transmit), the Manchester University implementation of the High Speed TCP (HS-TCP) and the Cambridge University Scalable TCP stack (S-TCP). By default we set the Maximum Transfer Unit (MTU) to 9000 Bytes and the transmit queue length (txqueuelen) to 2000 packets.

We started the first set of measurements at the same time as our bandwidth challenge demonstration (about 16:00 PST Wednesday 19th November 2003). The main emphasis at this time was to achieve the maximum throughput; the evaluation of different TCP stacks was a secondary goal. The duration of the tests was about 60 minutes.

We started the second set of measurements just before midnight on Wednesday 19th November. These measurements were between Phoenix and PAIX, Phoenix and Chicago (65 ms minimum RTT) and Phoenix and Amsterdam (175 ms minimum RTT). This was a reasonably controlled set of measurements with no Bandwidth Challenge in progress and little cross-traffic. Each test was for 1200 seconds, with a given stack, and fixed maximum window size. We finished the second set of tests at about 07:00 PST Thursday 20th November.

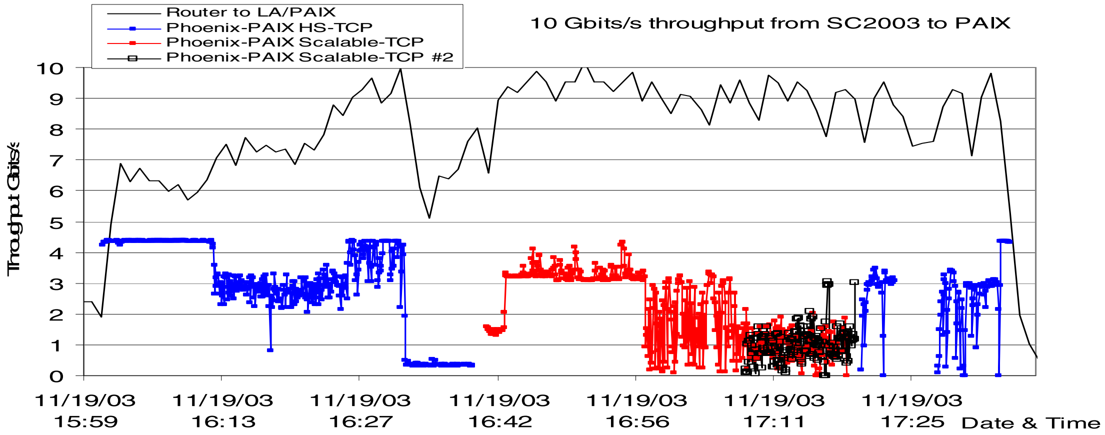

Figure 6: Points= TCP throughput from our booth to PAIX.
Smooth Curve= total SC2003 traffic on the link to LA, taken from the Juniper router.

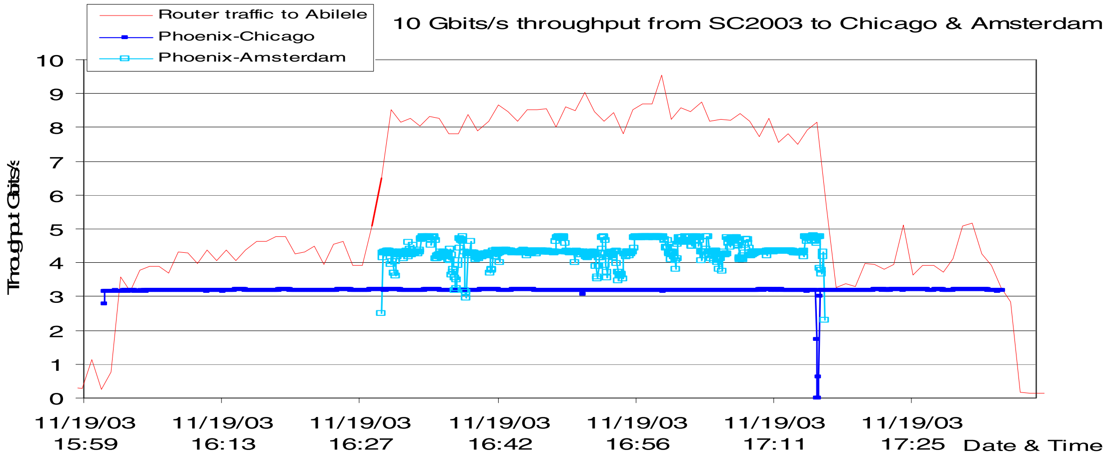

Figure 7: Points= TCP throughput from our booth to Amsterdam and Chicago.
Smooth Curve= total SC2003 traffic on the Abilene access link, taken from the Juniper router.

6.2.1 Tests made during the Bandwidth Challenge

During the Bandwidth Challenge, TCP flows were set up from our booth to PAIX, Chicago and

Amsterdam. They were generated using iperf with an MTU of 9000 Bytes and TCP window sizes of 30, 60 and 200 MBytes respectively. Separate Dell machines at the booth, running the DataTAG altAIMD stack, were used for the flow to each remote location. Following other work on PCI-X transactions [27], the PCI-X parameter "maximum memory read Byte count", mmrbc, was set to 4096 bytes on both local are remote machines. Setting mmrbc to the maximum 4096 bytes minimized the time taken for the packets to cross the PCI-X bus and thus increased throughput.

The flow to PAIX shared the SC2003-LA link with traffic from the Caltech booth to nodes at LA. The flows to Amsterdam and Chicago shared the 10 Gbps Ethernet link to the SCInet Force10 switch and then shared the Abilene access link with other ~ 1-2Gigabit traffic Bandwidth Challenge flows to US sites.

The points in Figure 6 show the TCP user level throughput (goodput) to the remote node in PAIX and the solid line shows the total traffic from SC2003 on the link to LA, taken from the link level counters on the Juniper router. The first transfer (16:00 – 16:40) used HS-TCP and initially the observed throughput was 4.37 Gbps and extremely stable, giving a Stability Index (standard deviation / average throughput) of 0.005. After 16:12 the throughput dropped to 2.87 Gbps with considerable variation, giving a Stability Index of 0.16. This could be due to the increase of the total traffic on the link (~7.5 Gbps upwards). The dramatic drop at 16:32 coincides with the link reaching its 10 Gbps capacity. The red points from 16:42 to 17:18 show the throughput using scalable TCP, S-TCP. Initially the throughput was 3.3 Gbps with a Stability Index of 0.08, similar or slightly better than HS-TCP given the load on the LA link. Between 17:08 and 17:18 a second S-TCP flow was started between the same end hosts. The average of each flow drops to ~1.0 Gbps and the Stability Index increases to 0.56. The sum of the two S-TCP flows was ~1.9 Gbps with a Stability Index of 0.39.The combined rate was less than that for 1 flow which could be due to the extra processing required for two flows.

In comparison, Figure 7 shows the TCP user level throughput to Chicago is quite steady at 3.1 Gbps with a Stability Index of 0.016, while to Amsterdam the throughput is greater at ~4.35 Gbps but less stable with a Stability Index of 0.069.The solid line in Figure 7 shows the total link level traffic from SC2003 on the Abilene access link, which is about 1 Gbps more than the total of the Amsterdam and Chicago traffic. It is worth noting that the host providing the flow to Chicago had 2.04 GHz CPUs, while that to Amsterdam had 3.06 GHz CPUs, this may account for the lower throughput recorded.

6.2.2 Tests to PAIX

On the Phoenix to PAIX link (minimum RTT 17 ms) we used maximum window sizes of 8 MBytes, 16MBytes and 32 MBytes[8]. This bracketed the nominal optimum window size calculated from the BDP of 17 ms * 10 Gbps ~ 20 MBytes. For the PAIX link, all the tests were made with a single TCP stream. For Reno, HS-TCP and S-TCP there was little observable differences between the four stacks in the achievable bandwidth behavior.

- For an MTU of 9000 Bytes, a window size of 8MBytes was inadequate and resulted in throughput being limited to about 2.9 Gbps for Reno single stream, HS-TCP and S-TCP
- For an MTU of 9000 Bytes with window sizes of 16 MBytes or 32 MBytes for Reno single stream, HS-TCP and S-TCP the throughput increased to about 4.3Gbps.
- The throughputs with an MTU of 9000 Bytes were very stable. Typical values of the Stability Index were < 0.004. The larger values for Reno with a single stream and an 8 MByte window and HS-TCP with a 16 MByte window were each caused by a single sudden drop of throughput for one 5 second periods.
- Reducing the MTU from 9000 Bytes to 1500 Bytes reduced the throughput for a 16 MByte window from 4.3 Gbps to about 700 Mbps, and for an 8 MByte window from 2.9 Gbps to ~ 360 Mbps. Also the throughputs were less stable with the 1500 Byte MTU (Stability Index > 0.10).
- With an MTU of only 1500 Bytes, Fast TCP gave similar performance to HS-TCP and STCP when they ran with 1500Byte MTUs.

The 4.3 Gbps limit was slightly less than the ~ 5.0 Gbps achieved with UDP transfers in our lab between back to back 3.06 GHz Dell PowerEdge 2650 hosts. On the other hand it is less than that calculated from the expected data transfer rate for a 10GE NIC with a 64 bit 133 MHz PCI-X bus [27]. The limitation in throughput is believed to be due to CPU factors (CPU speed, memory/bus speed or the

[8] More explanations are available on: http: //www-iepm.slac.stanford.edu/monitoring/bulk/sc2003/hiperf.html and http://www-iepm.slac.stanford.edu/monitoring/bulk/sc2003/stack-cf.html

I/O chipset). The relative decrease in throughput going from 9000 Byte MTU to a 1500 Byte MTU was roughly proportional to the reduction in MTU size. This maybe related to the extra CPU power / memory bandwidth required to process the 6 times as many, but 6 times as small MTUs. Back to back UDP transfers in our lab between 3.06 GHz Dell PowerEdge 2650 hosts achieved about 1.5 Gbps or about twice the 700 Mbps achieved with the SC03 long distance TCP transfers. Further work is required to understand this discrepancy.

7 Future experiments

In the near future, we plan on repeating the tests on higher speed networks, in particular on the emerging 10 Gbps test beds. We also plan to test other promising TCP stacks such as Westwood+ [20], and rate based protocols such as RBUDP [31], SABUL/UDT, and compare their performances with the TCP based protocols. Also we are planning to work with others to compare our real network results with those from simulators such as ns-2[9] or emulators such as Dummynet [23].

In the future, we would like to test a similar topology as described in [12] where the authors indicate that it may be beneficial for long RTT connections to become slightly more aggressive during the additive increase phase of congestion avoidance. In this paper we only made cross-traffic tests with two protocols having the same RTT. It means that all the senders' servers were at the same place. It was the same for the receiver. Thus, we have to check how each protocol behaves with different RTTs on the same link. The increase between the different protocols on the path will be different and it may affect the fairness.

We should also test the different protocols with more than one stream to see how aggressive or gentle a protocol is on this case. Finally, we plan to test other promising TCP stacks and rate based protocols and compare their performance with the TCP based protocols.

8 Conclusion

In this paper we presented the results of a two-month experiment to measure the performance of 7 TCP stacks from our site over various network paths. If we compare the various TCP stacks for the more important metrics (throughput achievable, impact on RTT, aggressiveness, stability and convergence) we observe for the set of measurements:

- The differences in the performances of the TCP stacks are more noticeable for the longer distances.
- TCP Reno single stream, as expected, is low performance and unstable on longer distances.
- P-TCP is too aggressive. It is also very unfair with the RTT on short distance.
- HSTCP-LP is too gentle and, by design, backs-off too quickly otherwise it performs well. It looks very promising to use to get Less than Best Effort (LBE) service without requiring network modifications.
- Fast TCP performs as well as most others but it is very handicapped by the reverse traffic.
- S-TCP is very aggressive on middle-distance and becomes unstable with UDP traffic on long distance but achieves high throughput.
- HS-TCP is very gentle and has some strange intra-fairness behavior.
- Bic-TCP overall performs very well in our tests.

It is also very important to choose a TCP stack that works well with and will not decrease the performance and efficiency of TCP Reno used all around the world. Moreover, we will always prefer an advanced TCP which has the recommendation of the IETF and which will be used by everybody.

9 Acknowledgements

Section removed as part of the double-blind review process.

10 References

[1] Globus Alliance. Available online: http://www.globus.org/datagrid/gridftp.html.

[2] D. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. In *Computer Networks and ISDN Systems*, pages 1–14, June 1989.

[3] T. Dunigan. http://www.csm.ornl.gov/~dunigan/net100/.

[4] G. Farrache. Available online: http://doc.in2p3.fr/bbftp/.

[5] W. Feng, M. Fisk, M. Gardner, and E. Weigle. Dynamic right-sizing: An automated, lightweight, and scalable technique for enhancing grid performance. In *7th IFIP/IEEE International Workshop, PfHSN 2002*, Berlin, April 2002.

[9] "The Network Simulator - ns-2", available at http://www.isi.edu/nsnam/ns/

[6] W. Feng, J. Hurwitz, H. Newman, S. Ravot, R. L. Cottrell, O. Martin, F. Coccetti, C. Jin, X. Wei, and S. Low. Optimizing 10-gigabit Ethernet for networks of workstations, clusters and grids: A case study. In *Supercomputing Conference 2003*, Phoenix, November 2003.

[7] S. Floyd. Limited slow-start for TCP with large congestion windows. IETF Internet Draft, draft-floyd-tcp-slowstart-01.txt, August 2002.

[8] S. Floyd. HighSpeed TCP for large congestion windows. IETF Internet Draft, draft-floyd-highspeed-02.txt, February 2003.

[9] Y. Gu, X. Hong, M. Mazzuci, and R. L. Grossman. SABUL: A high performance data transport protocol. *IEEE Communications Letters*, 2002.

[10] T. Hacker, B. Noble, and B. Athey. Improving throughput and maintaining fairness using parallel TCP. *Submitted to IEEE INFOCOM 2004*, Hong Kong, 2004.

[11] A. Hanushevsky, A. Trunov, and R. L. Cottrell. Peer-to-peer computing for secure high performance data copying. In *Computing in High Energy Physics*, Beijing, 2001.

[12] T. H. Henderson, E. Sahouria, S. McCanne, and R. H. Katz. On improving the fairness of TCP congestion avoidance. *IEEE Globecomm conference*, 1998.

[13] C. Jin, D. Wei, S. H. Low, G. Bushmaster, J. Bunn, D. H. Choe, R. L. A Cottrell, J. C. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, and S. Singh. FAST TCP: From theory to experiments. In *First International Workshop on Protocols for Fast Long-Distance Networks (PFLDNet 2003)*, Geneva, February 2003.

[14] C. Jin, D. X. Wei, and S. H. Low. FAST TCP: Motivation, architecture, algorithms, performance. In *IEEE INFOCOM 2004*, Hong Kong, March 2004.

[15] D. Katabi, M. Handley, and C. Rohrs. Internet congestion control for high bandwidth delay product network. In *ACM SIGCOMM*, Pittsburgh, August 2002.

[16] T. Kelly. Scalable TCP: Improving performance in highspeed wide area networks. *Submitted for publication*, December 2002.

[17] A. Kuzmanovic and E. W. Knightly. TCP-LP: A distributed algorithm for low priority data transfer. In *IEEE INFOCOM*, San Francisco, April 2003.

[18] Y. Li. URL: http://www.hep.ucl.ac.uk/~ytl/tcpip/hstcp/.

[19] Y. Li. URL: http://www.hep.ucl.ac.uk/~ytl/tcpip/linux/txqueuelen/.

[20] L. A. Grieco and S. Mascolo, Performance evaluation of Westwood+ TCP over WLANs with Local Error Control, *28th Annual IEEE Conference on Local Computer Networks (LCN 2003).*

[21] Indiana University Advanced Network Management Lab Tsunami Project. Available

http://www.indiana.edu/~anml/anmlresearch.html.

[23] Luigi Rizzo. Dummynet: A simple approach to the evaluation of network protocols. *ACM Computer Communications Review*, 27(1):31–41, 1997.

[24] R. Shorten, D. Leith, J. Foy, and R. Kildu, Analysis and design of congestion control in synchronised communication networks, 2003.

[25] M. Vojnovic, J-Y Le Boudec, and C. Boutremans. Global fairness of additive-increase and multiplicative-decrease with heterogeneous round-trip times. In *Proceedings of IEEE INFOCOM'2000*, pages 1303–1312, TelAviv, Israel, March 2000.

[26] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion Control (BIC) for Fast, Long-Distance Networks. To appear in *Infocom 2004, Hong Kong, March 2004*

[27] R.Hughes-Jones, P. Clarke, S. Dallison and G. Fairey, Performance of Gigabit and 10 Gigabit

Ethernet NICs with Server Quality Motherboards Submitted for publication in H*igh-Speed Networks and Services for Data-Intensive Grids, Special issue of Future Generation Computer Systems (FGCS),* 2003

[28] Available at: http://www.hep.uvl.ac.uk/~ytl/tcpip/linux/altaimd/

[29] Lawrence S. Brakmo, Sean W. O'Malley and Larry L. Peterson,TCP Vegas: New Techniques for Congestion Detection and Avoidance, *SIGCOMM 1994*

[30] L. Zhang, S. Shenker, and D. D. Clark, Observations and dynamics of a congestion control algorithm: the effects of two-way traffic, *Proc. ACM SIGCOMM '91, pages 133-147, 1991. 17*

[31] E. He and J. Leigh, Reliable Blast UDP. Available http://www.evl.uic.edu/eric/atp/RBUDP.doc