Performance of Competing High-Speed TCP Flows

Michele C. Weigle, Pankaj Sharma, and Jesse R. Freeman, IV

Department of Computer Science, Clemson University, Clemson, SC 29634
{mweigle, pankajs, jessef}@cs.clemson.edu

Key words: High-speed TCP, congestion control, performance evaluation, network simulation

Abstract. The goal of recent high-speed TCP implementations is to allow scientists who have access to new high-speed networks to efficiently transfer large datasets to their remote colleagues. As of yet, there is no standard high-speed TCP. Because of this, scientists using one high-speed protocol may find themselves sharing a link with scientists using a different high-speed protocol. Previous work has evaluated such inter-protocol performance, but only with both flows starting at the same time – an unlikely situation. We perform an evaluation study using *ns*-2 to investigate the performance of competing high-speed TCP flows where one flow enters a network in which another high-speed flow has already reached its maximum data rate. The fairest result would be for the existing flow to cede half of its bandwidth to the new flow in order to allow both flows to evenly share the link. Our results show that in most cases this does not happen, but rather one high-speed flow dominates the other. Surprisingly, it is not always the existing flow that dominates.

1 Introduction

Recently, several new variants of TCP have been developed to take advantage of high capacity networks. It has been shown that Standard TCP, which handles most Internet traffic, has limitations when a single connection attempts to send data at very high speeds (*i.e.*, faster than 100 megabits per second) over long distances [6]. These new high-speed variants of TCP were designed to solve the limitations with high-speed transfers while maintaining reliability and fairness to Standard TCP flows. The most prominent of these are HighSpeed TCP (HS-TCP) [6, 7], Scalable TCP (S-TCP) [10], FAST TCP [8, 9], H-TCP [15], BIC-TCP [18], and CUBIC [13].

The target users of high-speed protocols are scientists who have access to fast, long-distance links that connect them to their colleagues in other locations. Distributed, collaborative applications for analyzing large data sets require a reliable and fast mechanism for distributing the data. Since the use of a dedicated high-speed link from one lab to another is prohibitively expensive, it is more likely that a network of connected research labs, much like the National LambdaRail project [1], will be developed. Scientists cannot rely on a single high-speed flow being allowed to consume the entire capacity of a link. The high-speed flow will likely have to share the capacity with other high-speed flows, as well as flows from low-speed applications such as web, email, and file sharing. Further, it may not be the case that a single high-speed technology is adopted by the entire community, but that several of these TCP variants will co-exist on the same links. For this reason, these new high-speed TCP implementations should be tested in an environment as close as possible to their likely real-world deployment.

Most previous work (including [2, 3, 16, 17]) has investigated either how these high-speed TCP implementations perform individually on dedicated links or how fairly they share link bandwidth with Standard TCP. Bullot *et al.* [3] determined that most of these protocols compete rather fairly against each other when two flows using different protocols share a single link, but they only tested the case where both flows started at the same time. Unlike Bullot *et al.*, we assume that one flow is sending at a high data rate before another high-speed flow enters the network. The ideal outcome should be for the existing flow to cede half of its bandwidth to the new flow so that both can fairly share the link.

We ran sets of experiments in the *ns*-2 network simulator [12] where we tested two competing high-speed TCP flows in the situation where one flow started well before the other flow. We studied all combinations of HS-TCP, S-TCP, FAST, H-TCP, BIC-TCP, and CUBIC in a network with a 622 Mbps (OC-12) bottleneck and a 100 ms RTT. We ran experiments with the maximum router queue buffer length at 100% of the bandwidth-delay product (BDP), 20% of the BDP, and 40 packets. We consider this to be the first step in a larger study of high-speed TCP protocols that also investigates the impact of other parameters such as RTTs, background traffic, reverse path traffic, and queuing mechanism.

We find that a 20% BDP router queue buffer results in high link utilization for these flows, intra-protocol fairness suffers when competing flows are started at different times, S-TCP is too aggressive in obtaining throughput from other high-speed flows, and in general, most of the high-speed protocols are not fair when competing with other high-speed protocols.

2 Background

All of the high-speed protocols that we evaluate attempt to be fair to Standard TCP flows that might be sharing the link. These protocols use Standard TCP when the TCP window w is less than a threshold value, and only use the high-speed version when w is above the threshold. Here we present a very brief overview of each of the protocols.

HS-TCP: When an acknowledgment (ACK) is received, HS-TCP increases w by $a(w)/w$. When one or more losses is detected in an RTT, HS-TCP sets w to $(1-b(w))w$. The goal is for a more aggressive increase and less aggressive decrease than Standard TCP in low-loss environments (*i.e*, environments where w is allowed to grow past the threshold, $LowWindow$). Current implementations of HS-TCP use a lookup table to determine the values of $a(w)$ and $b(w)$. Recommended settings allow $a(w)$ in the range of [1, 72] segments and $b(w)$ in the range [0.1, 0.5].

S-TCP: S-TCP is a simplification of HS-TCP, where the window adjustment functions $a(w)$ and $b(w)$ no longer depend on w. When the congestion window w is greater than $LowWindow$, S-TCP sets a to $0.01w$ (so that w increases by 0.01 for each returning

ACK) and b to 0.125. Like HS-TCP, when w is less than *LowWindow*, S-TCP behaves like Standard TCP.

FAST TCP: FAST TCP is a delay-based protocol that uses increasing RTTs as a form of congestion notification. In FAST, the congestion window w is updated every other RTT based on a function of the observed RTT and α, the number of segments that FAST attempts to keep in the network. If the RTT increases, FAST may decrease w even though loss has not occurred. Upon loss, FAST TCP behaves the same as Standard TCP (*i.e.,* it reduces w by half and enters TCP loss recovery).

H-TCP: H-TCP increases w based on Δ, the time between successive congestion events, and β, the ratio of the minimum-observed RTT to the maximum-observed RTT. H-TCP uses Δ^L (default of 1 second) as a threshold for entering high-speed mode. Upon detection of packet loss, $w = \beta w$, where $0.5 \leq \beta \leq 0.8$. These settings allow H-TCP to be *RTT-fair*, meaning that flows with longer RTTs will see similar throughput to flows with shorter RTTs. In addition, H-TCP flows in high-speed mode (where $\Delta > \Delta^L$) will cede some of their throughput to newer flows that have not yet reached high-speed mode.

BIC-TCP: Like H-TCP, BIC-TCP strives to maintain RTT-fairness. BIC-TCP sets a minimum window size w_{min}, maximum window size w_{max}, and a target window size w_{target}, which is the midpoint between w_{min} and w_{max}. BIC-TCP uses a binary search algorithm to reach the target window size (with a maximum increment of S_{max} segments in one step). When loss occurs, w_{max} is set to the current window size w, and w_{min} is set to the reduced window size $(1-\beta)w$, where $\beta = 0.125$. A new target w_{target} is then computed (as the midpoint between w_{min} and w_{max}). When the congestion window reaches the target without experiencing loss, the current window size becomes the new minimum ($w_{min} = w$) and a new target is computed.

CUBIC: CUBIC is a modification of BIC-TCP with the goal of improving on BIC-TCP's fairness. In CUBIC, the window increase is determined by a cubic function $w = C(t-K)^3 + W_{max}$, where C is a constant used for scaling, t is the time since the window was last reduced, W_{max} is the size of the window just before the window was last reduced, and $K = \sqrt[3]{W_{max}\beta/C}$, where β is a constant decrease factor. When a loss occurs, the window is reduced to $W_{max}\beta$, where $\beta = 0.8$.

3 Methodology

We ran all experiments in the *ns*-2 network simulator using the topology shown in Figure 1. Two senders are on the left side of the network, and two receivers are on the right side of the network. Each end node is connected to a router by a 1 Gbps link with a propagation delay of 1 ms. The two routers are connected by a 622 Mbps bottleneck link with a 48 ms propagation delay. This topology gives each sender a 100 ms round-trip time (RTT). The network has a bandwidth-delay product (BDP) of 7775 1000-byte segments, and drop-tail queuing is used at both routers. We performed the full set of

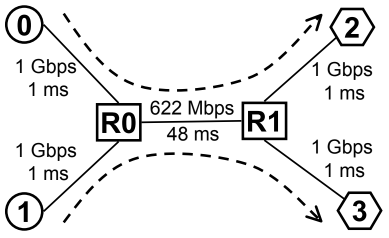

Fig. 1. Network Topology

experiments with three different router queue buffer lengths: 100% of the BDP, 20% of the BDP, and 40 segments. To ensure that TCP window size is not a limiting factor, each TCP connection has a maximum window size of 67,000 segments, which is about 64 MB. In each simulation, two connections are started, one from node 0 and one from node 1. The first connection (flow 1) is started at time 0, and the second connection (flow 2) is started 50 seconds later. Each simulation is run for 500 seconds.

Even though using the same RTT for both flows could cause synchronized loss, we were more concerned with factoring out the effects of RTTs on our results. Previous work [18] has shown that many of the high-speed protocols are not *RTT-fair*, meaning that flows with shorter RTTs achieve higher throughput than flows with longer RTTs. In order to concentrate on the effects of competing flows, we chose to equalize the RTTs.

We tested six high-speed protocols (HS-TCP, S-TCP, FAST, H-TCP, BIC-TCP, and CUBIC) by running a set of six experiments for each protocol and maximum router queue buffer size. Within each set, flow 1 uses the same protocol, and flow 2 uses a different one of the six protocols. For example, in the HS-TCP set, flow 1 is always HS-TCP and flow 2 is either HS-TCP, S-TCP, FAST, H-TCP, BIC-TCP, or CUBIC.

For each protocol, we used the parameters recommended by the protocol's authors. More details of the experimental setup we used (including *ns*-2 scripts) can be found at `http://www.cs.clemson.edu/~mweigle/research/hstcp/`.

We use the asymmetry metric from Bullot *et al.* [3] to evaluate the fairness of the various proposals. This metric, as opposed to the Chiu and Jain fairness index [4], provides information about which flow is more aggressive rather than just if the flows share the link fairly. The asymmetry metric is defined as $A = (\bar{x_1} - \bar{x_2})/(\bar{x_1} + \bar{x_2})$, where $\bar{x_i}$ is the average throughput obtained for flow i. Average throughput was measured starting at time 250 seconds to focus on steady-state throughput. The closer the asymmetry metric A is to 0, the more fair the distribution of throughput. The closer A is to 1, the more flow 1 dominates the transfer, and the closer A is to -1, the more flow 2 dominates.

4 Results

We found that with a router queue buffer length of 40 packets, utilization suffered, with many pairs of flows together obtaining less than 50% of the total link capacity. A queue buffer length of 100% BDP provides the best link utilization, but may be an unrealistic size for real networks. In our network, 100% BDP is 7775 packets, which is larger than the maximum queue size on many commercial routers.[1] With a 20% BDP queue buffer length, the maximum queue size was 1555 packets (in the range of commercial routers) and all experiments had a total link utilization of 99%-100%. With regard to fairness, the results obtained with the 20% BDP queue buffer length were slightly fairer in many cases than with either 100% BDP or 40 packets. For the remainder of the paper, we focus on results obtained with a queue buffer length of 20% BDP. We sorted the results of the experiments into five regions, based on the absolute value of *A* for the experiment (Figure 2). We will discuss the experiments that fell into each region separately. In general for all experiments, the behavior of the flows reached some steady state before the simulation ended. For each region, we will show graphs of the congestion window (in segments) for two representative experiments.

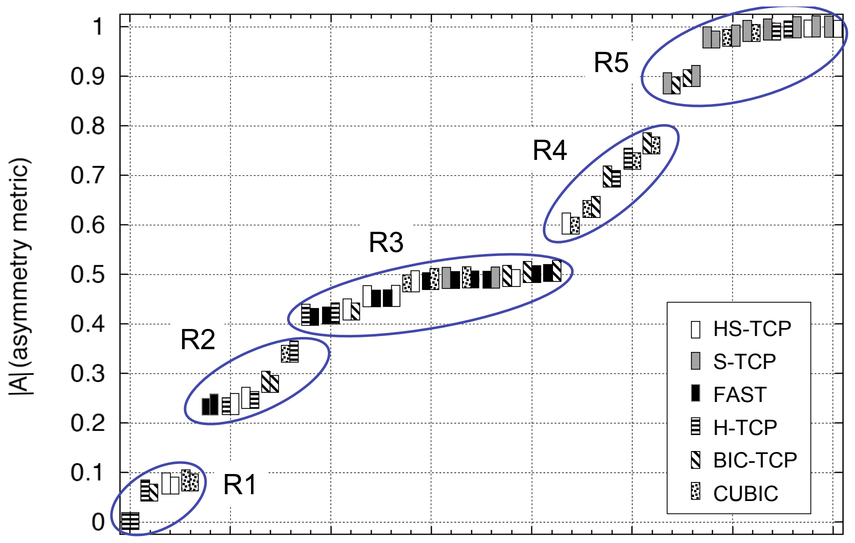

Fig. 2. We show the absolute value of the asymmetry metric. Each experiment is represented by two rectangles, in which the pattern on the left indicates flow 1's protocol and the pattern on the right indicates flow 2's protocol. The taller rectangle indicates the flow that received the larger share of the throughput. We divide the experiments into 5 regions based on fairness.

4.1 Region 1

In Figure 3, we show the congestion windows of two representative experiments from Region 1, which represents the fairest of the protocol pairings. Three out of the four pairings are intra-protocol (H-TCP, HS-TCP, and CUBIC), which is not surprising. Since both flows are running the same AIMD window adjustment algorithm, they should

[1] For example, Cisco routers typically have a default output queue size of 40 packets, with a maximum size of 4096 packets.

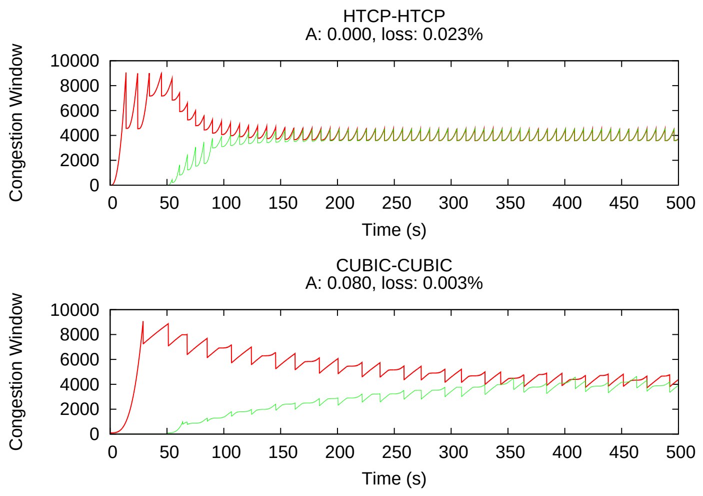

Fig. 3. Region 1: We show the congestion window for the H-TCP intra-protocol and CUBIC intra-protocol experiments.

eventually converge to a fair share no matter when the individual flows are started [4]. For the intra-protocol pairings that fall into Region 1, the two H-TCP flows are the fastest to converge - around time 150 seconds. Since we do not measure throughput for computing *A* until time 250, the H-TCP intra-protocol pairing has an *A* value of 0.0. The CUBIC intra-protocol flows do not converge until about 450 seconds, and the two HS-TCP flows do not converge until about 500 seconds.

The pairing of H-TCP and BIC-TCP when H-TCP starts first is the only inter-protocol experiment in Region 1. When the order of protocols is reversed though, the behavior is much less fair (falls into Region 4), with BIC-TCP controlling most of the throughput. H-TCP is able to cede some of the bandwidth when the BIC-TCP flow enters late, but when the roles are reversed, BIC-TCP does not cede a fair share of the bandwidth to the entering H-TCP flow. The behavior of both BIC-TCP and H-TCP is dependent upon competing traffic, especially if that traffic is causing the queue to overflow. Both of these protocols include a *time since last loss* factor in their window adjustment policies (BIC-TCP implicitly and H-TCP explicitly). These protocols will increase their aggressiveness the longer they go without experiencing a loss. This is in contrast to protocols like HS-TCP or S-TCP whose aggressiveness only depends on the current window size.

4.2 Region 2

Figure 4 shows the congestion windows for two representative pairings from Region 2. Both the FAST and BIC-TCP intra-protocol experiments fell into this region. Neither of these pairings converged by time 500, which is why these did not fall into Region 1 instead. All of the inter-protocol experiments in Region 2 involved H-TCP, including both pairings of H-TCP and HS-TCP. In these experiments, HS-TCP obtained more

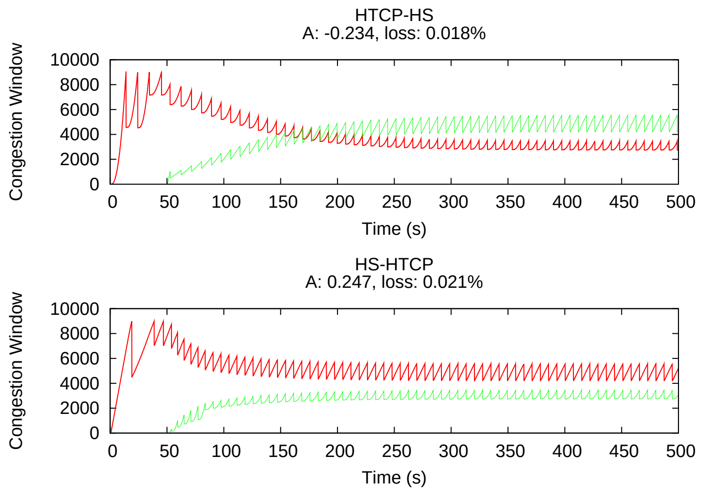

Fig. 4. Region 2: We show the congestion window for both pairings of H-TCP and HS-TCP. The top graph shows when HS-TCP is flow 1, and the bottom graph shows when H-TCP is flow 1.

throughput than H-TCP even when HS-TCP started later. With CUBIC and H-TCP, even though H-TCP started later, it gained higher throughput than CUBIC.

For the FAST intra-protocol experiments, when the flows start at different times, flow 1 occupies its share of the queue (according to α) and keeps the queue stable. When the second flow enters, its estimate of the minimum RTT is inaccurate because it sees the queuing delay caused by flow 1's packets as the minimum. With this underestimate of the actual queuing delay, flow 2 takes more than its fair share of the network resources. This problem does not occur when competing with other types of protocols because the other protocols drive the queue to overflow and drain completely.

4.3 Region 3

In Figure 5, we show the congestion windows of two representative experiments from Region 3. The experiments in this region all produced very similar results. Ten of the thirteen experiments involved FAST. Since FAST is a delay-based protocol, as the queuing delay increases, FAST adjusts its window either by increasing more slowly than before, or by decreasing, depending on the degree of the increase in the queuing delay. When competing against loss-based protocols that have to fill the queue to determine when available bandwidth is exhausted, FAST will see poorer performance than when competing against flows that are also sensitive to changes in the queuing delay (*i.e.*, another FAST flow). When we look at the queue size in experiments where the FAST flow starts first, the queue size is very stable – the queue neither fills nor drains. (This behavior, though is very dependent upon the size of the queue buffer and on FAST's α parameter.) Once the second flow enters, the queue becomes bursty. For all of these experiments, the FAST flow sees lower throughput than the other protocols. This is more

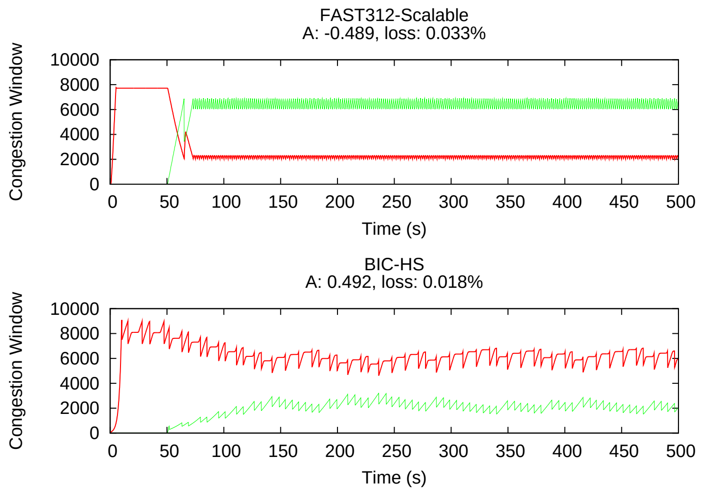

Fig. 5. Region 3: We show the congestion window for the pairings of FAST and S-TCP and of BIC-TCP and HS-TCP.

a characteristic of how FAST competes against loss-based protocols than the aggressiveness of the other protocols.

One interesting point about FAST is its competition with S-TCP. The experiments in which S-TCP competed with FAST fall into Region 3. All other experiments involving S-TCP fall into Region 5, where S-TCP is always the more aggressive flow. FAST keeps its share of packets in the queue even as S-TCP drives the queue to overflow. Additionally, since FAST backs off as the queuing delay increases, it is able to avoid much of the loss caused by S-TCP overflowing the queue. Once both flows are in steady-state, the queue never completely drains, resulting in very high link utilization.

Also in Region 3 are both pairings of BIC-TCP and HS-TCP. For these, flow 1 sees higher throughput regardless of the protocol. When competing against each other, both of these protocols are too aggressive in keeping bandwidth already obtained and not aggressive enough in grabbing its share of bandwidth from the existing flow.

4.4 Region 4

In Figure 6, we show the congestion windows of two representative experiments from Region 4. All but one of the pairings in Region 4 contain a CUBIC flow, and in all of these CUBIC gets less throughput than its competitor. Over all of the experiments, the only time that CUBIC saw higher throughput than its competitor was against FAST, but we have already mentioned that this is more due to FAST's behavior than CUBIC's. The designers of CUBIC consciously made the protocol behave less aggressively than BIC-TCP so that it would be fairer to competing flows. We see that two CUBIC flows share more fairly than two BIC-TCP flows, but CUBIC is not aggressive enough when competing against other protocols. One reason for the behavior we see might be due to the synchronized loss patterns that we get in a study such as ours without background traf-

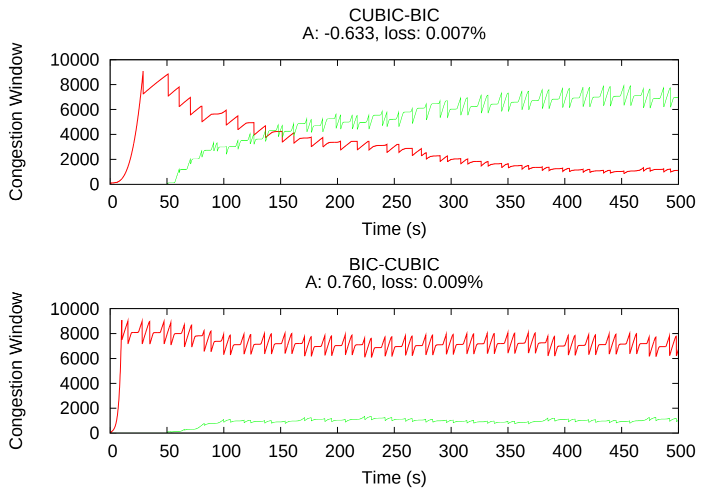

Fig. 6. Region 4: We show the congestion window for both pairings of CUBIC and BIC-TCP. The top graph shows CUBIC as flow 1, and the bottom graph shows when BIC-TCP is flow 1.

fic. Like BIC-TCP and H-TCP, CUBIC has an window increase function that depends upon the time elapsed between successive congestion events. If CUBIC can avoid some losses when the queue overflows (*i.e.*, the other flows see loss, but not CUBIC), then the CUBIC flow will be able to gain some of the available bandwidth released when the other flow backs off.

4.5 Region 5

All of the experiments in Region 5 contain at least one S-TCP flow, and the S-TCP flow always sees much higher throughput than the other flow. In Figure 7, we show how aggressive S-TCP is in obtaining and keeping bandwidth. The top graph shows S-TCP when competing against a BIC-TCP flow that is using the entire link before the S-TCP flow begins. Relatively quickly, S-TCP increases its window and pushes BIC-TCP down to very little throughput. In the bottom graph, S-TCP is competing against an H-TCP flow that enters late. The congestion window of the H-TCP flow is not visible on the graph because the S-TCP flow does not back off long enough after loss to allow the H-TCP flow to take advantage of the newly available bandwidth. S-TCP in essence is a MIMD (multiplicative increase, multiplicative decrease) protocol instead of the standard AIMD. Chiu and Jain [4] have shown that MIMD protocols do not converge to fairness. With S-TCP, the higher the window, the larger the amount of increase. So, when S-TCP does encounter loss, it is able to increase its window very quickly to take back the available bandwidth it had given up.

4.6 Flows Starting at the Same Time

We also ran experiments where we started both flow 1 and flow 2 at the same time. We found that intra-protocol results improved for all cases, with convergence times

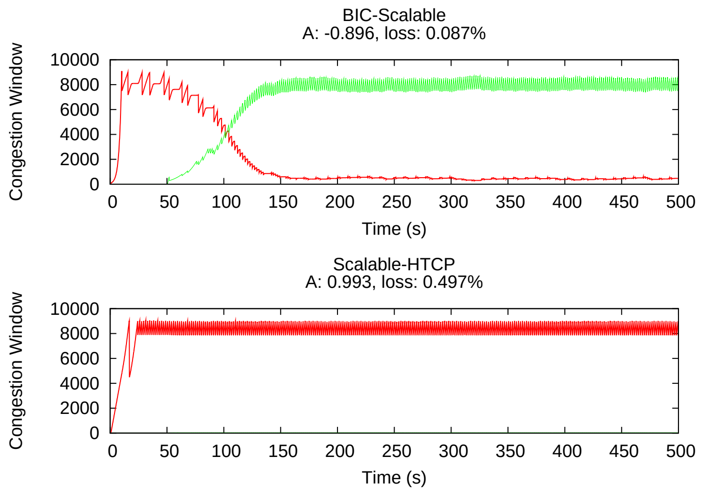

Fig. 7. Region 5: We show the congestion window for the pairings of BIC-TCP and S-TCP and of S-TCP and H-TCP. Note that on the bottom graph, the line for HTCP is not visible as its congestion window was very close to 0.

essentially going to 0. The improvement was most dramatic for S-TCP, shown in Figure 8. In the top graph, we see the same behavior as when S-TCP was competing with H-TCP in Figure 7. When the two S-TCP flows start at the same time (and since they have the same RTT), the windows match and the competition is fair. Once one of the S-TCP flows gains an advantage, it will keep increasing its advantage due to its MIMD window adjustment algorithm.

FAST also improves its intra-protocol fairness performance when both flows start at the same time. When discussing the intra-protocol FAST results in Region 2, we mentioned that the later-joining, second FAST flow has an inaccurate estimate of the minimum RTT. When both FAST flows start at the same time, they both have the same estimate of the minimum RTT (and thus, the queuing delay).

We found that flow start-time also had an effect on BIC-TCP. When BIC-TCP and either HS-TCP or H-TCP started at the same time, BIC-TCP increased its window aggressively and did not let the other flows share fairly. The behavior was very similar to results obtained when the BIC-TCP flow started first and either an HS-TCP or an H-TCP flow joined later.

For all of the other pairings, the start time did not seriously impact fairness. In general, protocols that were so unfair that they overtook the flow that started first were still unfair when the flows started at the same time.

5 Conclusions and Future Work

We studied the performance of two competing TCP flows in the scenario where one flow enters the network after the other flow has fully utilized the link. We evaluated

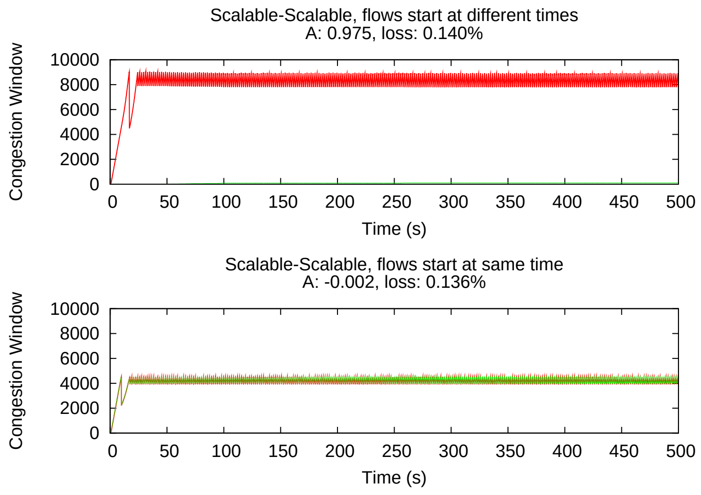

Fig. 8. We show the congestion window for two intra-protocol experiments with S-TCP. The top graph is when flow 1 starts 50 seconds before flow 2, and the bottom graph shows when both flows start at the same time. Note that on the top graph, the line for flow 2 is not visible as its congestion window was very close to 0.

the performance of six high-speed protocols using three different router queue buffer lengths. We concentrated on the results obtained with a queue buffer length of 20% BDP, which provided high link utilization and a realistic buffer size. We make the following findings:

– In general, most of the high-speed protocols are not fair when competing with other high-speed protocols.
– Intra-protocol fairness suffers when the flows are started at different times, due to slower convergence times.
– The performance of S-TCP and FAST do not depend upon the competing flow, but rather are dependent only upon their own operation.
– S-TCP is too aggressive in obtaining bandwidth, even when competing with another S-TCP flow.

As part of our future work, we would like to add background HTTP traffic and reverse path traffic to simulate typical non-high-speed Internet traffic that may be sharing the link with the high-speed flows. Also, we plan to study how using active queue management (AQM) techniques in these more realistic environments might affect the high-speed protocols. For this work, we plan to build upon previous studies of AQM and high-speed protocols [16, 17, 18].

Acknowledgements

We thank Injong Rhee and Lisong Xu for the *ns*-2 source code for S-TCP, BIC-TCP, and CUBIC and for the use of simulation scripts [14]. We thank Tony Cui and Lachlan

Andrew from the University of Melbourne's Centre for Ultra-Broadband Information Networks (CUBIN) for *ns*-2 source code for FAST [5], and we thank Douglas Leith *et al.* for the *ns*-2 source code for H-TCP [11].

References

[1] National LambdaRail Project. `http://www.nlr.net/`.

[2] A. Antony, J. Blom, C. de Laat, J. Lee, and W. Sjouw. Microscopic examination of TCP flows over transatlantic links. *iGrid 2002 Special Issue, Future Generation Systems*, 19(6), 2003.

[3] H. Bullot, R. L. Cottrell, and R. Hughes-Jones. Evaluation of advanced TCP stacks on fast long-distance production networks. *Journal of Grid Computing*, 1(4):345–359, 2003. Graphs available at `http://www-iepm.slac.stanford.edu/bw/tcp-eval/`.

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

[5] T. Cui and L. Andrew. FAST TCP simulator module for ns-2, version 1.1, 2004. Available at `http://www.cubinlab.ee.mu.oz.au/ns2fasttcp`.

[6] S. Floyd. Highspeed TCP for large congestion windows, Dec. 2003. RFC 3649, Experimental.

[7] S. Floyd, S. Ratnasamy, and S. Shenker. Modifying TCP's congestion control for high speeds, May 2002. Technical Note, available at `http://www.icir.org/floyd/papers/hstcp.pdf`.

[8] C. Jin, D. X. Wei, and S. H. Low. FAST TCP: motivation, architecture, algorithms, performance. In *Proceedings of IEEE INFOCOM*, Hong Kong, Mar. 2004.

[9] C. Jin, D. X. Wei, S. H. Low, G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, and S. Singh. FAST TCP: From theory to experiments. *IEEE Network*, 19(1):4–11, January/February 2005.

[10] T. Kelly. Scalable TCP: Improving performance in highspeed wide area networks. In *Proceedings of PFLDnet*, Geneva, Switzerland, Feb. 2003.

[11] D. Leigh, R. Shorten, et al. H-TCP ns-2 implementation, 2005. Available at `http://www.hamilton.ie/net/research.htm#software`.

[12] S. McCanne and S. Floyd. ns Network Simulator. Software available at `http://www.isi.edu/nsnam/ns/`.

[13] I. Rhee and L. Xu. CUBIC: A new TCP-friendly high-speed TCP variant. In *Proceedings of PFLDnet*, Lyon, France, Feb. 2005.

[14] I. Rhee and L. Xu. Simulation code and scripts for CUBIC, 2005. Available at `http://www.csc.ncsu.edu/faculty/rhee/export/bitcp/cubic-script/script.htm`.

[15] R. N. Shorten and D. J. Leith. H-TCP: TCP for high-speed and long-distance networks. In *Proceedings of PFLDnet*, Argonne, Illinois, Feb. 2004.

[16] E. Souza and D. A. Agarwal. A HighSpeed TCP study: Characteristics and deployment issues. Technical Report LBNL-53215, 2003. Available at `http://dsd.lbl.gov/~evandro/hstcp/hstcp-lbnl-53215.pdf`.

[17] K. Tokuda, G. Hasegawa, and M. Murata. Performance analysis of HighSpeed TCP and its improvements for high throughput and fairness against TCP Reno connections. In *High-Speed Networking Workshop (HSN)*, 2003.

[18] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion control for fast, long distance networks. In *Proceedings of IEEE INFOCOM*, Hong Kong, Mar. 2004.