Experimental Evaluation of TCP Protocols for High-Speed Networks

Yee-Ting Li, Douglas Leith and Robert N. Shorten
Hamilton Institute, NUI Maynooth

***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.**

***Index Terms*— TCP Congestion control; Evaluation of TCP protocols; High-speed networks.**

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 too slowly. This is illustrated in Figure I which plots evolution of the $cwnd$ of a single flow, and its throughput time histories measured on a 1Gb/s trans-atlantic path between Dublin, Ireland and Chicago. The propagation delay is 100ms 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 218Mb/s. This inability to utilise network capacity is not confined to long distance inter-continental 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 standard TCP on paths with low bandwidth-delay product. This immediately leads to consideration of some form of low-speed/high-speed mode switch as part of proposed algorithms. Early work along these lines includes the HS-TCP proposal of Floyd[7], the Scalable-TCP proposal of Kelly[11] and the FAST-TCP proposal of Low *et al*[8]; more recent new proposals include BIC-TCP[20] and H-TCP[13]. These proposals have all been the subject of considerable interest and experimentation in recent years.

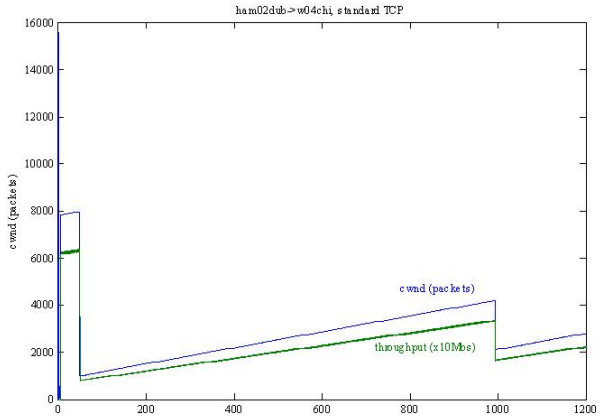

Fig. 1. Measured $cwnd$ and throughput time histories on 1Gb/s path between Dublin, Ireland and Chicago, USA. Over 1200s, the average throughput achieved is only 218Mb/s. These particular measurements were taken on the afternoon of Dec 9th 2003 using a dedicated trans-atlantic link with no significant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

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 form the lack of an agreed 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. Most existing work also fails to 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 conlusions 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. 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-

TCP[1] 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.

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 behaviour. The mode switch mechanism in H-TCP is a source for concern with this algorithm and needs to be studied in more detail before this algorithm, in its current form, can be deployed.

The paper is structured as follows. In Section II we discuss some issues that limit the utility of previous evaluation studies and motivate the present work. In Sections III and IV we introduce our testing framework and in Section V we present our experimental measurements. The implications of these results are discussed in more detail in Section VII and the conclusions are summarised in Section IX.

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 almost all 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 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 characterise TCP performance across a broad range of bandwidths (not just on high-speed links), propagation delays (not just trans-continental links) and router buffer sizes (not just very large or very small buffers).

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 characterising some important aspects of the behaviour of new protocols in a consistent and objective manner. While we lack agreed 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 a comparative approach to the protocol evaluation problem. Namely, we propose taking the performance of the current start-of-the-art TCP algorithm[2] as a baseline against which the behaviour of new proposals can be compared.

It is also important to emphasise 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 the 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 restricting our attention to 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. Short-lived TCP flows, and indeed non-TCP flows, constitute

[1]We note that H-TCP is developed by some of the authors of this paper. We emphasise 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.

[2]Implementations of standard TCP do differ in their behaviour. However, differences in implementation are largely confined to areas such as timeout 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.

a large proportion of traffic in real networks. Similarly, not all routers operate drop-tail queueing disciplines. However, as we shall see, restricting our attention to long-lived TCP flows operating in drop-tail environments is already sufficient to highlight many important features of new TCP proposals. The corresponding behaviour of the standard TCP algorithm is well studied and its use as a baseline for comparisons immediately suggests a number of fundamental characteristics to consider.

A. 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 [16]). 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.

B. Efficiency.

That is, utilisation 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 synchronised flows) link utilisation falls as the queue size is reduced below the delay-bandwidth product of a path. The number of competing flows is also known to influence efficiency (e.g. see [2]). As a minimum we therefore expect our tests to characterise efficiency with respect to these parameters. Since network conditions are not static, we are also interested in the ability to rapidly acquire and release bandwidth as conditions change, e.g. flows start and stop.

C. Overhead.

The standard TCP AIMD congestion control algorithm generates packet losses as part of its proper operation; that is, even a single flow on an uncongested link will generate packet losses owing to the probing action of the AIMD algorithm. These packet losses reduce goodput and are an overhead of the congestion control algorithm that we expect our tests should measure.

D. Backward compatibility.

It seems clear that a basic, practical, requirement on new protocols is that they support incremental rollout. That is, they immediately offer a tangible benefit without creating a large negative impact on users operating legacy protocols.

IV. Benchmark Tests

A. Definitions

Letting $u_i(t)$ denote the rate of packets transferred by the i'th flow per unit time, the *average throughput* is

$$\bar{u}_i := \lim_{T\to\infty} \frac{1}{T} \int_0^T u_i(s)ds, \tag{1}$$

The average *aggregate throughput*, $\bar{U}$, of n flows sharing a link is

$$\bar{U} := \sum_{i=1}^{n} \bar{u}_i. \tag{2}$$

It will also be useful to distinguish between the throughput of the i'th flow and the *goodput* of the i'th flow. The *goodput*, $\bar{x}_i$, is the rate at which data is successfully transferred by the i'th flow per unit time, i.e. throughput less losses. The *aggregate goodput* of n flows sharing a link is defined similarly. We also define the short-term average throughput as the exponentially weighted moving average

$$\hat{u}(t)_i = \int_0^t exp[-\lambda(t-s)]u_i(s)ds \tag{3}$$

Here, the exponential term introduces fading memory so that $\hat{u}(t)_i$ is, roughly speaking, the running average over a window of past data with the window size determined by the parameter λ. We choose λ to be proportional to the bandwidth-delay product so that the averaging window scales with the congestion epoch duration. 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. Test Setup

The observations in Section III directly motivate the tests defined in this section. Before proceeding, we consider some issues common to all of our proposed tests.

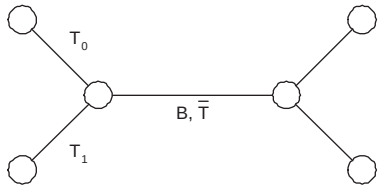

Fig. 2. Dumbbell topology. Bandwidth of bottleneck link is B, round-trip propagation delay of bottleneck link is $\bar{T}$ and queue size is q_{max}; round-trip propagation delays of edge links are T_1 and T_2 respectively.

We define our tests on the dumbbell topology shown in Figure 2. We recognize that this topology is a limited one, but the behaviour of standard TCP on this topology is well studied and so it provides a natural starting point.

We consider round-trip propagation delays in the range 16ms-320ms and bandwidths ranging from 1Mb/s-250Mb/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 characterises current communication networks. In all of our tests we consider delay values of 16ms, 40ms, 80ms, 160ms, 320ms and bandwidths of 1Mb/s, 10Mb/s, 100Mb/s and 250Mb/s. This defines a grid of measurement points where, for each value of delay, performance is measured for each of the values of bandwidth.

Test durations are chosen long enough to provide accurate measurements of average throughput.

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.

C. Fairness

To evaluate fairness, we consider two TCP flows and propose the following tests:

(i) *Fairness under symmetric conditions*. Measure the average throughput of each flow under symmetric network conditions i.e. when each flow operates the same congestion control algorithm, has the same propagation delay and a shared bottleneck link. Measurements are taken for a range of propagation delays and link bandwidths (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 RTT's*. Measure the average throughputs as the propagation delay of the first flow is held constant and that of the second flow is varied from 16ms-320ms. Measurements are taken for a range of link bandwidths 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

To evaluate efficiency (link utilisation) and overhead (intrinsic packet loss[3]), we consider two TCP flows having the same propagation delay and propose the following two tests:

(i) *Efficiency vs 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 vs 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.

As noted above, these tests are carried out for a range of propagation delays and link bandwidths.

[3]In our experiments we measure the loss overhead by measuring packet retransmissions.

F. Response Function

To evaluate the impact of random packet loss on efficiency, similarly to Padhye *et al* [16] and Floyd[6] we consider the following test. Configure the network to generate random packet losses with constant per-packet drop probability – this can be achieved at the TCP receiver end host using standard tools. Measure the average throughput of a single TCP flow as the level of random packet losses is varied. As usual, these measurements are carried out for a range of propagation delays and link bandwidths.

G. Convergence Time

We evaluate the responsiveness 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, measurements are carried out for a range of propagation delays and link bandwidths.

V. 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 of these competing proposals. The reader is referred to the original literature for more detailed information.

A. Scalable-TCP [11]

.

The basic idea in Scalable-TCP 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

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

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 [7]

.

HS-TCP 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 $cwnd$.

That is, HS-TCP proposes that the TCP $cwnd$ be updated as follows

$$\begin{array}{rll} \text{Ack:} & cwnd & \leftarrow cwnd + \dfrac{f_\alpha(cwnd)}{cwnd} \\ \text{Loss:} & cwnd & \leftarrow g_\beta(cwnd) \times cwnd \end{array}$$

In [7] 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 [13]

.

HTCP 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 utilisation based on an estimate of the queue provisioning on a path. In more detail, H-TCP proposes that $cwnd$ be updated as follows

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

with

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

where Δ_L is a specified threshold such that the standard TCP update algorithm is used while $\Delta \le \Delta_L$. A quadratic increase function $\bar{f}_\alpha$ is suggested in [13], 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 [20]

. BIC-TCP 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, the BIC-TCP update algorithm is as follows.

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

with

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

BIC-TCP also implements an algorithm whereby upon low utilisation 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 [8]

.

FAST-TCP is a delay based algorithm. In outline, FAST-TCP proposes updating $cwnd$ as follows

$$\begin{array}{rll} \text{Ack:} & cwnd & \leftarrow min(2 \times cwnd, \\ & & (1-\gamma)cwnd + \gamma[\dfrac{T_{min}}{\bar{T}}cwnd + f_\alpha(B, T_q)]) \\ \text{Loss:} & cwnd & \leftarrow cwnd/2 \end{array}$$

with

$$f_\alpha(B, T_q) = \begin{cases} a \times cwnd & T_q = 0 \\ \bar{f}_\alpha(B) & otherwise \end{cases}$$

where γ is a design parameter, T_{min} and $\bar{T}$ are the minimum and average observed latencies of the flow respectively and T_q is the estimated round trip queuing delay. The function $\bar{f}_\alpha(B)$ depends upon the measured throughput B achieved by the flow: currently, $\bar{f}_\alpha(B)$ is set to 8, 20 and 200 for achieved throughputs of less than 10Mbit/sec, less than 100Mbit/sec and greater than 100Mbit/sec 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.

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 Figure 3. All sender and receiver machines used in the tests have identical hardware and software configurations as shown in Table I (see Appendix) and are connected to the switches at 1Gb/sec. The router, running the FreeBSD dummynet software, can be configured with various bottleneck queue-sizes, capacities

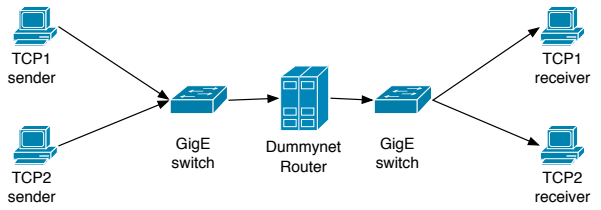

Fig. 3. Experimental set-up.

and round trip propagation delays to emulate a range 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[12]. 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 Appendix for further details[4].

The kernel is instrumented with the web100 extensions [15] to allow measurement of TCP variables.

In order to minimise the effects of local hosts queues and flow interactions, unless otherwise stated we only ran one flow per PC. Flows are injected using `iperf` into the testbed. Each individual test was run at least ten minutes 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 5 times and the arithmetic mean taken. An error on the measurement was taken as the standard error from this mean.

VI. 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 Test.

Figure 4 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 minutes duration. Results are shown both for a bottleneck link bandwidth of 10 Mb/s and 250Mb/s, roughly corresponding to low and high-speed network conditions. Figure 5 shows the ratio of measured throughputs when the propagation delay of the first flow is held constant at 162ms and the propagation delay of the second flow is varied. Again, Results are shown both for a bottleneck link bandwidth of 10 Mb/s and 250Mb/s. Results are shown when the queue is sized at 20% BDP but similar results are also obtained when the queue is 100% BDP.

Comment: As a validation check, also plotted on Figure 5 are the throughputs for standard TCP predicted by the theoretical analysis in [18]. Namely, the throughput given by the expression

$$\bar{u_i} = \frac{\alpha}{\lambda_i T_i (1-\beta))} \tag{4}$$

where α and β are the AIMD increase and decrease parameters (having values of 1 and 0.5, respectively, for standard TCP), T_i is the round-trip time of flow i, λ_i is the synchronisation factor i.e. the proportion of network congestion events at which the flow i experiences a packet drop and backs off its $cwnd$ (λ_i is estimated from the measured $cwnd$ time histories). It can be seen that the experimental and theoretical throughputs are in good agreement.

B. Backward Compatibility Test.

Figure 6 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 Mb/s and 250Mb/s.

C. Efficiency Test.

Figure 7 shows measured aggregate throughput of two TCP flows with the same propagation delay as a function of queue size on a 100Mb/s link.

Comment: As a validation check, also plotted on Figure 7 is the efficiency for standard TCP predicted by NS simulations. It can be seen that the experimental and simulation throughputs are in good agreement.

D. Overhead Test.

Figure 8 shows the measured packet loss rate of the various protocols versus queue size. Figure 9 also shows measured packet loss rate versus propagation delay.

E. Response Function Test.

Measurements of the response functions are shown in Figure 10.

[4]We note that the implementation of BIC-TCP included in the standard Linux 2.6.6 kernel distribution is known [14] 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.

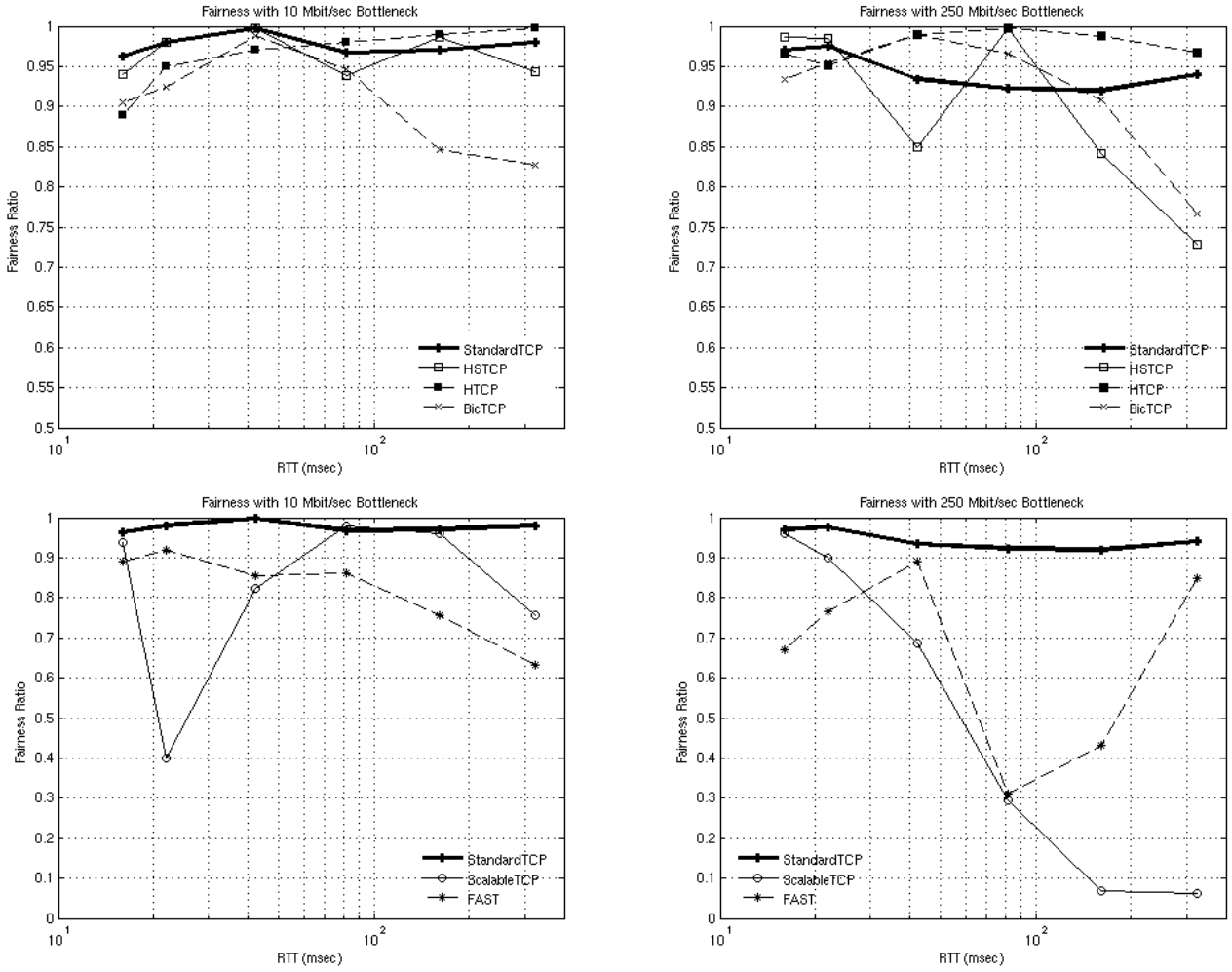

Fig. 4. Ratio of throughputs of two flows under symmetric conditions (same propagation delay, shared bottleneck link, same congestion control algorithm) as path propagation delay is varied. Results are shown for 10Mbit/sec and 250Mbit/sec bottleneck bandwidths. The bottleneck queue size is 20% BDP. Observe 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.

F. Convergence Time Test.

Figure 11 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 10Mb/s and 250Mb/s.

VII. Discussion

Two questions that we seek to answer in this paper are: (i) by using the performance of standard TCP as a baseline for evaluating new TCP proposals can we obtain meaningful results without the need to define absolute metrics (e.g. without adopting a specific fairness concept) and (ii) in view of the complexity of modern networks, do simple tests have any real value. We argue that our case study on high-speed protocols answers both questions positively.

A. Fairness

Perhaps the most striking results are obtained from the fairness test where the throughputs of competing flows with the same propagation delay are compared, see Figure 4. Under these conditions, the standard TCP congestion control algorithm consistently ensures that each flow achieves the same (to within less than 5%) average throughput. However, the measurements shown in Figure 4 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.* Figure 12 shows typical examples of measured $cwnd$ time histories. It can seen that the $cwnd$'s either do not converge to fairness or else converge

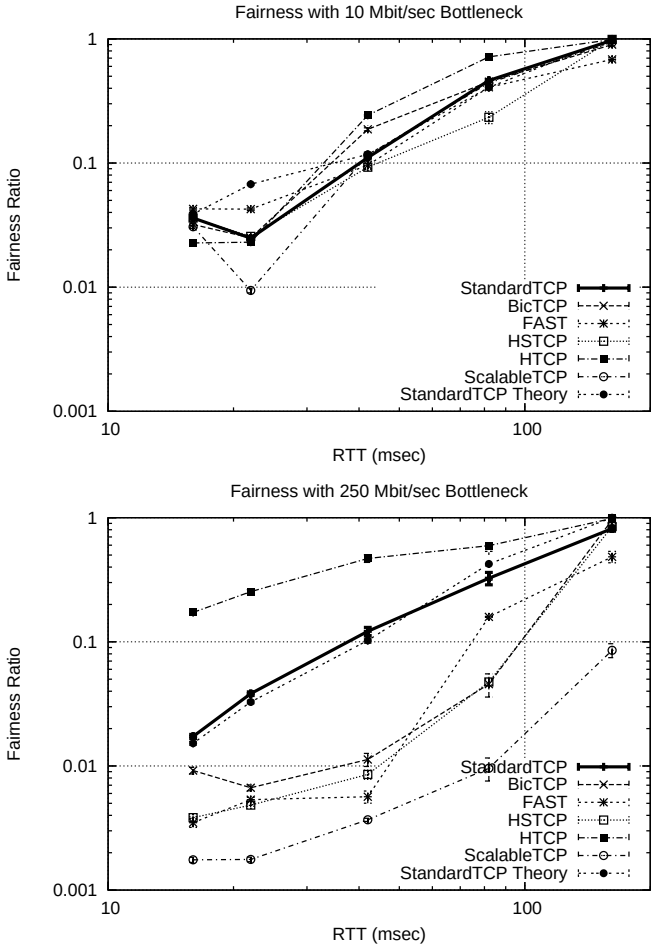

Fig. 5. Ratio of throughputs of two competing flows as the propagation delay of the second flow is varied. Results are shown for 10Mbit/sec (top) and 250Mbit/sec (bottom) bottleneck bandwidths. Flow 1 has RTT of 162ms, the RTT of Flow 2 is marked on the x-axis of the plots. Queue size is 20% BDP.

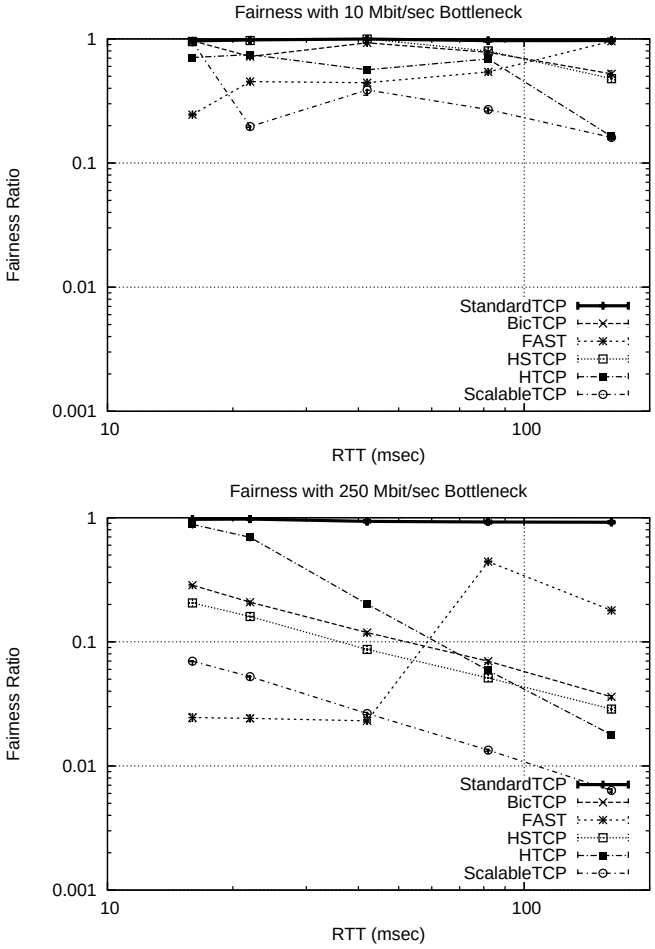

Fig. 6. Ratio of throughputs of competing New-TCP and standard TCP flows as path propagation delay is varied. Results are shown for 10Mbit/sec (top) and 250Mbit/sec (bottom) bottleneck bandwidths. Both flows have the same RTT. Queue size is 20% BDP.

very slowly indeed (not reaching fairness within the 10 minute 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. It has been known since the late 1980s [4] that in drop-tail networks such algorithms may not converge to fairness.

- *FAST-TCP.* Figure 13 shows typical examples of measured $cwnd$ time histories when using the FAST-TCP algorithm. The upper figure shows measurements taken on a 250Mb/s path with 42ms propagation delay. Rapid variations in $cwnd$ are evident which 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 162ms. 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 emphasising that these examples are representative of our measurements across a wide range of network conditions and are not selected as worst case behaviours. Our purpose in this paper is not to analyse or explain the FAST-TCP algorithm or its performance. We do, however, comment that the behaviour in the low latency example appears to be associated with use of a large value of $\bar{\alpha}$ leading to flooding of the queue and consequently generating many packet losses (e.g. see Figures 7 and 8), while the behaviour in the high-latency example appears to be associated with the adaptive switching of the $\bar{\alpha}$ parameter value.
- *HS-TCP.* Figure 14 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 [17]. Recall that the AIMD increase parameters are functions of $cwnd$ in HS-TCP. The slow convergence appears to originate in

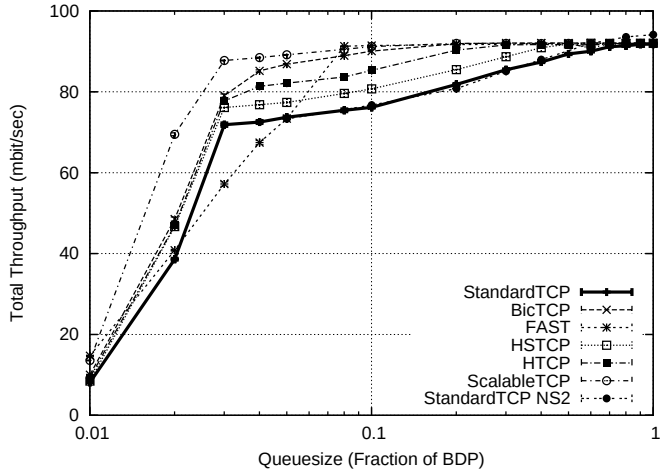

Fig. 7. Aggregate throughput of two competing TCP flows with 100Mbit/sec bottleneck bandwidth. Both flows have end-to-end round-trip propagation delays of 82ms. BDP is 683 packets.

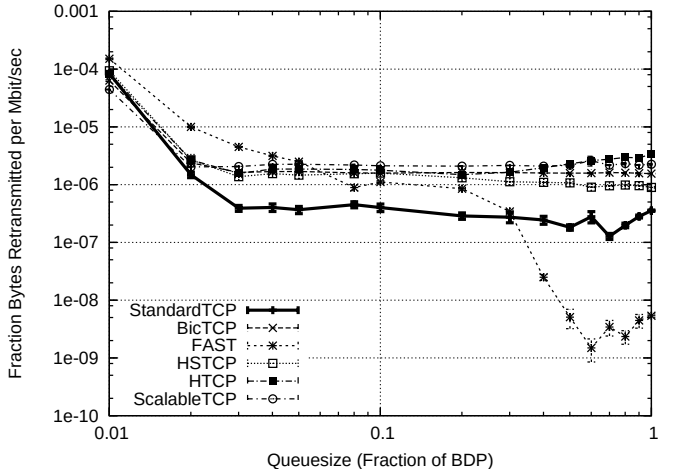

Fig. 8. Loss overhead versus queue size for two competing TCP flows. 100Mbit/sec bottleneck bandwidth, both flows have an 82ms propagation delay.

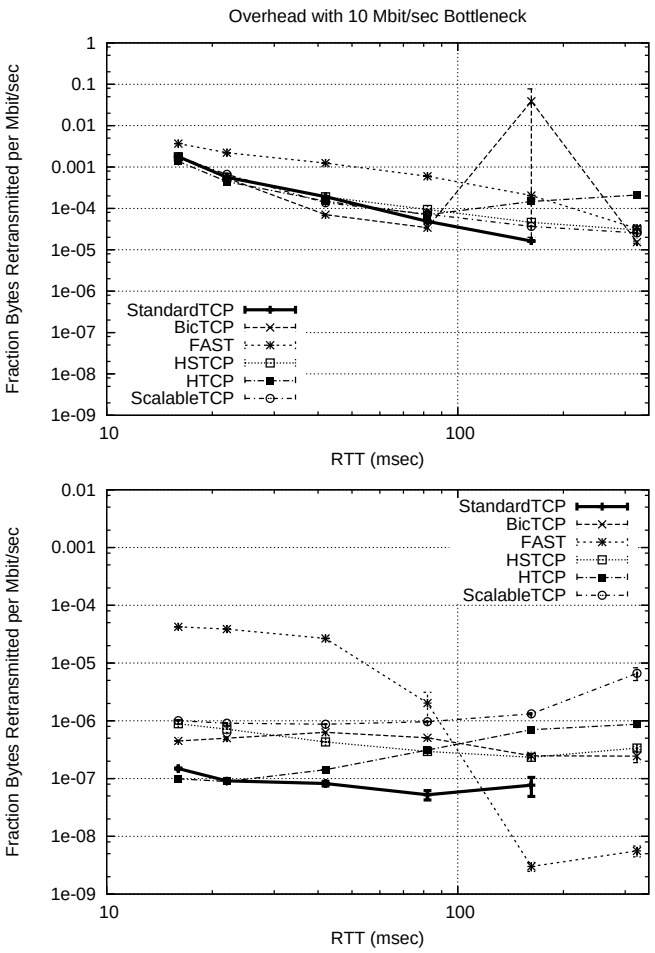

Fig. 9. Loss overhead versus propagation delay for two competing flows with 10Mbit/sec (top) and 250Mbit/sec (bottom) bottleneck bandwidth. Both flows have same RTT. Queue size is 20% BDP.

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 more aggressive values of increase and decrease parameters than do newly started flows which have small $cwnd$. Hence, sustained unfairness can occur. We also comment briefly upon the 250Mb/s, 42ms measurement for HS-TCP shown in Figure 4. The $cwnd$ time histories corresponding to this measurement are shown in Figure 14, and in more detail in Figure 15. 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 Figure 15 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, although this issue requires further investigation that is beyond the scope of the present paper.

- *BIC-TCP.* Figure 16 shows examples of the $cwnd$ time history of BIC-TCP following startup of a second flow. It can seen that as the path propagation delay increases the $cwnd$'s converge increasingly slowly, not reaching fairness within the 10 minute duration of these tests when the path propagation delay is large. This behaviour manifests itself in Figure 4 as a fall in the measured fairness as propagation delay increases.
- *HTCP.* Figure 17 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. RTT Unfairness

Considering now the situation when competing flows have different round-trip times, with the exception of H-TCP it can be seen from Figure 5 that all of the new proposals 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;

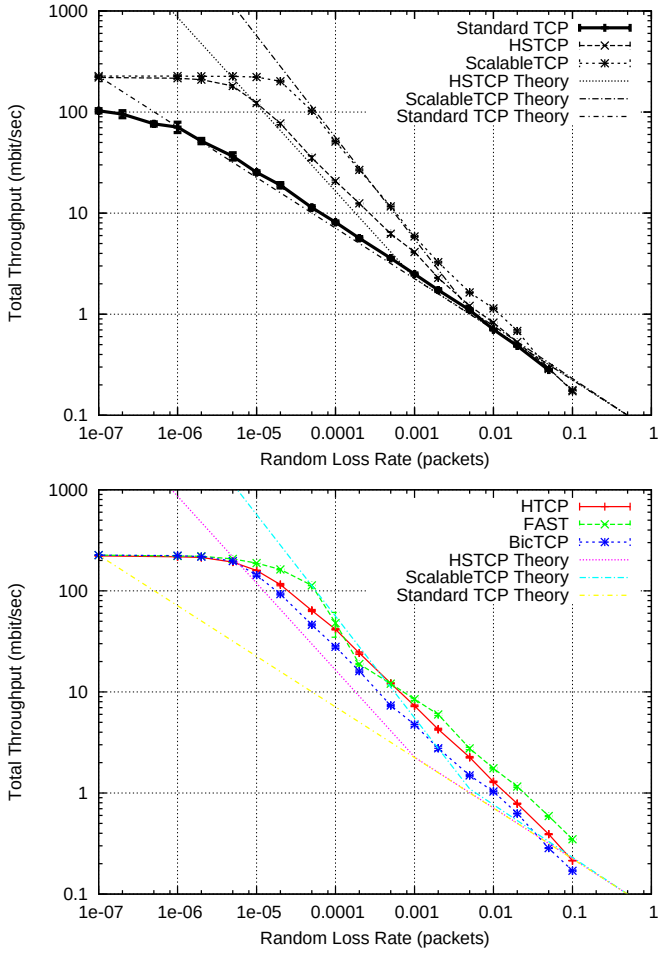

Fig. 10. Measured response functions with 250Mbit/sec bottleneck link and 162ms RTT.

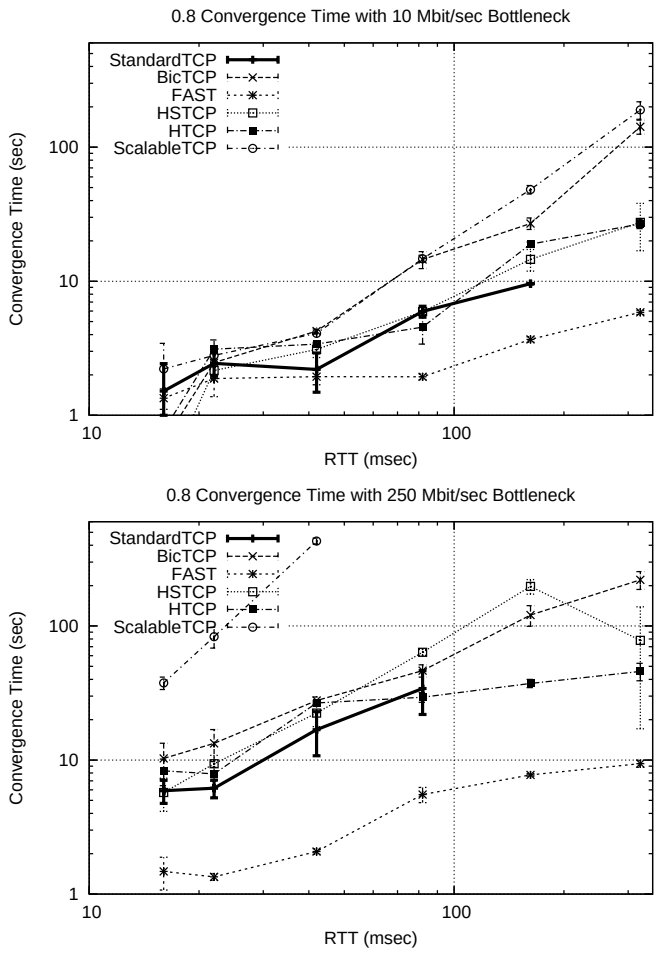

Fig. 11. 80% Mean 80% convergence time following startup of a second flow. Results are shown for 10Mbit/sec (top) and 250Mbit/sec (bottom) bottleneck bandwidths. Both flows have same RTT. Queue size is 20% BDP. Missing points along the ordinate axis indicate that the flows did not converge to within the 80% fairness ratio over the 10 minute duration of the test – this is especially evident with Scalable-TCP and standard TCP for long delays at 250Mb/sec.

for example, see Figure 18.

C. Backward Compatibility

Figure 6 shows the unfairness between new TCP proposals and standard TCP in low and high-speed network conditions. It can be seen that Scalable-TCP and FAST-TCP exhibit the greatest degree of unfairness in both low and high-speed conditions.

D. Efficiency

It can be seen from Figure 7 that for queue 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 queue size becomes less than about 2% of the bandwidth-delay product (or less than about 8% BDP in the case of FAST-TCP). Further investigation indicates that this is associated with micro-scale packet bursts (associated with ACK clocking and scheduling granularity within the end host operating systems) flooding the queue in this extreme operating regime. That packet bursts lead to a significant performance degradation in FAST-TCP is also independently noted in [19].

Figure 8 shows the associated packet loss overhead versus queue size, while Figure 9 shows the loss overhead versus path propagation delay. With the notable exception of FAST-TCP, it can be seen that the new protocols behave similarly and uniformly carry a greater overhead than standard TCP. The packet loss overhead of FAST-TCP is observed to be strongly dependent on queue provisioning: with small queues FAST-TCP has the largest overhead, while for large queues FAST-TCP has smallest overhead. On a 250Mb/s link, for propagation delays below 40ms, the overhead of FAST is more than two orders of magnitude greater than standard TCP and more than an order of magnitude greater than the other new protocols. Above 100ms, the overhead of FAST-TCP is an order of magnitude less than standard TCP.

E. Convergence Time

The mean convergence times following startup of a second TCP flow are shown in Figure 11. Results are shown for both 10Mb/s and 250Mb/s links, corresponding roughly to

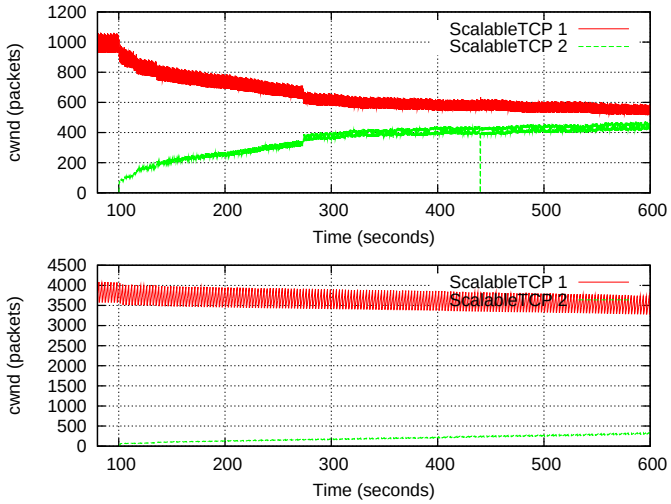

Fig. 12. Scalable-TCP *cwnd* time histories following startup of a second flow. RTT of both flows is 42ms (top) and 162ms (bottom). Bottleneck bandwidth is 250Mbit/sec, queue size 20% BDP.

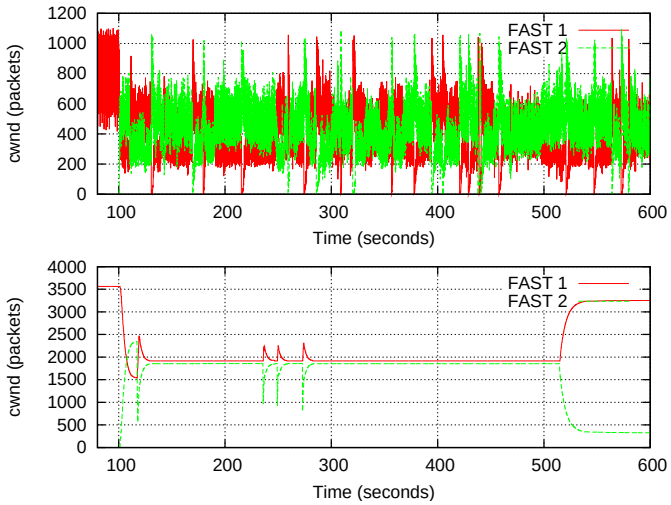

Fig. 13. FAST-TCP *cwnd* time histories following startup of a second flow. RTT is 42ms (top) and 162ms (bottom). Bottleneck bandwidth is 250Mbit/sec, queue size 20% BDP.

low and high speed conditions. We note that, in line with the previous discussion, that Scalable-TCP, HS-TCP and BIC-TCP all exhibit extremely slow convergence times (or, indeed, non-convergence). 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 30s. This behaviour is illustrated, for example, in Figure 17.
- *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 Figure 13 that while FAST may converge quickly initially, flows may later diverge again. It is important to emphasise that only the initial convergence time is captured by our convergence time metric.

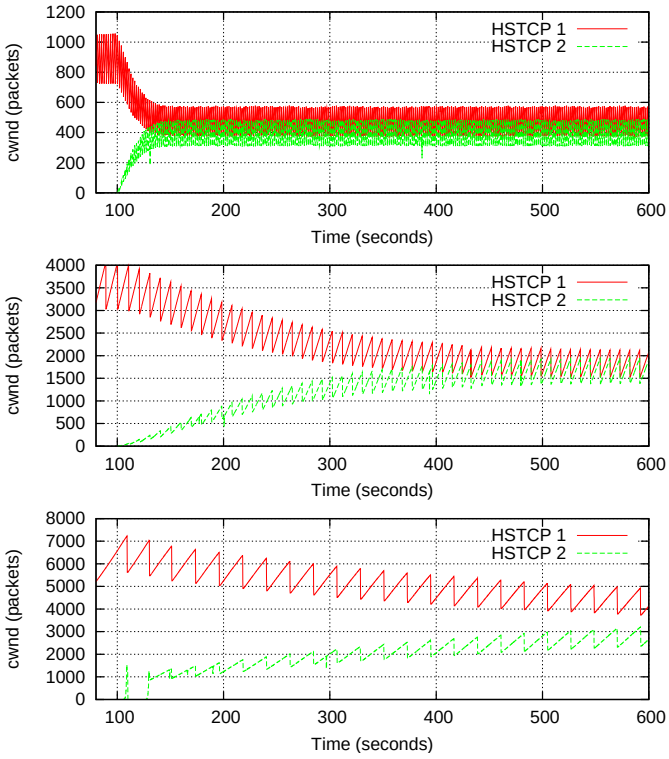

Fig. 14. HS-TCP *cwnd* time histories following startup of a second flow. RTT is 42ms (top), 162ms(middle) and 324ms (bottom). Bottleneck bandwidth 250Mbit/sec, queue size 20% BDP.

VIII. 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 [11], Kelly presents an experimental comparison of the aggregate throughput performance of Scalable-TCP and standard TCP. In [10], Low and co-authors present throughput and packet loss measurements from a lab-scale test network for FAST-TCP, HS-TCP, Scalable-TCP, BIC-TCP and TCP-Reno. Only aggregate throughput measurements are presented, thus preventing the fairness of the TCP algorithms from being evaluated; only a single queue size is used and network convergence time is not considered. In [9], aggregate throughput measurements are presented for FAST-TCP and TCP Reno. In [8], 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, but only for an 800Mb/s bottleneck bandwidth setting and 2000 packet queue. 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 thus present

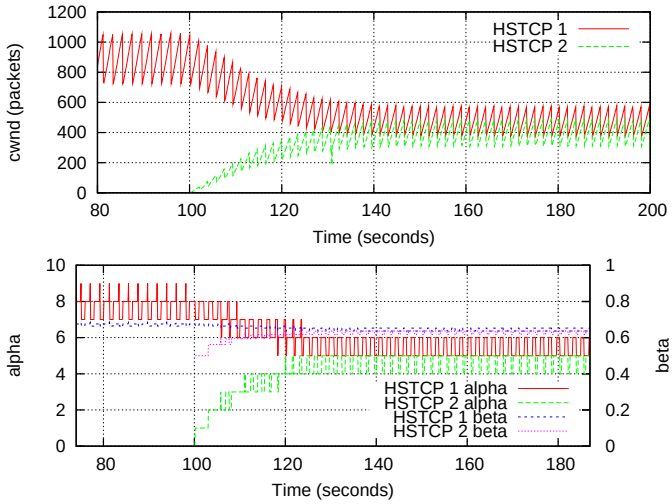

Fig. 15. Detailed HS-TCP $cwnd$ time histories (top) and α, β time histories (bottom) following startup of a second flow. RTT is 42ms, bottleneck bandwidth 250Mbit/sec, queue size 20% BDP.

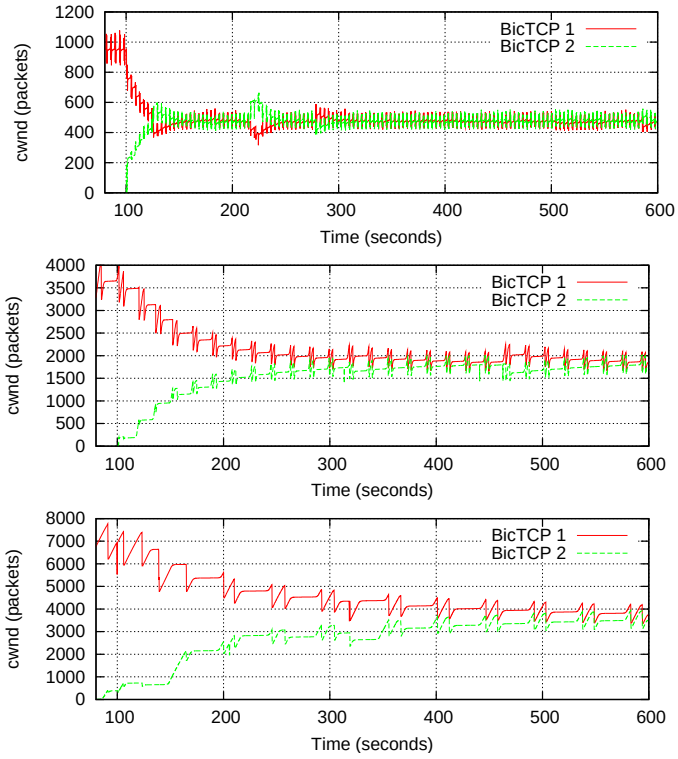

Fig. 16. BIC-TCP $cwnd$ time histories following startup of a second flow. RTT is 42ms (top), 162ms(middle) and 324ms (bottom). Bottleneck bandwidth is 250Mbit/sec, queue size 20% BDP.

performance measurements in support of these changes. While the design of congestion control algorithms for high-speed networks has been the subject of considerable interest, the evaluation of competing proposals per se has received far less attention. Notably, [3], [5] 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 U.S and between the U.S 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.

In all of the experimental tests noted above, no attempt is made to control for changes to the Linux network stack implementation that are unrelated to the congestion control algorithm.

IX. Summary and Conclusions

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 utilisation in a relatively static environment with long-lived flows, 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 ($> 100s$) 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.

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

References

[1] M.Allman, TCP Congestion Control with Appropriate Byte Counting (ABC). IETF RFC 3465, February 2003.
[2] G. Appenzeller, I. Keslassy, N. McKeown, Sizing router buffers. Proc. SIGCOMM 2004.
[3] H. Bullot, R.L. Cottrell, R. Hughes-Jones, Evaluation of Advanced TCP Stacks on Fast Long Distance Production Networks. J.Grid Comput, 2003.
[4] D.M. Chiu, R. Jain, Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. Computer Networks and ISDN Systems, 1989.
[5] R.L. Cottrell, S. Ansari, P. Khandpur, R. Gupta, R. Hughes-Jones, M. Chen, L. MacIntosh, F. Leers, Characterization and Evaluation of TCP and UDP-Based Transport On Real Networks. . Proc. 3rd Workshop on Protocols for Fast Long-distance Networks, Lyon, France, 2005.
[6] S.Floyd, K.Fall, Promoting the use of end-to-end congestion control in the internet. IEEE/ACM Transactions on Networking, August 1999
[7] S.Floyd, HighSpeed TCP for Large Congestion Windows . Sally Floyd. IETF RFC 3649, Experimental, Dec 2003.

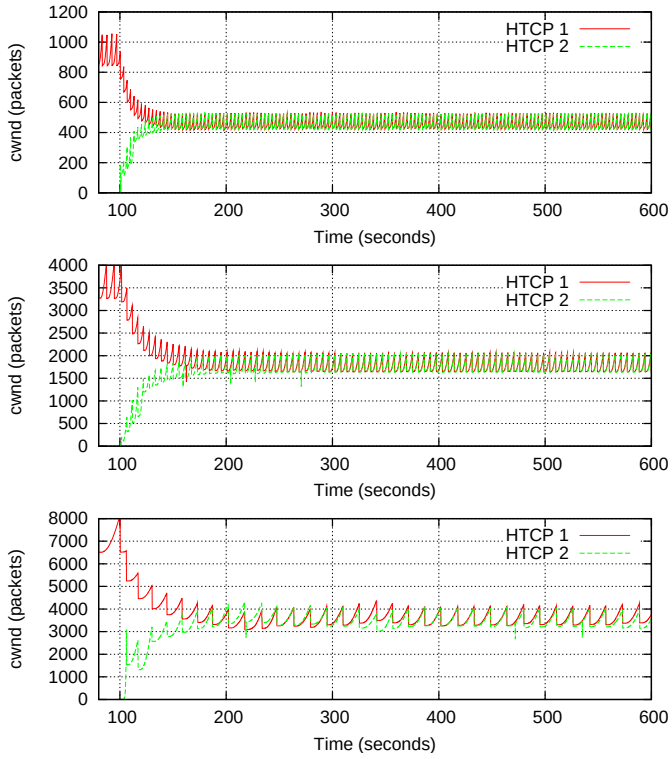

Fig. 17. H-TCP $cwnd$ time histories following startup of a second flow. RTT is 42ms (top), 162ms (middle) and 324ms (bottom). Bottleneck bandwidth is 250Mbit/sec, queue size 20% BDP.

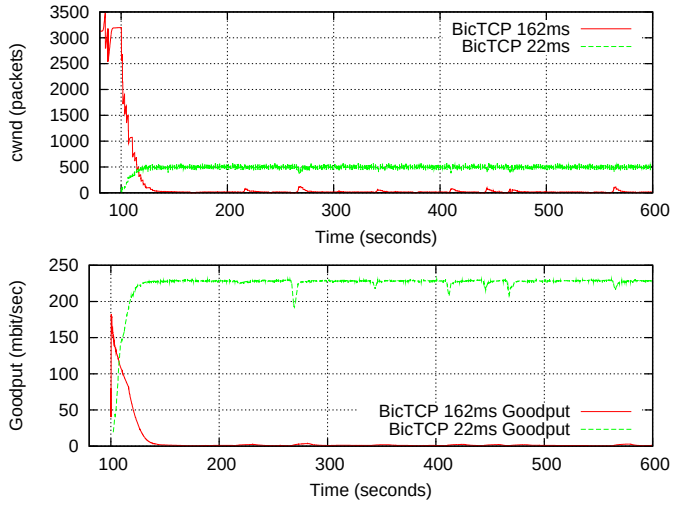

Fig. 18. BIC-TCP $cwnd$ and goodput time histories following startup of a second flow. RTT of first flow is 162ms and of second flow 22ms. Bottleneck bandwidth is 250Mbit/sec, queue size 20% BDP.

[8] C. Jin, D.X. Wei, S,H. Low, FAST TCP: motivation, architecture, algorithms, performance. Proc IEEE INFOCOM 2004.

[9] 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, S. Singh, FAST TCP: From Theory to Experiments. IEEE Network, 19(1):4-11, 2005

[10] S. Hegde, D. Lapsley, B. Wydrowski, J. Lindheim, D.Wei, C. Jin, S. Low, H. Newman, FAST TCP in High Speed Networks: An Experimental Study. Proc. GridNets, San Jose, 2004.

[11] T. Kelly, On engineering a stable and scalable TCP variant, Cambridge University Engineering Department Technical Report CUED/F-INFENG/TR.435, June 2002.

[12] D.J.Leith, Linux implementation issues in high-speed networks. Hamilton Institute Technical Report, 2003, `www.hamilton.ie/net/LinuxHighSpeed.pdf`.

[13] D.J.Leith, R.N.Shorten, H-TCP Protocol for High-Speed Long-Distance Networks. Proc. 2nd Workshop on Protocols for Fast Long Distance Networks. Argonne, Canada, 2004.

[14] Y.T.Li, D.J.Leith, BicTCP implementation in Linux kernels. Hamilton Institute Technical Report, 2004, `www.hamilton.ie/net/LinuxBicTCP.pdf`.

[15] M. Mathis, J Heffner and R Reddy, Web100: Extended TCP Instrumentation for Research, Education and Diagnosis. ACM Computer Communications Review, July 2003.

[16] J. Padhye, V. Firoiu, D.F. Towsley, J.F. Kurose, Modeling TCP Reno performance: a simple model and its empirical validation. Proc. SIGCOMM 1998 (Also IEEE/ACM Transactions on Networking, 2000).

[17] R.N.Shorten, D.J.Leith,J.Foy, R.Kilduff, Analysis and design of congestion control in synchronised communication networks. Automatica, 2004.

[18] R.N.Shorten, F. Wirth,F., D.J. Leith, A positive systems model of TCP-like congestion control: Asymptotic results. IEEE/ACM Trans Networking, to appear.

[19] D.X. Wei, S. Hegdesan, S.H. Low, A burstiness control for TCP, Proc. PFLDNET 2005, Lyon.

[20] L. Xu, K. Harfoush, I. Rhee, Binary Increase Congestion Control for Fast Long-Distance Networks. Proc. INFOCOM 2004

X. Appendix

	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 Gigabit Ethernet Controller
NIC Driver	e1000 5.2.39-k2
TX & RX Descriptors	4096

TABLE I
Hardware and Software Configuration.

The *altAIMD* kernel incorporates the following changes:

(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 have incorporated the implemented congestion control algorithms into a common network stack. A single sysctl is used to switch between congestion control algorithms on-the-fly, without the requirement of rebooting the machine.

(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 [12].* The current implementation of SACK processing in the Linux kernels requires a processing time which is $O(cwnd)$. This has serious performance implications when dealing with a large number of packets in flight which is common with large bandwidth-delay product paths. We have im-

TCP Protocol	Parameters
HS-TCP	High_P=1^{-7}, Low_Window=31 and High_Window=$83,000$
Scalable-TCP	$\alpha = 0.01$, $\beta = 0.875$ and Low_Window=16
H-TCP	$\Delta^L = 1sec$, $\Delta_B = 0.2$
BIC-TCP	$S_{max} = 32$, $B = 4$, $\sigma = 20$, $\beta = 0.8$ Low_Util=15%, Util_Check=2 and Low_Window=14
FAST-TCP	$\gamma = 50$, m0a=8, m1a=20, m2a=200 m0u=1500, m1l=1250, m1u=15000 and m2l=12500

TABLE II

DEFAULT NEW-TCP PARAMETERS USED IN ALL TESTS.

plemented a more robust algorithm with complexity of $O(\text{lost packets})$.

(iv) *Throttle Disabled [12].* 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 have modified this behaviour so that the ring buffers operate a pure drop-tail discipline.

(v) *Web100 [15].* The TCP stack was instrumented using Web100.

The various parameters associated with each high speed TCP algorithm that were used in the tests are shown in Table II. The values used are taken from the publicly released Linux patches by the relevant proposal authors implementing the TCP proposals.