

Experimental Evaluation of TCP Protocols for High-Speed Networks

Yee-Ting Li, Douglas Leith, and Robert N. Shorten

Abstract—In this paper, we present experimental results evaluating the performance of the scalable-TCP, HS-TCP, BIC-TCP, FAST-TCP, and H-TCP proposals in a series of benchmark tests. In summary, we find that both Scalable-TCP and FAST-TCP consistently exhibit substantial unfairness, even when competing flows share identical network path characteristics. Scalable-TCP, HS-TCP, FAST-TCP, and BIC-TCP all exhibit much greater RTT unfairness than does standard TCP, to the extent that long RTT flows may be completely starved of bandwidth. Scalable-TCP, HS-TCP, and BIC-TCP all exhibit slow convergence and sustained unfairness following changes in network conditions such as the start-up of a new flow. FAST-TCP exhibits complex convergence behavior.

Index Terms—Evaluation of TCP protocols, high-speed networks, TCP congestion control.

I. INTRODUCTION

THE TCP congestion control algorithm has been remarkably successful in making the current Internet function efficiently. However, in recent years, it has become clear that it can perform very poorly in networks with high bandwidth-delay product (BDP) paths. The problem stems from the fact that the standard TCP AIMD congestion control algorithm increases the congestion window $cwnd$ too slowly. This is illustrated in Fig. 1, which plots evolution of the congestion window $cwnd$ of a single flow, and its throughput time histories measured on a 1-Gb/s transatlantic path between Dublin, Ireland, and Chicago, IL. The propagation delay is 100 ms, and the bandwidth-delay product approximately 8000 packets. On reducing $cwnd$ by a half, when delayed acking is used, it takes 8000 round-trip times i.e., 800s for the $cwnd$ to fill the pipe again. This is simply too slow for most applications as it would lead to prohibitively long file transfer times. In the example shown, it takes over 1200s for the flow to recover after a backoff, and the average throughput achieved is only 218 Mb/s. This poor utilization of network capacity is not confined to long-distance intercontinental paths. With the continuing rollout of gigabit-speed (and faster) links, latencies of only a few tens of milliseconds are quite sufficient to create bandwidth-delay products that yield poor throughput performance with the current TCP congestion control algorithm.

A solution to this problem that has been pursued by many authors is to increase the rate at which $cwnd$ is increased and thereby shorten the congestion epoch duration. However, backward compatibility requirements with existing TCP flows requires that any new protocol should behave similarly to stan-

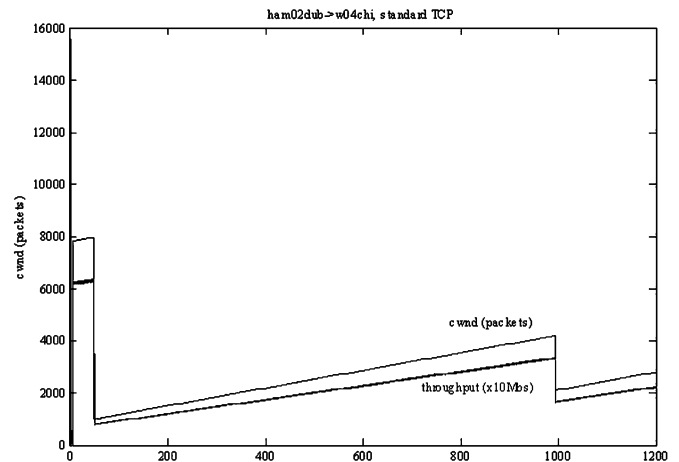


Fig. 1. Measured $cwnd$ and throughput time histories on 1-Gb/s path between Dublin, Ireland, and Chicago, IL. Over 1200s, the average throughput achieved is only 218 Mb/s. These particular measurements were taken on the afternoon of December 9, 2003 using a dedicated transatlantic link with no significant competing traffic.

dard TCP on paths with low bandwidth-delay product. Early work along these lines includes the HS-TCP proposal of Floyd [8], the scalable-TCP proposal of Kelly [12] and the FAST-TCP proposal of Low *et al.* [9]; more recent new proposals include BIC-TCP [20] and H-TCP [14]. These proposals have all been the subject of considerable interest and experimentation in recent years.

Due in no small part to the volume of work that has been carried out in this area, a real need has developed for systematic screening of proposals to identify suitable candidates for more detailed evaluation. Evaluating the performance of new TCP proposals is not easy. One principal difficulty arises from the lack of an agreed-on set of performance measures. As a result of the latter, different studies typically employ performance tests that highlight particular aspects of TCP performance while casting little light on other, equally important, properties of proposed protocols. Several existing studies also do not control for variations in performance associated with differences in network stack implementation that are unrelated to the congestion control algorithm (see below). This is an important practical aspect that is frequently ignored in academic studies on the topic. In view of these facts, it is not surprising that concrete conclusions relating to the merits of competing proposals have been difficult to make based on currently available published results.

Our aim in this paper is to compare the performance of competing TCP proposals in a systematic and repeatable manner. It is important to emphasize that our goal in this paper is not to achieve exhaustive testing, but rather to perform initial

Manuscript received May 26, 2005; revised August 18, 2006; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor R. Srikant.

The authors are with the Hamilton Institute, National University of Ireland, Maynooth, Kildare, Ireland (e-mail: doug.leith@nuim.ie).

Digital Object Identifier 10.1109/TNET.2007.896240

screening of proposals. Our approach is to define and use a set of benchmark tests that probe a number of important aspects of new protocols and to consistently apply these tests to all proposals. Specifically, we present experimental measurements of the performance of the HS-TCP, scalable-TCP, FAST-TCP, BIC-TCP, and H-TCP¹ proposals. These tests highlight a number of specific deficiencies of the protocols studied and suggest future research directions to render these suitable for deployment in real networks.

II. SOME PITFALLS

Comparing the performance of TCP proposals is not always easy and many pitfalls exist. Examples include the following.

Different Network Stack Implementations: In several recent studies on high-speed networks, publicly available Linux patches provided by the authors of TCP proposals are used. The performance of these patches are then compared directly. However, patches may relate to different operating system versions. More seriously, performance issues relating to the inefficiency of the network stack implementation, particularly in relation to SACK processing, are known to have a significant impact on performance. As a result, most patches implementing proposed changes to the TCP congestion control algorithm also implement numerous changes to the network stack that are unrelated to the congestion control algorithm. Consequently, direct performance comparisons of these patches risk revealing more about the efficiency of the network stack implementation than about the performance of the congestion control algorithm. In this paper, we use a common network stack implementation with all of the congestion control algorithms studied in order to focus solely on the latter's performance.

Congestion Control Action Not Exercised: It is important to design experiments that exercise the TCP congestion control algorithm rather than other elements of the network stack. For example, it is essential that the bandwidth of the network is lower than that of the server network interface card (NIC), i.e., that the network bottleneck lies external to the server being tested. Otherwise, it is often the case that the transport layer congestion control algorithm is effectively inactive (packet drops are virtual), and performance measurements merely evaluate the efficiency of the NIC driver.

Performance Measures Too Narrow: We argue that it is not sufficient to focus solely on TCP throughput performance. Fairness, responsiveness, backward compatibility, support for incremental rollout, etc., should also be evaluated.

Range of Network Conditions: Frequently, results are presented tests from a single test run only and/or for a specific network condition or small range of network conditions. A huge variety of conditions exist in modern networks. We argue that it is essential, as a minimum, to characterize TCP performance across a broad range of bandwidths (not just on high-speed links), propagation delays (not just transcontinental links), router buffer sizes (not just very large or very small buffers), and mix of connection sizes.

¹We note that H-TCP is developed by some of the authors of this paper. We emphasize, therefore, that all of the protocols studied are put through identical tests yielding quantitative and repeatable measurements. While space restrictions prevent us from including all of our experimental measurements in this paper, the measurements are available at www.hamilton.ie/net/eval.

Such issues limit the utility of previous evaluation studies and motivate the approach taken in the present paper. We do not claim that our tests in this paper are exhaustive. We do, however, seek to demonstrate their utility and discriminating power and to initiate wider debate on this topic in the networking community.

III. COMPARATIVE TESTING

An immediate difficulty that arises in our work, even for the limited scenarios that we consider, is that the question as to what exactly constitutes a good network protocol is itself a topic of much debate. We do not attempt to answer this question here. Instead, we seek to support decision making by characterizing some important aspects of the behavior of new protocols in a consistent and objective manner. While we lack agreed-on metrics for ranking performance, we do have the existing TCP standards-based algorithm against which to compare the performance of new protocols. We therefore propose taking the performance of the current start-of-the-art TCP algorithm² as a baseline against which the behavior of new proposals can be compared.

It is also important to emphasize that our goal in this paper is not to achieve exhaustive testing, but rather to perform initial screening of proposals. We therefore seek to define a series of benchmark tests that can be consistently applied and that exercise core functionality of TCP. The performance problems of standard TCP over high bandwidth-delay product paths are largely associated with bulk data transfers. It is therefore natural to take this as our starting point in testing new TCP proposals. In addition to focusing our attention on the performance of long-lived flows, we also confine consideration to drop-tail queues, since this is the prevalent queueing discipline in current networks and to a single shared bottleneck link.

We recognize that short-lived TCP flows, and indeed non-TCP flows, constitute a large proportion of traffic in real networks. Similarly, not all routers operate drop-tail queueing disciplines, and multiple bottlenecks including cross traffic can occur. However, as a minimum, we expect that TCP algorithms should function well over a single bottleneck link with drop-tail queueing and, as we shall see, the range of network conditions that we consider is already sufficient to highlight many interesting features of new TCP proposals. Moreover, a single bottleneck link with drop-tail queueing is an obvious starting point for investigating new algorithms as the behavior of the standard TCP algorithm in this setting is well studied. Indeed, our understanding of standard TCP behavior under these conditions immediately suggests a number of fundamental characteristics to consider in making comparisons.

A. Definitions

Before proceeding, the following definitions will be useful. Letting $U_i(t)$ denote the number of packets transferred by the i th flow in the time interval $[0, t]$, the *average throughput* is

$$\bar{u}_i := \lim_{T \rightarrow \infty} \frac{U_i(T)}{T}. \quad (1)$$

²Implementations of standard TCP do differ in their behavior. However, differences in implementation are largely confined to areas such as time-out handling, undo actions, etc., and there is generally consistency in the implementation of the congestion control algorithm itself. In this paper, we consider the Linux 2.6 TCP implementation.

We also define the short-term average throughput as the **moving average**

$$\hat{u}_i(t + \delta) = (1 - \lambda)\hat{u}_i(t) + \lambda \frac{U(t + \delta) - U(t)}{\delta}. \quad (2)$$

We have that $U(t + \delta) - U(t)$ is the number of packets sent in interval $[t, t + \delta]$ and $(U(t + \delta) - U(t))/\delta$ is the average sending rate over this interval. In this paper, we use a sampling interval δ of 0.1 s. We have fading memory so that $\hat{u}_i(t)$ is, roughly speaking, the running average over a window of past data with the window size determined by the parameter λ . We choose λ so that the averaging window scales is approximately 100 round-trip times. We define the ε -convergence time following startup of a new flow to be the time before the short-term average throughput of the new flow is within a factor ε of its long-term average value. Typically, we use $\varepsilon = 0.8$ yielding the 80% convergence time.

B. Range of Network Conditions

We consider round-trip propagation delays in the range 16–320 ms and bandwidths ranging from 1–250 Mb/s. We do not consider these values to be definitive—the upper value of bandwidth considered can, in particular, be expected to be subject to upwards pressure. We do, however, argue that these values are sufficient to capture an interesting range of network conditions that characterizes current communication networks. In all of our tests, we consider delay values of 16, 40, 80, 160, and 320 ms and bandwidths of 1, 10, 100, and 250 Mb/s. In addition, we perform each test with 0, 10, 20, 30, 40, and 50 competing bidirectional Web sessions. This defines a three-dimensional grid of measurement points where, for each value of delay and level of Web traffic, performance is measured for each of the values of bandwidth.

C. Fairness

The formal fairness requirement on new protocols is unclear, and many definitions of fairness exist. Nevertheless, we can make the following observations. On a path with a single bottleneck, we expect that competing long-lived flows with the same round-trip time should achieve approximately the same average throughput. Flows with different round-trip times will be unfair when the standard TCP congestion control algorithm is used, with short round-trip time flows generally achieving greater average throughput than long round-trip time flows (e.g., see [17]). We therefore require that our tests of new TCP proposals should, as a minimum, evaluate the impact of round-trip time on the relative throughputs of competing flows. Specifically, to evaluate fairness, we consider two TCP flows and propose the following tests.

- i) **Fairness with same RTT:** Measure the average throughput of each flow when each flow operates the same congestion control algorithm, has the same propagation delay, and has a shared bottleneck link. Measurements are taken for a range of propagation delays, link bandwidths, and level of competing bidirectional Web traffic (see above), and the queue is sized as a constant proportion of the bandwidth-delay product (we suggest 20% and 100% of the bandwidth-delay product, roughly corresponding to conditions with small and large queues).
- ii) **Fairness with different RTTs:** Measure the average throughputs as the propagation delay of the first flow is

held constant and that of the second flow is varied from 16–320 ms. Measurements are taken for a range of link bandwidths, Web traffic, and propagation delays of the first flow; the queue is sized as a constant proportion of the bandwidth-delay product.

D. Backward Compatibility

To evaluate backward compatibility, we repeat the foregoing fairness measurements but now with the first flow operating the standard TCP algorithm and the second flow operating the new TCP congestion control algorithm being studied.

E. Efficiency

Efficiency refers to the utilization of the available network resources. It is known that the efficiency of standard TCP is influenced by the queue provisioning within the network: for a single flow (or with multiple synchronized flows) link, utilization falls as the queue size is reduced below the delay-bandwidth product of a path. As a minimum, we therefore expect our tests to characterize efficiency with respect to this parameter.

To evaluate link utilization, we consider two TCP flows having the same propagation delay and propose the following two tests.

- i) **Efficiency versus queue provisioning:** Measure average throughput and loss overhead as the queue provisioning is varied from 1% to 100% of the bandwidth-delay product.
- ii) **Efficiency versus RTT:** Measure average throughput and loss overhead as the propagation delay is varied and the queue size scaled to be a constant proportion of the bandwidth-delay product.

F. Responsiveness

Since network conditions are not static, we are also interested in the ability to rapidly acquire and release bandwidth as conditions change.

- i) **Response function:** On links with many flows, the backoff events experienced by a single flow are often modeled as a random process (e.g., see Padhye *et al.* [17]). Motivated by this, we evaluate the impact of random packet loss on efficiency via the following test. Configure the network to generate random packet losses with constant per-packet drop probability (in our tests, we implemented this on a software router). Measure the average throughput of a single TCP flow as the level of random packet losses is varied.
- ii) **Convergence time:** On links with a smaller numbers of flows, it is known that interactions between competing flows can have a strong impact on network convergence time following a disturbance (e.g., see [19]). We evaluate the responsiveness of small numbers of TCP flows to changing network conditions by measuring the 80% convergence time following the startup of a second flow. We recommend that tests be repeated with a range of start times of the second flow that span at least one congestion epoch of the first flow. In this way, we can evaluate the average performance independent of the specific start time used.

As usual, these measurements are carried out for a range of propagation delays, Web traffic, and link bandwidths.

IV. EVALUATING HIGH-SPEED PROTOCOLS

In this section, we measure the performance of the following high-speed proposals: scalable-TCP, high-speed TCP (HS-TCP), BIC-TCP, FAST TCP, and H-TCP. These proposals have all been the subject of considerable interest and experimentation in recent years, with patches implementing each of these protocols on the Linux operating system publicly available.

Before proceeding, we very briefly review the basic operation of each of these competing proposals. The reader is referred to the original literature for more detailed information.

A. Scalable-TCP

The basic idea in scalable-TCP [12] is to make the recovery time after a congestion event independent of window size. Specifically, scalable-TCP proposes that the TCP *cwnd* be updated as follows:

$$\begin{aligned} \text{Ack : } cwnd &\leftarrow cwnd + \alpha \\ \text{Loss : } cwnd &\leftarrow \beta \times cwnd. \end{aligned}$$

Suggested values for the parameters α and β are 0.01 and 0.875, respectively. A mode switch is used whereby the standard TCP *cwnd* update rules are used when *cwnd* is less than a threshold, *Low_Window*, and the scalable-TCP update rules are used for larger *cwnd* values.

B. HS-TCP

HS-TCP [8] uses the current TCP *cwnd* value as an indication of the bandwidth-delay product on a path. The AIMD increase and decrease parameters are then varied as functions of *cwnd*, as follows:

$$\begin{aligned} \text{Ack : } cwnd &\leftarrow cwnd + \frac{f_\alpha(cwnd)}{cwnd} \\ \text{Loss : } cwnd &\leftarrow g_\beta(cwnd) \times cwnd. \end{aligned}$$

In [8], logarithmic functions are proposed for $f_\alpha(cwnd)$ and $g_\beta(cwnd)$, whereby $f_\alpha(cwnd)$ increases with *cwnd* and $g_\beta(cwnd)$ decreases. Similarly to scalable-TCP, HS-TCP uses a mode switch so that the standard TCP update rules are used when *cwnd* is below a specified threshold.

C. H-TCP

H-TCP [14] uses the elapsed time Δ since the last congestion event, rather than *cwnd*, to indicate path bandwidth-delay product, and the AIMD increase parameter is varied as a function of Δ . The AIMD increase parameter is also scaled with path round-trip time to mitigate unfairness between competing flows with different round-trip times. The AIMD decrease factor is adjusted to improve link utilization based on an estimate of the queue provisioning on a path. In more detail

$$\begin{aligned} \text{Ack : } cwnd &\leftarrow cwnd + \frac{2(1-\beta)f_\alpha(\Delta)}{cwnd} \\ \text{Loss : } cwnd &\leftarrow g_\beta(B) \times cwnd \end{aligned}$$

with

$$\begin{aligned} f_\alpha(\Delta) &= \begin{cases} 1, & \Delta \leq \Delta_L \\ \max(\bar{f}_\alpha(\Delta)T_{\min}, 1), & \Delta > \Delta_L \end{cases} \\ g_\beta(B) &= \begin{cases} 0.5, & \left| \frac{B(k+1)-B(k)}{B(k)} \right| > \Delta_B \\ \min\left(\frac{T_{\min}}{T_{\max}}, 0.8\right), & \text{otherwise} \end{cases} \end{aligned}$$

where Δ_L is a specified threshold such that the standard TCP update algorithm is used while $\Delta \leq \Delta_L$. A quadratic increase function \bar{f}_α is suggested in [14], namely $\bar{f}_\alpha(\Delta) = 1 + 10(\Delta - \Delta_L) + 0.25(\Delta - \Delta_L)^2$. T_{\min} and T_{\max} are measurements of the minimum and maximum round-trip time experienced by a flow. $B(k+1)$ is a measurement of the maximum achieved throughput during the last congestion epoch.

D. BIC-TCP

BIC-TCP [20] employs a form of binary search algorithm to update *cwnd*. Briefly, a variable w_1 is maintained that holds a value halfway between the values of *cwnd* just before and just after the last loss event. The *cwnd* update rule seeks to rapidly increase *cwnd* when it is beyond a specified distance S_{\max} from w_1 , and update *cwnd* more slowly when its value is close to w_1 . Multiplicative backoff of *cwnd* is used on detecting packet loss, with a suggested backoff factor β of 0.8. In more detail

$$\begin{aligned} \text{Ack : } &\begin{cases} \delta = (w_1 - cwnd)/B \\ cwnd \leftarrow cwnd + \frac{f_\alpha(\delta, cwnd)}{cwnd} \end{cases} \\ \text{Loss : } &\begin{cases} w_1 = \begin{cases} \frac{1+\beta}{2} \times cwnd, & cwnd < w_1 \\ cwnd, & \text{otherwise} \end{cases} \\ w_2 = cwnd \\ cwnd \leftarrow \beta \times cwnd \end{cases} \end{aligned}$$

with

$$f_\alpha(\delta, cwnd) = \begin{cases} \frac{B}{\sigma}, & (\delta \leq 1, cwnd < w_1) \\ \text{or } (w_1 \leq cwnd < w_1 + B) \\ \delta, & 1 < \delta \leq S_{\max}, cwnd < w_1, \\ \frac{w_1}{B-1}, & B \leq cwnd - w_1 < S_{\max}(B-1) \\ S_{\max}, & \text{otherwise} \end{cases}$$

BIC-TCP also implements an algorithm whereby upon low utilization detection, it increases its window more aggressively. This is controlled with the *Low_Util* and *Util_Check* parameters. In order to maintain backwards compatibility, it uses the standard TCP update parameters when *cwnd* is below threshold *Low_Window*.

E. FAST-TCP

FAST-TCP [9] is a delay-based algorithm. In outline

$$\begin{aligned} \text{Each RTT : } cwnd &\leftarrow \left[cwnd + \frac{T_{\min}}{T} cwnd + f_\alpha(B) \right] / 2 \\ \text{Loss : } cwnd &\leftarrow 0.5 \times cwnd \end{aligned}$$

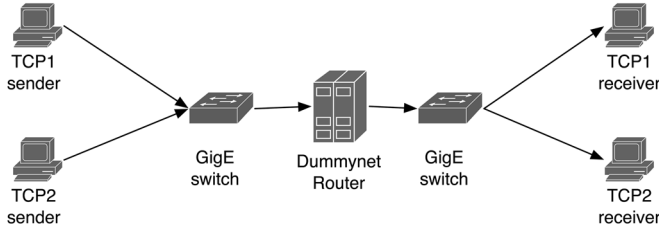


Fig. 2. Experimental setup.

TABLE I
HARDWARE AND SOFTWARE CONFIGURATION

| | Description |
|---------------------|-------------------------|
| CPU | Intel Xeon CPU 2.80GHz |
| Memory | 256 Mbytes |
| Motherboard | Dell PowerEdge 1600SC |
| Kernel | Linux 2.6.6 altAIMD-0.6 |
| txqueuelen | 1,000 |
| max_backlog | 300 |
| NIC | Intel 82540EM |
| NIC Driver | e1000 5.2.39-k2 |
| TX & RX Descriptors | 4096 |

TABLE II
DEFAULT NEW-TCP PARAMETERS USED IN ALL TESTS

| TCP Protocol | Parameters |
|--------------|---|
| HS-TCP | High_P=10 ⁻⁷ , Low_Window=31, High_Window=83,000 |
| Scalable-TCP | $\alpha = 0.01$, $\beta = 0.875$, Low_Window=16 |
| H-TCP | $\Delta^L = 1\text{sec}$, $\Delta_B = 0.2$ |
| BIC-TCP | $S_{max} = 32$, $B = 4$, $\sigma = 20$, $\beta = 0.8$ Low_Util=15%, Util_Check=2, Low_Window=14 |
| FAST-TCP | m0a=8, m1a=20, m2a=200 m0u=1500, m1l=1250, m1u=15000 and m2l=12500 |

where T_{\min} and \bar{T} are the minimum and average observed latencies of the flow, respectively. The function $f_{\alpha}(B)$ depends upon the measured throughput B achieved by the flow: currently, $f_{\alpha}(B)$ is set to 8, 20, and 200 for achieved throughputs of less than 10 Mb/s, less than 100 Mb/s, and greater than 1 Gb/s, respectively. (These thresholds are specified by the `sysctl` entries $(m0a, m0u, m1l)$, $(m1a, m1l, m1u)$ and $(m2a, m1l, m2u)$, respectively.) FAST-TCP also includes rate pacing. Note that rate pacing is a functional change and is thus viewed here as being part of the congestion control algorithm (unlike network stack issues such as efficient SACK processing implementation which fundamentally involve no functional change, only a change in computational burden).

F. Experimental Setup

All tests were conducted on an experimental testbed. Commodity high-end PCs were connected to gigabit switches to form the branches of a dumbbell topology (see Fig. 2). All sender and receiver machines used in the tests have identical hardware and software configurations as shown in Table I (see the Appendix) and are connected to the switches at 1 Gb/s. The router, running the FreeBSD dummynet software, can be configured with various bottleneck queue sizes, capacities, and round-trip propagation delays to emulate a wide range of network conditions.

Apart from the router, all machines run a modified version of the Linux 2.6.6 kernel. Each of the congestion control algorithms studied have independent patches that are publicly

available. However, these patches are often for different versions of Linux and typically also make changes to the network stack that are not directly related to the congestion control algorithm; for example, it is common for patches to alter the SACK processing algorithm to improve its efficiency as the standard implementation has known performance problems in high-speed environments [13]. To provide consistency, and control against the influence of differences in implementation as opposed to differences in the congestion control algorithm itself, we therefore built the congestion control algorithms into a common kernel. This kernel is referred to as the *altAIMD* kernel (see the Appendix for further details).³

The kernel is instrumented with the Web100 extensions [16] to allow measurement of TCP variables.

In order to minimize the effects of local hosts queues and flow interactions, unless otherwise stated, we only ran one long-lived flow per PC with flows injected into the testbed using *iperf*. Web traffic sessions are generated by dedicated client and server PCs, with exponentially distributed intervals between requests and Pareto distributed page sizes. This is implemented using a client-side script and custom CGI script running on an Apache server. Following [21], we used a mean time between requests of 1 s and a Pareto-shape parameter of 1.2. Each individual test was run at least 10 min each. In the case of tests involving Standard TCP, we ran individual tests for up to an hour as the congestion epoch duration becomes very long on large bandwidth-delay products paths. In order to obtain a good representation of the run-to-run variability in performance metrics, all individual tests were repeated at least five times and the arithmetic mean taken. An error on the measurement was taken as the standard error from this mean.

As discussed previously, an essential feature of the proposed approach is that we always carry out the full range of tests for standard TCP so as to provide a baseline against which we can evaluate the performance of new TCP proposals. By always taking measurements for standard TCP, we have a common baseline for making comparisons.

V. RESULTS

Owing to space restrictions, we cannot include the results of all our tests here. We therefore present results for a subset of network conditions that are representative of the full test results obtained.

A. Fairness With Same RTT

Fig. 3 plots the ratio of measured throughputs for two flows with the same propagation delay sharing a common bottleneck link as the path propagation delay is varied. Tests are of 10-min duration. Results are shown both for a bottleneck link bandwidth of 10 and 250 Mb/s, roughly corresponding to low and high-speed network conditions. The results shown are with no Web traffic, but similar behavior is observed when Web traffic is present.

It can be seen that this basic test reveals some striking behavior. Under these conditions, the standard TCP congestion

³We note that the implementation of BIC-TCP included in the standard Linux 2.6.6 kernel distribution is known [15] to be incorrect (this has subsequently been corrected). In our tests, we use a corrected implementation based upon the original Linux patch developed by the BIC-TCP authors.

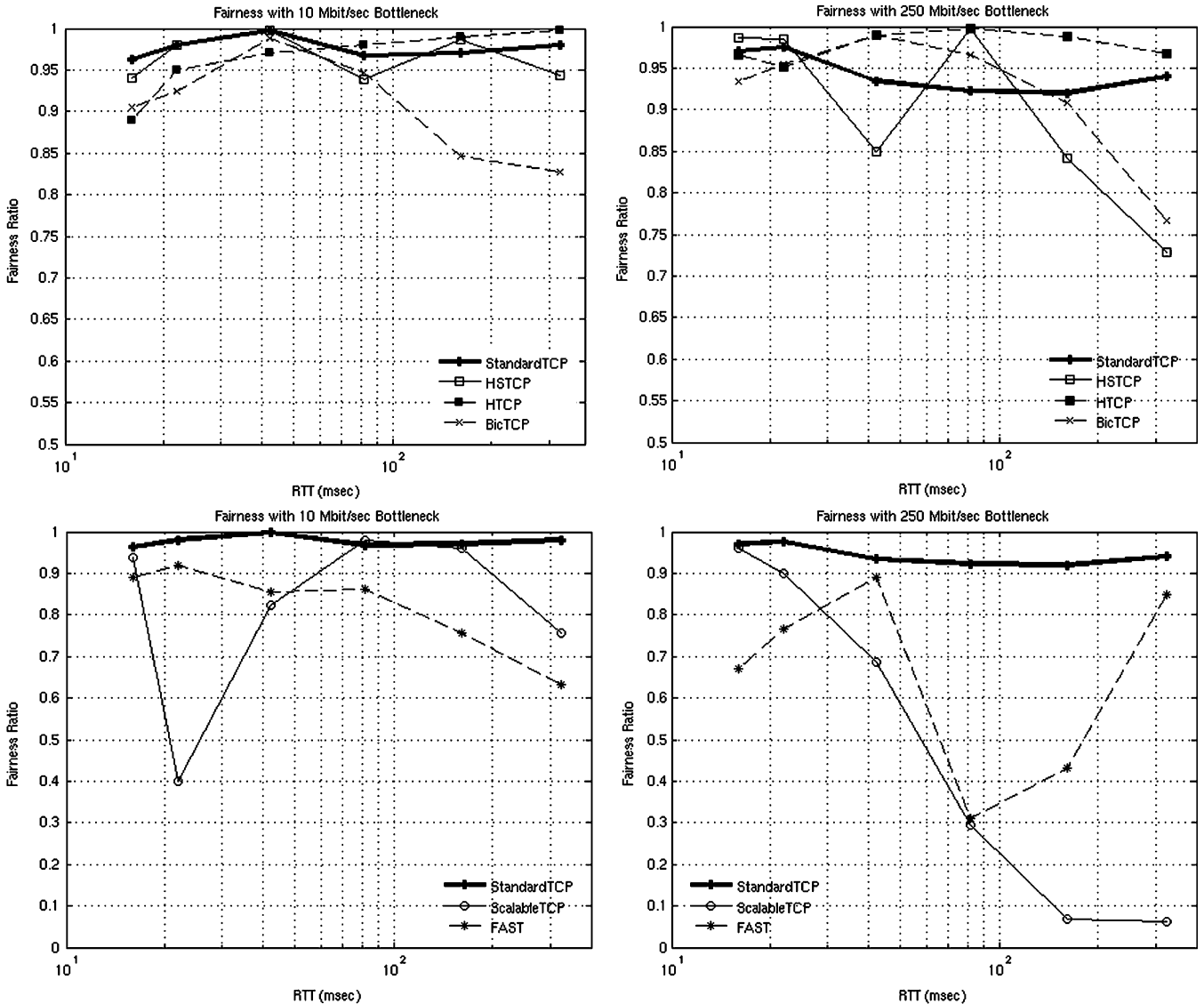


Fig. 3. Ratio of throughputs of two flows with the same RTT (also sharing same bottleneck link and operating same congestion control algorithm) as path propagation delay is varied. Results are shown for 10 and 250-Mb/s bottleneck bandwidths. The bottleneck queue size is 20% BDP, no web traffic. Observe that while standard TCP and H-TCP are essentially fair (the competing flows achieve, to within 5%, the same average throughput) under these conditions, scalable-TCP and FAST-TCP are notably unfair. HS-TCP and BIC-TCP can also be seen to exhibit significant unfairness, albeit to a lesser degree than scalable-TCP and FAST-TCP.

control algorithm consistently ensures that each flow achieves the same (to within less than 5%) average throughput. However, the measurements shown in Fig. 3 indicate that many of the proposed protocols exhibit substantial unfairness under the same conditions. While both FAST-TCP and scalable-TCP display very large variations in fairness, BIC-TCP and HS-TCP also display significant levels of unfairness.

In view of the somewhat surprising nature of these results, it is worthwhile investigating this behaviour in more detail. We consider, in turn, each of the protocols exhibiting greater levels of unfairness than standard TCP.

- *Scalable-TCP*: Fig. 4 shows typical examples of measured *cwnd* time histories. It can be seen that the *cwnd*s either do not converge to fairness or else converge very slowly indeed (not reaching fairness within the 10-min duration of these tests). Although sometimes expressed as a modified additive increase algorithm, it is easily shown that the scalable-TCP algorithm is, in fact, a multiplicative-increase

multiplicative-decrease (MIMD) algorithm, and this appears to explain much of the observed behavior. It has been known since the late 1980s [5] that in drop-tail networks, such algorithms may not converge to fairness. Further, in the case of MIMD flows with different round-trip times, theory predicts that the flow with the shortest round-trip time can seize essentially the entire link capacity [3], and this type of behavior is evident in our experimental results presented later. Note that this behavior is *not* confined to synchronized patterns of packet drop and is also observed when significant levels of Web traffic are present (although [3] considers synchronized drop-tail environments, the analysis can be readily extended the unsynchronized drops with similar conclusions).

- *FAST-TCP*: Fig. 5 shows typical examples of measured *cwnd* time histories when using the FAST-TCP algorithm. The upper figure shows measurements taken on a 250-Mb/s path with 42-ms propagation delay. Rapid vari-

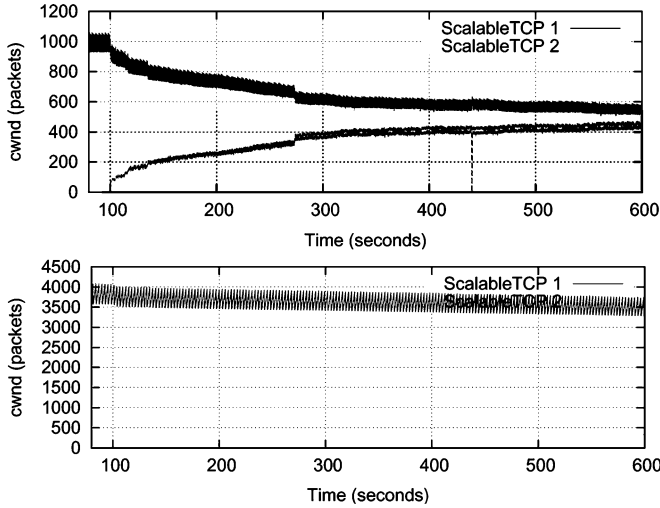


Fig. 4. Scalable-TCP $cwnd$ time histories following startup of a second flow. RTT of both flows is 42 ms (top) and 162 ms (bottom). Bottleneck bandwidth is 250 Mb/s, queue size 20% BDP, no Web traffic.

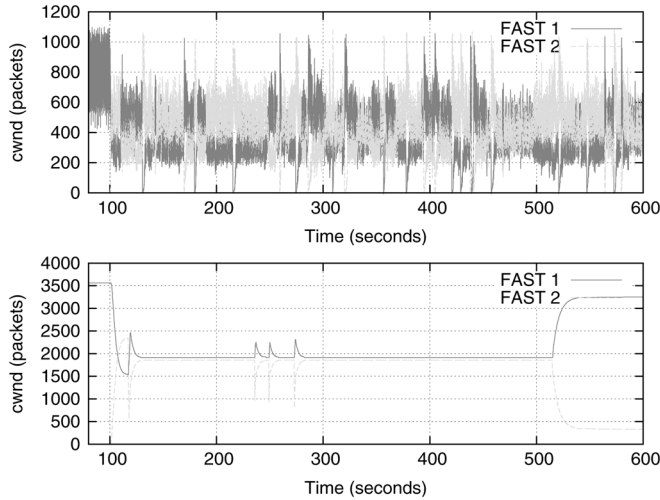


Fig. 5. FAST-TCP $cwnd$ time histories following startup of a second flow. RTT is 42 ms (top) and 162 ms (bottom). Bottleneck bandwidth is 250 Mb/s, queue size 20% BDP, no Web traffic.

ations in $cwnd$ are evident that are somewhat surprising in view of the delay-based rather than loss-based nature of the FAST-TCP algorithm. The lower figure shows the $cwnd$'s measured when the propagation delay on the path is increased to 162 ms. The rapid variations in $cwnd$ are no longer present, but the flows now exhibit a number of abrupt changes in $cwnd$, including a sharp increase in unfairness after 500s. It is perhaps worth emphasizing that these examples are representative of our measurements across a wide range of network conditions and are not selected as worst case behaviors. The purpose in this paper is not to explain the performance of the FAST-TCP algorithm. We do, however, comment that the behavior in the low latency example appears to be associated with use of a large value of f_α . Roughly speaking, each FAST flow attempts to maintain $f_\alpha/2$ packets in the queue at the bottleneck link. Hence, with n flows a queue size of at least $n \cdot f_\alpha/2$ is needed to avoid flooding the queue and inducing

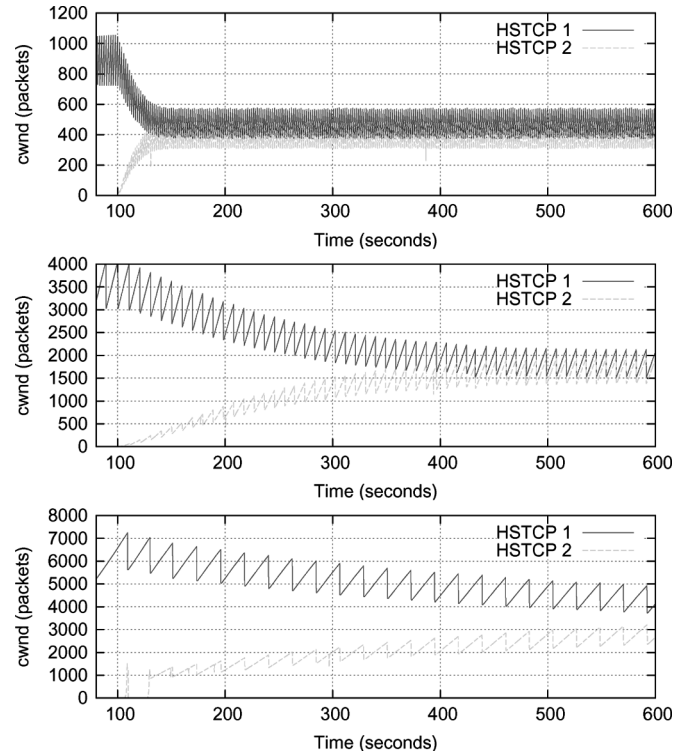


Fig. 6. HS-TCP $cwnd$ time histories following startup of a second flow. RTT is 42 ms (top), 162 ms (middle) and 324 ms (bottom). Bottleneck bandwidth 250 Mb/s, queue size 20% BDP, no Web traffic.

many packet losses. For link speeds above 100 Mb/s, $f_\alpha/2 = 100$, and so with two flows, we need a queue of at least 200 packets. However, for a 250-Mb/s link and 42-ms delay, a 20% BDP queue is only 175 packets. The behavior in the high-latency example in Fig. 5 appears to be associated with the adaptive switching of the f_α parameter value. If flows happen to adapt to different values of f_α , this can lead to substantial unfairness as f_α can take values in a range covering two orders of magnitude. Moreover, this unfairness can be sustained since the f_α is updated based on throughput. For example, choosing a low value of f_α leading to a low throughput share, in turn, leads to the continuing choice of a low value for f_α , while conversely once a flow chooses a high value of f_α such that receives a high throughput share then this leads to it maintaining a high value of f_α . As a result, the network can remain indefinitely in an unfair configuration.

- **HS-TCP:** Fig. 6 shows examples of HS-TCP $cwnd$ time histories for flows with the same round-trip time following startup of a second flow. It can be seen that the flows do converge to fairness, but that the convergence time can be long. This effect becomes more pronounced as the path propagation delay is increased. These experimental measurements are in good agreement with the simulation results previously reported in [18]. Recall that the AIMD increase parameters are functions of $cwnd$ in HS-TCP. The slow convergence appears to originate in the asymmetry that exists in HS-TCP between the AIMD parameters of newly started flows (with small $cwnd$) and existing flows (with large $cwnd$). Existing flows with large $cwnd$ have

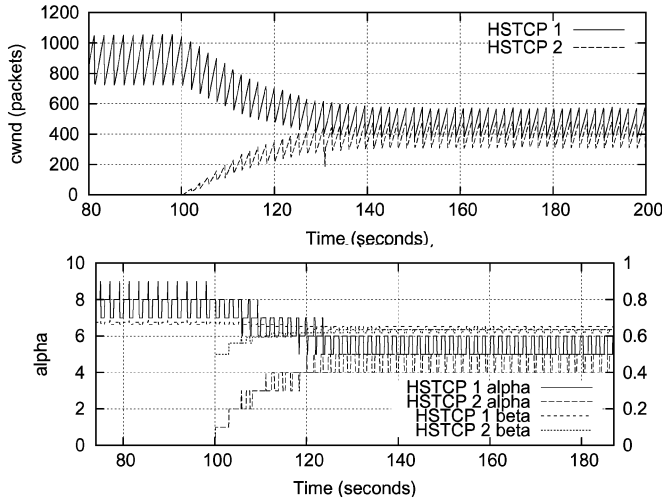


Fig. 7. Detailed HS-TCP $cwnd$ time histories (top) and α , β time histories (bottom) following startup of a second flow. RTT is 42 ms, bottleneck bandwidth 250 Mbit/sec, queue size 20% BDP, no Web traffic.

more aggressive values of increase and decrease parameters than do newly started flows, which have small $cwnd$. Hence, new flows are at a disadvantage and sustained unfairness can occur. Note that similar behavior is also observed as we vary the level of Web traffic. We also comment briefly upon the 250-Mb/s, 42-ms measurement for HS-TCP shown in Fig. 3. The $cwnd$ time histories corresponding to this measurement are shown in Fig. 6, and in more detail in Fig. 7. It can be seen that there appears to be long-term unfairness between the two flows that persists after the flows have converged to steady state. Also shown in Fig. 7 are the measured values of the AIMD α and β parameters for each flow. The long-term unfairness appears to be due to the granularity of the lookup table used to implement the HS-TCP $cwnd$ update rules. The current implementation uses a simple nearest neighbor type of table lookup to find the α and β values for the current value of $cwnd$. The granularity of this process could be readily reduced, e.g., by including more table entries or by interpolating between entries when performing a lookup, and our measurements indicate that it would be of benefit to refine the implementation in this manner.

- **BIC-TCP:** Fig. 8 shows examples of the $cwnd$ time history of BIC-TCP following startup of a second flow. It can be seen that as the path propagation delay increases the $cwnd$'s converge increasingly slowly, not reaching fairness within the 10-min duration of these tests when the path propagation delay is large. This behavior manifests itself in Fig. 3 as a fall in the measured fairness as propagation delay increases.
- **HTCP:** Fig. 9 shows $cwnd$ time histories of H-TCP following startup of a second flow. The equal sharing achieved between the two competing flows is evident.

B. Fairness With Different RTTs

Fig. 10 shows the ratio of measured throughputs when the propagation delay of the first flow is held constant at 162 ms and the propagation delay of the second flow is varied. Again, results

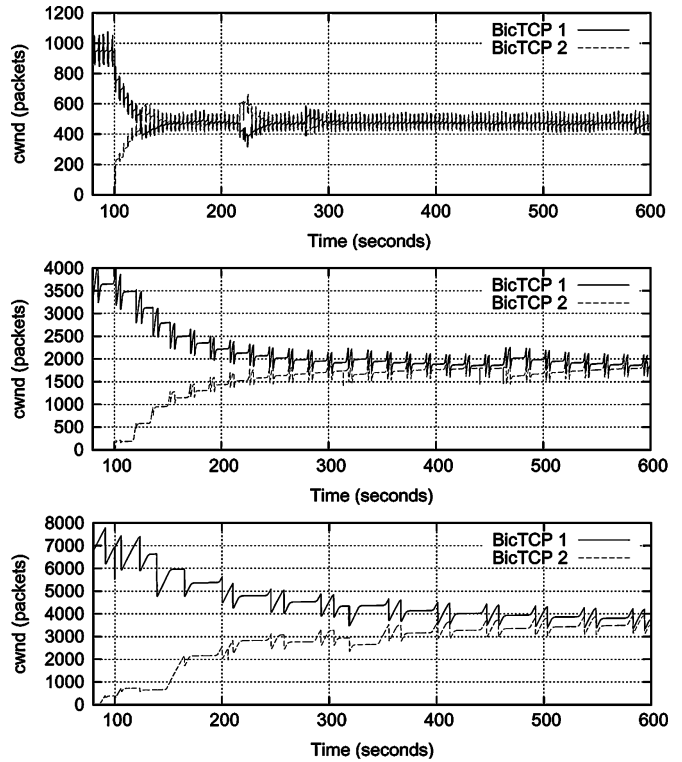


Fig. 8. BIC-TCP $cwnd$ time histories following startup of a second flow. RTT is 42 ms (top), 162 ms (middle), and 324 ms (bottom). Bottleneck bandwidth is 250 Mb/s, queue size 20% BDP, no Web traffic.

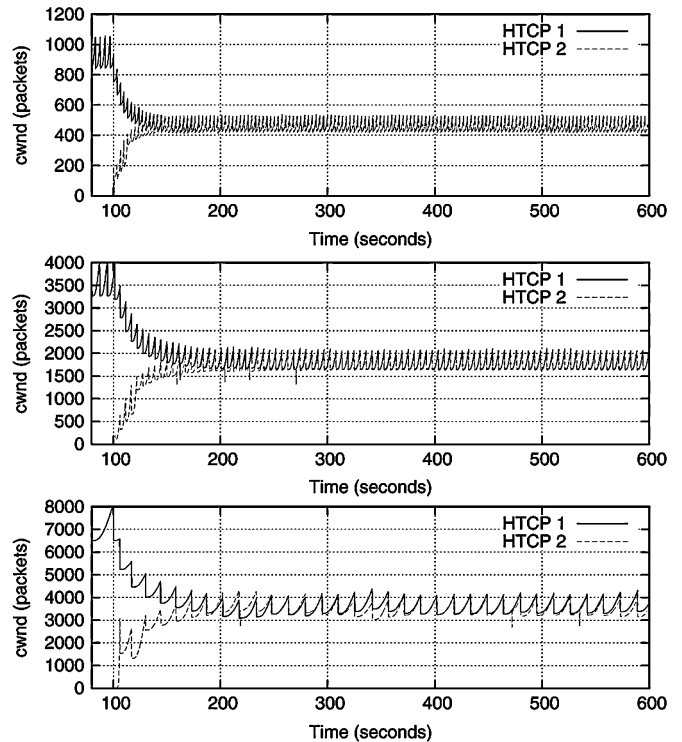


Fig. 9. H-TCP $cwnd$ time histories following startup of a second flow. RTT is 42 ms (top), 162 ms (middle), and 324 ms (bottom). Bottleneck bandwidth is 250 Mb/s, queue size 20% BDP, no Web traffic.

are shown both for a bottleneck link bandwidth of 10 and 250 Mb/s. Results are shown when the queue is sized at 20% BDP,

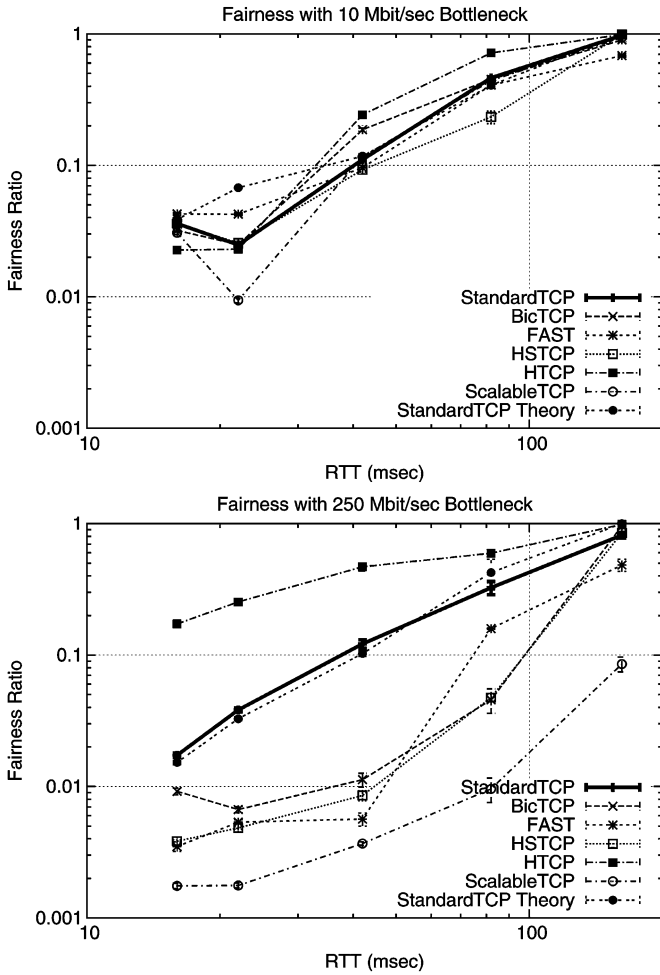


Fig. 10. Ratio of throughputs of two competing flows as the propagation delay of the second flow is varied. Results are shown for 10-Mb/s (top) and 250-Mb/s (bottom) bottleneck bandwidths. Flow 1 has RTT of 162 ms, the RTT of Flow 2 is marked on the x axis of the plots. Queue size is 20% BDP, no Web traffic.

but similar results are also obtained when the queue is 100% BDP. The results shown are also for no Web traffic as we find that the level of Web traffic has little impact on the measured fairness (see Section V-G for further details). As a check on our experimental setup, also plotted on Fig. 10 are the throughputs for standard TCP predicted by theory [19].

It can be seen from Fig. 10 that scalable-TCP, HS-TCP, BIC-TCP, and FAST-TCP all exhibit significantly greater RTT unfairness than standard TCP. The degree of unfairness can be nearly an order of magnitude greater than that with standard TCP and is such that long round-trip time flows may be essentially starved of bandwidth. To give a feel for this, a ratio of 0.003 in flow throughputs (the lowest ratio observed with FAST-TCP, HS-TCP in Fig. 10; note that scalable-TCP exhibits still greater unfairness) corresponds to a throughput of approximately 249.5 Mb/s for the short RTT flow and a throughput of only 0.5 Mb/s for the long RTT flow. This compares with throughputs of 245 Mb/s and 5 Mb/s for a ratio of 0.02 (the lowest ratio observed with standard TCP)—observe that the throughput of the long RTT flow is now an order of magnitude greater—and 200 Mb/s and 50 Mb/s for a ratio of 0.2 (the lowest ratio observed with H-TCP).

With regard to scalable-TCP, as noted previously this adopts an MIMD strategy. In the case of MIMD flows with different round-trip times, theory predicts that the flow with the shortest round-trip time can seize essentially the entire link capacity [3], and this is indeed what we observe.

The increased level of RTT unfairness evident with HS-TCP is associated with the AIMD increase and decrease parameters being functions of flow $cwnd$. This means that unfairness tends to be amplified. For example, suppose the network is perturbed so that the $cwnd$ of one flow is increased while that of another flow is decreased. The flow with larger $cwnd$ adjusts its AIMD parameters to become more aggressive; at the same time, the flow with the smaller $cwnd$ adjusts its AIMD parameters to be less aggressive. There is thus a reinforcing action that tends to increase the level of unfairness. A similar effect also appears to occur with BIC-TCP.

The lower level of RTT unfairness in H-TCP compared with standard TCP is associated with the use of RTT scaling in H-TCP. This yields RTT unfairness whereby the measured throughput ratio is proportional to the ratio of flow RTTs. With standard TCP, the measured throughput ratio is, of course, proportional to the square of the flow RTT ratio.

C. Backward Compatibility

Fig. 11 plots the ratio of measured throughputs of two flows with the same propagation delay and a shared bottleneck link. The first flow operates the standard TCP algorithm, while the second flow operates a new TCP variant. Results are shown both for bottleneck link bandwidths of 10 and 250 Mb/s. It can be seen that scalable-TCP and FAST-TCP exhibit the greatest degree of unfairness in both low and high-speed conditions.

The unfairness between scalable-TCP and standard TCP in low-speed conditions is perhaps surprising in view of the mode switch whereby scalable-TCP behaves as standard TCP at low $cwnd$ sizes. The observed unfairness appears to occur due to the following effect. When the flow $cwnd$ is below the *Low_Window* threshold, it indeed behaves as standard TCP, and in the low-speed tests, fair operation has the flow $cwnd$'s below this threshold. However, perturbations in $cwnd$ (e.g., due to unsynchronized packet drops) can lead to it crossing the *Low_Window* threshold. When this occurs, the flow switches to the scalable algorithm. The scalable algorithm is more aggressive than standard TCP and so once it is activated this can lead to long-term unfairness whereby the flow thereafter maintains its $cwnd$ above *Low_Window*. Note that we did not observe such behavior with HS-TCP, which employs a similar mode switch. This appears to be due to the fact that the transition to high-speed operation is smoother in the sense that a relatively large increase in $cwnd$ above *Low_Window* is required before the HS-TCP becomes significantly more aggressive than Standard TCP.

D. Efficiency

Fig. 12 shows measured aggregate throughput of two TCP flows with the same propagation delay as a function of queue size on a 100-Mb/s link. As a validation check, also plotted on Fig. 12 is the efficiency for standard TCP predicted by *NS* simulations. It can be seen that the experimental and simulation throughputs are in good agreement.

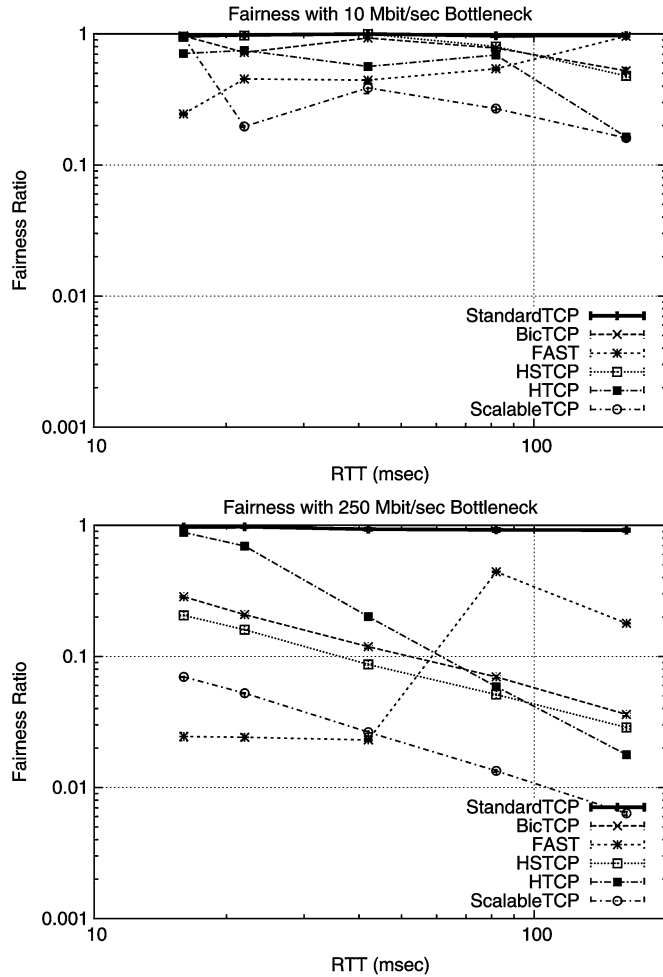


Fig. 11. Ratio of throughputs of competing new-TCP and standard TCP flows as path propagation delay is varied. Results are shown for 10-Mb/s (top) and 250-Mb/s (bottom) bottleneck bandwidths. Both flows have the same RTT. Queue size is 20% BDP, no Web traffic.

It can be seen from Fig. 12 that for buffer sizes above 10% of the bandwidth-delay product, the new protocols uniformly achieve better throughput than standard TCP. Observe, however, that in all cases the throughput falls rapidly when the buffer size becomes less than about 3% of the bandwidth-delay product (or less than about 8% BDP in the case of FAST-TCP). It can be seen from the packet loss measurements in Fig. 12 that the drop in link utilization corresponds to a substantial (around two orders of magnitude) rise in packet loss rate.

The drop in link utilization appears to be associated with an increased incidence of packet bursts flooding the buffer when it becomes very small. In this example, a 2% BDP buffer is only 13 packets, while a 1% BDP buffer is only six packets, compared with a BDP of 683 packets. Delayed acking leads to many back-to-back pairs of data packets being sent. Growth of the flow *cwnd*'s leads to injection of new packets following receipt of an ACK, thereby also generating regular packet triples. In fact, we have observed frequent transmission of six to ten packets per ACK, presumably due to end host scheduling granularity—the 1-ms clock tick used corresponds to approximately eight 1500-byte packets at 100 Mb/s. Since the packet streams of two flows are aggregated at the router, we therefore have that

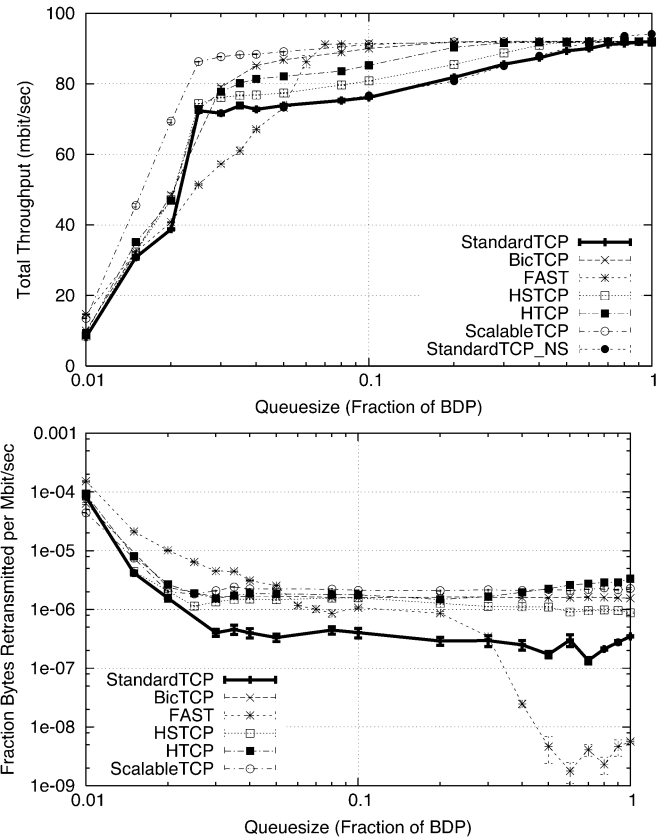


Fig. 12. Aggregate throughput (top) and packet loss (bottom) of two competing TCP flows with 100-Mb/s bottleneck bandwidth. Both flows have end-to-end round-trip propagation delays of 82 ms. BDP is 683 packets.

bursts of 1%–2% BDP at the router are common. It is interesting to note that, despite the more aggressive nature of scalable-TCP, HS-TCP, BIC-TCP, and H-TCP, the corresponding threshold in queue size below which throughput rapidly falls is similar to that for standard TCP. This suggests that this short time-scale burst structure of the packet stream is largely unaffected by the changes introduced in these congestion control algorithms.

In the case of FAST-TCP, rate pacing is used, but the accuracy of the pacing is limited by end host scheduling granularity. At 100 Mb/s, it takes 0.12 ms to transmit a 1500-byte packet and 0.24 ms to transmit a packet pair, whereas the scheduling granularity is on the order of 1 ms. Pacing, therefore, has only a limited impact in the context of the buffer sizes considered here. In addition, as noted previously, two FAST flows will attempt to maintain a standing queue of f_α packets, with $f_\alpha = 8$ packets at 10 Mb/s and 20 packets at 100 Mb/s. This standing queue reduces the space within the router buffer to accommodate packet bursts. Hence, a 5% BDP buffer of 34 packets may be reduced to an effective buffer of only 14 packets i.e., a similar effective buffer size to that at which the throughput of the other congestion control algorithms collapses.

E. Response Function

Measurements of the response functions are shown in Fig. 13. Also marked are the response functions for standard TCP, scalable-TCP, and high-speed TCP predicted by theory [8]. It can be seen that the measured response functions of standard TCP,

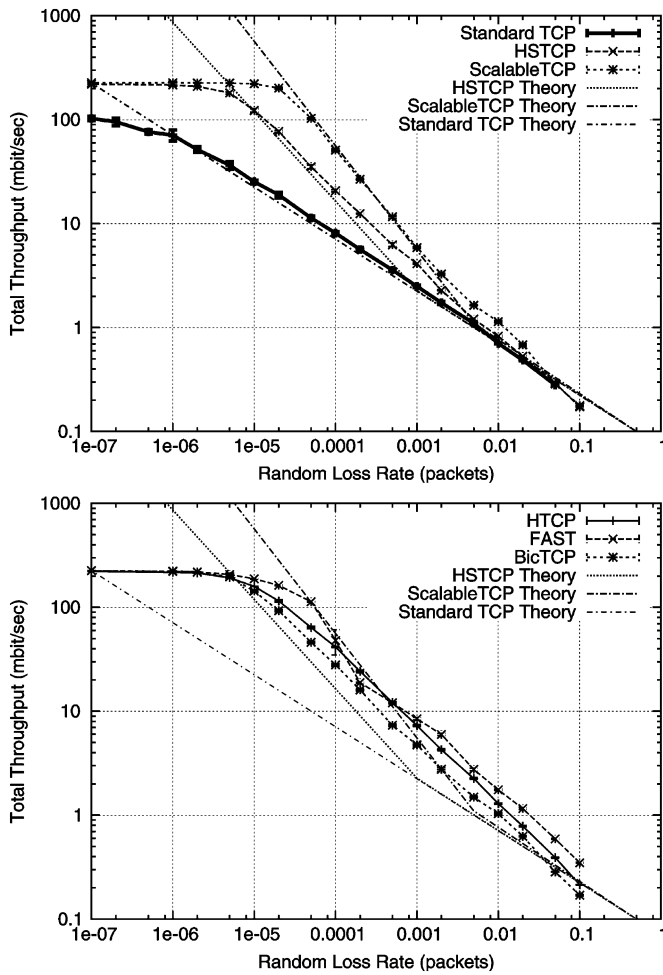


Fig. 13. Measured response functions, 250-Mb/s bottleneck, 162-ms RTT.

scalable-TCP, and HS-TCP are in fairly good agreement with theory, although some discrepancy is evident around the mode switch transition from standard to high-speed operation.

F. Convergence Time

Fig. 14 plots the measured convergence time following startup of a second flow. The values plotted are the average of multiple tests and a range of random start times for the second flow. The convergence time is plotted versus path propagation delay (both flows have the same propagation delay in this experiment), and results are presented for link rates of 10 and 250 Mb/s.

It can be seen that, in line with the previous discussion, that scalable-TCP, HS-TCP, and BIC-TCP all exhibit extremely slow convergence times (or, indeed, nonconvergence). We comment briefly on H-TCP and FAST-TCP.

- *H-TCP*. H-TCP exhibits similar convergence times to standard TCP under low-speed conditions. In higher speed conditions, the 80% convergence time levels off at around 30 s. This is illustrated, for example, in Fig. 9.
- *FAST-TCP*. FAST-TCP has the smallest measured convergence time of all the algorithms studied. These results need to be interpreted with some care, however. For example, it can be seen from Fig. 5 that while FAST may converge

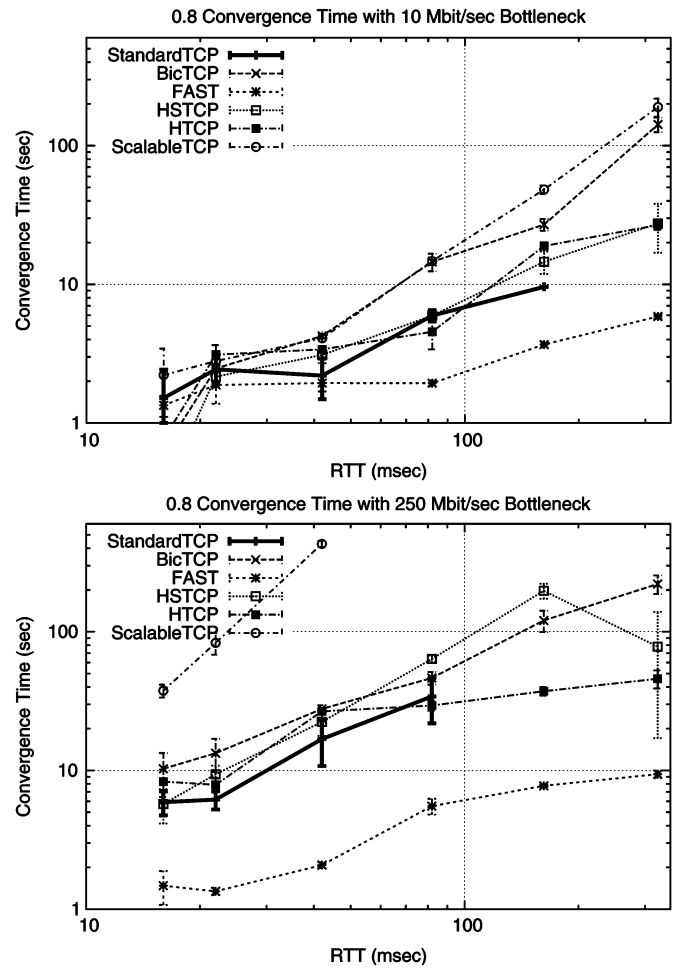


Fig. 14. Mean 80% convergence time following startup of a second flow. Results are shown for 10-Mb/s (top) and 250-Mb/s (bottom) bottleneck bandwidths. Both flows have same RTT. Queue size is 20% BDP. Missing points along the ordinate axis indicate that the flows did not converge to within the 80% fairness ratio over the 10-min duration of the test—this is especially evident with scalable-TCP and standard TCP at 250 Mb/s.

quickly initially, flows may later diverge again. It is important to emphasize that only the initial convergence time is captured by our convergence time metric.

G. Impact of Web Traffic

We observed that the level of Web traffic present made little difference to our measurements of fairness and responsiveness. For example, Fig. 15 plots the RTT unfairness between two long-lived flows as background Web traffic is varied from 0 to 50 sessions. Note that 50 Web sessions generate significant levels of traffic: mean throughput is typically around 1.5% link bandwidth with bursts (on the order of 1-s duration) in throughput of around 10% link bandwidth. It can be seen that both the trend and the actual unfairness values are nevertheless insensitive to the level of Web traffic.

On the face of it, this result is somewhat surprising. It has, for example, been well known for many years that deterministic phase effects can have a strong impact on fairness in networks with small numbers of long-lived flows, and it is also been observed, e.g., [19], that even small amounts of bidirectional Web traffic can randomize packet drops sufficiently to mitigate phase

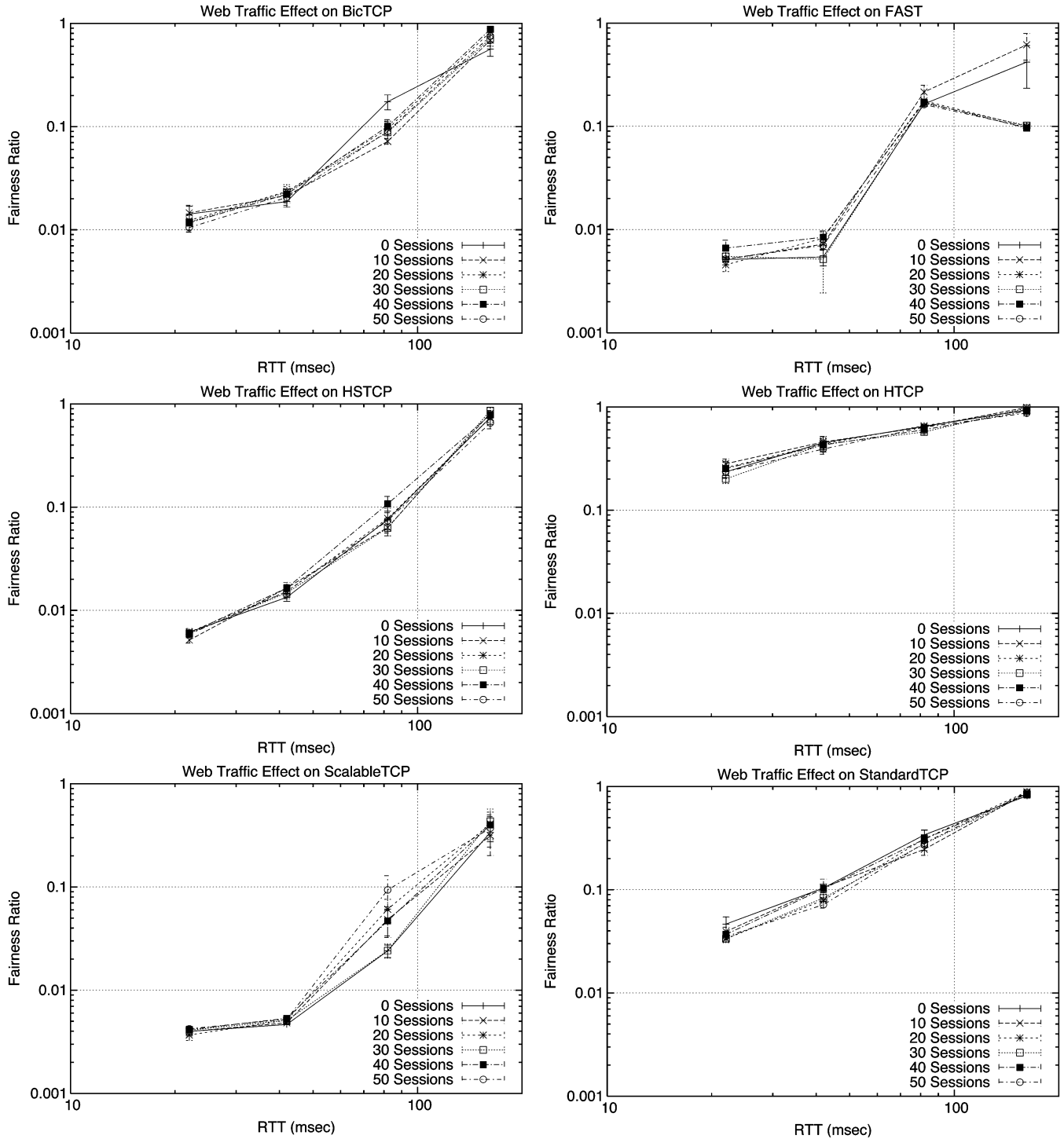


Fig. 15. Impact of bidirectional Web traffic on RTT unfairness of long-lived flows. Plots show ratio of throughputs of two competing long-lived flows as the propagation delay of the second flow and the number of background Web sessions is varied. Results are shown for 250-Mb/s bottleneck bandwidth. Flow 1 has RTT of 162 ms, the RTT of Flow 2 is marked on the x axis of the plots. Queue size is 20% BDP.

effects. We note, however, that these results are based on simulation data. Two differences between our experimental tests and these simulation studies are i) delayed acking is used in our experimental tests, and ii) on high-speed links, end host scheduling granularity can have a significant impact on the burst structure of the packet stream arriving at a router. Delayed acking introduces additional variable delays. Delayed acking also directly changes the burst structure of TCP packet streams as each ACK

arriving at the TCP sender generates a back-to-back packet pair rather than a single packet. This is compounded by end host scheduling granularity. In our tests, the operating system scheduling granularity (determined via the *HZ* kernel parameter) was left at its default setting of 1 ms. At 250 Mb/s, 1 ms is the transmission time of 21 1500-byte packets, and so the scheduling granularity can potentially have a significant impact on packet stream burstiness. Hence, taken together, it seems plau-

sible that these factors may well be sufficient to disrupt the delicate timing patterns that underly phase effects even when no Web traffic is present. This is of importance because once phase effects are mitigated, previous simulation studies [19] indicate that the impact of additional Web traffic on the fairness of competing long-lived flows is relatively minor, and this would be consistent with our present experimental measurements.

VI. RELATED WORK

Performance measurements are included in many papers proposing modifications to the TCP congestion control algorithm and we briefly mention here the main studies relevant to the present paper. In [12], Kelly presents an experimental comparison of the aggregate throughput performance of scalable-TCP and standard TCP. In [11], Low and coauthors present throughput and packet loss measurements from a lab-scale test network for FAST-TCP, HS-TCP, scalable-TCP, BIC-TCP, and TCP-Reno. In [10], aggregate throughput measurements are presented for FAST-TCP and TCP-Reno. In all of these studies, measurements focus on aggregate throughput i.e., link utilization. Measurements are also essentially confined to single-case studies. Hence, efficiency as a function of queue size is not considered, nor fairness, friendliness, responsiveness, and convergence times.

In [9], throughput and *cwnd* time histories of FAST-TCP, HS-TCP, scalable-TCP, and TCP-Reno are presented for a lab-scale experimental testbed. Aggregate throughput, throughput fairness (measured via Jain's index) and a number of other measures are presented. However, results are confined solely to an 800-Mb/s bottleneck link with a 2000-packet buffer. No attempt is made to control for changes to the Linux network stack implementation that are unrelated to the congestion control algorithm. The impact of link rate, RTT, queue size, and level of Web traffic on fairness and responsiveness are not considered, nor is the impact of queue size on efficiency. In [20], *NS* simulation results are presented comparing the performance of HS-TCP, scalable-TCP, BIC-TCP, and standard TCP.

We note that the foregoing papers all propose changes to the TCP congestion control algorithm and present performance measurements in support of these changes. The evaluation of competing proposals, per se, has received far less attention. Notably, [4] and [6] present evaluation studies specifically targeted at measuring the performance of TCP proposals. Experimental measurements are presented for scalable-TCP, HS-TCP, FAST-TCP, H-TCP, BIC-TCP, HSTCP-LP, and P-TCP (i.e., 16 parallel standard TCP flows) over network paths within the United States and between the United States and Europe. Measurements presented include aggregate throughput and throughput fairness (via Jain's index). RTT unfairness, convergence time, and impact of queue provisioning are not considered. No attempt is made to control for changes to the Linux network stack implementation unrelated to the congestion control algorithm.

VII. SUMMARY AND CONCLUSION

In this paper, we present experimental results evaluating the performance of the scalable-TCP, HS-TCP, BIC-TCP, FAST-TCP, and H-TCP proposals in a series of benchmark tests.

We find that many recent proposals perform surprisingly poorly in even the most simple test, namely achieving fairness

between two competing flows in a dumbbell topology with the same round-trip times and shared bottleneck link. Specifically, both scalable-TCP and FAST-TCP exhibit very substantial unfairness in this test.

We also find that, with the notable exception of H-TCP, all of the proposals studied induce significantly greater RTT unfairness between competing flows with different round-trip times. The unfairness can be an order of magnitude greater than that with standard TCP and is such that flows with longer round-trip times can be completely starved of bandwidth.

While the TCP proposals studied are all successful at improving the link utilization in a relatively static environment with long-lived flows, in our tests many of the proposals exhibit poor responsiveness to changing network conditions. We observe that scalable-TCP, HS-TCP, and BIC-TCP can all suffer from extremely slow (>100 -s) convergence times following the startup of a new flow. We also observe that while FAST-TCP flows typically converge quickly initially, flows may later diverge again to create significant and sustained unfairness.

With regard to link utilization, for moderate to large buffer sizes, we find that all of the proposed high-speed algorithms yield higher throughput than standard TCP on a high-speed path. With very small buffers, we observe that microscale packet bursts lead to a rapid fall in throughput efficiency. The threshold buffer size below which this occurs is approximately the same for all congestion control algorithms studied, with the exception of FAST-TCP, where the threshold is somewhat higher, owing to the standing queue created by the delay-based congestion control action used in FAST-TCP.

We argue that our results demonstrate that the consistent application of standardized tests can yield results of considerable value. This can not only be used to screen new proposals prior to full-scale experimental testing, with its associated costs, but can also provide a useful step towards establishing a sound basis for the development of new protocols.

APPENDIX I

The base Linux kernel includes rate-halving and delayed acking. In addition, the *altAIMD* kernel incorporates the following.

- i) *New-TCP Stacks*. Each of the congestion control algorithms studied have independent patches that are publicly available. To provide consistency, and control against the influence of differences in network stack implementation as opposed to differences in the congestion control algorithm itself, we incorporated the implemented congestion control algorithms into a common network stack.
- ii) *Appropriate Byte Sizing (RFC3465)* [1]. The counting of *ack*'s by the number of bytes acknowledged rather than the number of *ack*'s received to counter the problems of *cwnd* growth under delayed *ack*'s.
- iii) *SACK Processing Improvements* [13]. The implementation of SACK processing in the Linux kernels requires a processing time that is $O(cwnd)$. This has serious performance implications on large bandwidth-delay product paths. We implemented a more robust algorithm with complexity of $O(\text{lost packets})$.
- iv) *Throttle Disabled* [13]. A build-up of *ack* packets at the sender can cause an overflow in the Linux network ring

buffers which invokes a throttle action that causes all packets to be dropped. We modified the ring buffers to operate a pure drop-tail discipline.

v) *Web100* [16]. Kernel was instrumented using Web100.

REFERENCES

- [1] M. Allman, "TCP congestion control with appropriate byte counting (ABC)," IETF RFC 3465, Feb. 2003.
- [2] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," in *Proc. ACM SIGCOMM*, 2004.
- [3] E. Altman, K. Avrachenkov, and B. J. Prabhu, "Fairness of MIMD congestion control algorithms," in *Proc. IEEE INFOCOM*, 2005, pp. 1350–1361.
- [4] H. Bullot, R. L. Cottrell, and R. Hughes-Jones, "Evaluation of advanced TCP stacks on fast long distance production networks," *J. Grid Comput.*, vol. 1, no. 4, pp. 345–359, Dec. 2003.
- [5] D. M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Comput. Netw. ISDN Syst.*, vol. 17, no. 1, pp. 1–14, Jun. 1989.
- [6] R. L. Cottrell, S. Ansari, P. Khandpur, R. Gupta, R. Hughes-Jones, M. Chen, L. MacIntosh, and F. Leers, "Characterization and evaluation of TCP and UDP-based transport on real networks," presented at the 3rd Workshop on Protocols for Fast Long-distance Networks, Lyon, France, 2005.
- [7] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the Internet," *IEEE/ACM Trans. Netw.*, vol. 7, no. 4, pp. 458–472, Aug. 1999.
- [8] S. Floyd, "High speed TCP for large congestion windows," IETF RFC 3649, Dec. 2003, Experimental.
- [9] C. Jin, D. X. Wei, and S. H. Low, "FAST TCP: Motivation, architecture, algorithms, performance," in *Proc. IEEE INFOCOM*, Hong Kong, 2004.
- [10] C. Jin, D. X. Wei, S. Low, G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, and S. Singh, "FAST TCP: From theory to experiments," *IEEE Network*, vol. 19, no. 1, pp. 4–11, 2005.
- [11] S. Hegde, D. Lapsley, B. Wydrowski, J. Lindheim, D. Wei, C. Jin, S. Low, and H. Newman, "FAST TCP in high speed networks: An experimental study," presented at the GridNets, San Jose, CA, 2004.
- [12] T. Kelly, "Scalable TCP: Improving performance in high-speed wide area networks," *Comput. Commun. Rev.*, vol. 33, no. 2, pp. 83–91, Apr. 2003.
- [13] D. J. Leith, "Linux implementation issues in high-speed networks," Hamilton Inst., Nat. Univ. of Ireland, Maynooth, Kildare, Ireland, Tech. Rep., 2003 [Online]. Available: www.hamilton.ie/net/Linux-HighSpeed.pdf
- [14] D. J. Leith and R. N. Shorten, "H-TCP protocol for high-speed long-distance networks," presented at the 2nd Workshop Protocols Fast Long Distance Networks, Argonne, Canada, 2004.
- [15] Y. T. Li and D. J. Leith, "BicTCP implementation in Linux kernels," Hamilton Inst., Nat. Univ. of Ireland, Maynooth, Kildare, Ireland, Tech. Rep., 2004 [Online]. Available: www.hamilton.ie/net/LinuxBicTCP.pdf
- [16] M. Mathis, J. Heffner, and R. Reddy, "Web100: Extended TCP instrumentation for research, education and diagnosis," *ACM Comput. Commun.*, vol. 33, no. 3, pp. 69–79, Jul. 2003.
- [17] J. Padhye, V. Firoiu, D. F. Towsley, and J. F. Kurose, "Modeling TCP Reno performance: A simple model and its empirical validation," in *Proc. ACM SIGCOMM 1998*, Vancouver, Canada.
- [18] R. N. Shorten, D. J. Leith, J. Foy, J. Foy, and R. Kilduff, "Analysis and design of AIMD congestion control algorithms in communication networks," *Automatica*, vol. 41, pp. 725–730, 2005.
- [19] R. N. Shorten, F. Wirth, and D. J. Leith, "A positive systems model of TCP-like congestion control: Asymptotic results," *IEEE/ACM Trans. Netw.*, vol. 14, no. 3, pp. 616–629, Jun. 2006.
- [20] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control for fast long-distance networks," in *Proc. IEEE INFOCOM*, Hong Kong, 2004.
- [21] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson, "Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 25, no. 4, pp. 100–113, Oct. 1997.



Yee-Ting Li received the Ph.D. degree in eScience from the University of London, London, U.K., in 2006, specializing in TCP transport protocols and their application in large-scale data intensive science projects. He also received the M.Sc. degree in physics from the University of London in 2001.

He is currently with Stanford Linear Accelerator. His research interests include transport protocols, on-demand bandwidth allocation, network monitoring, event detection and isolation, and Grid middleware.



Doug Leith received the B.Sc. (Eng.) degree from the University of Glasgow, Scotland, U.K., in 1986 and the Ph.D. degree from the University of Glasgow in 1989.

In 2001, he joined the National University of Ireland, Maynooth, as an SFI Principal Investigator, where he established the Hamilton Institute (Website: www.hamilton.ie) of which he is Director. His current research interests include the analysis and design of network congestion control and distributed resource allocation in wireless networks.



Robert N. Shorten is a Professor and Senior Researcher at the Hamilton Institute, National University of Ireland, Maynooth. He is also Co-Founder of the Hamilton Institute.

His research interests are stability theory, hybrid systems, and internet congestion control.