# Linux beats Windows! – or the Worrying Evolution of TCP in Common Operating Systems[1]

Kashif Munir, Michael Welzl, Dragana Damjanovic
Institute of Computer Science, University of Innsbruck, Austria

Abstract—**For a long time, operating system designers have followed the recommendation of the IETF for TCP congestion control. Recently, this has changed: the Linux kernel uses BIC-TCP by default since June 2004, and Compound TCP is going to be a part of Microsoft Windows Vista, where it is likely to be enabled by default. This raises concerns about the future of congestion control in the Internet and the role of the IETF in this context. Simulation results indicate that there is indeed a reason to be concerned, as TCP flows of the above mentioned current and future operating systems appear to exhibit significant fairness problems when they share the same network.**

## I. INTRODUCTION

In the 1980s, Internet users experienced a phenomenon that was later referred to as a global "congestion collapse" - all of a sudden, end-to-end throughput was drastically reduced despite the continuous increase of connectivity. This problem was solved by introducing congestion control in TCP, and the change was documented in a SIGCOMM paper that became a seminal piece of Internet literature [1].

The increased use of multimedia in the Internet led to some concern about UDP based applications: with UDP, implementing congestion control is up to the programmer of the application, and this is a difficult task which will not necessarily lead to a performance improvement for the application itself. From the selfish perspective of a single sender and receiver, congestion control can sometimes even degrade the performance (in terms of raw throughput) perceived by an end system. On the other hand, it has been shown that even a single unresponsive flow can cause severe harm to a large number of responsive flows [2], and, because TCP plays a major role for the Internet, the term "TCP-friendliness" (also called "TCP-compatibility") was coined. In order to maintain the stability of the Internet, it was said that flows should be TCP-friendly, which means that, in steady state, they must not use more bandwidth than a conforming TCP running under comparable conditions [3].

This led the research community to focus on two distinct topics for a while:

1. TCP-friendly congestion control for multimedia applications, where a smoother rate is desirable
2. better-than-TCP congestion control, where the bandwidth is probed in a more aggressive fashion in order to make better use of links with a high bandwidth-delay product; the congestion control mechanisms in this category would not work well for the Internet because they would cause harm to competing TCP flows.

A large number of proposals were made in both fields for a couple of years. Regarding real use of these mechanisms in the Internet, there is reason to believe that we will see representatives of the first category in practice in the near future because the IETF has recently standardized the DCCP protocol; DCCP enables multimedia application programmers to make use of smooth yet TCP-friendly congestion control without having to implement this function within their applications [4].

As for the second category, the behavior of these mechanisms contradicts the very idea of TCP-friendliness, which has been regarded as the cornerstone of stability in the Internet for a long time. While we do not claim to know the exact historical progression of things, in our impression, it was HighSpeed TCP that opened the door to the possibility of *sometimes* being *somewhat* more aggressive than TCP. Consider this quote from the HighSpeed TCP specification, RFC 3649 [5]:

> *"Because HighSpeed TCP's modified response function would only take effect with higher congestion windows, HighSpeed TCP does not modify TCP behavior in environments with heavy congestion, and therefore does not introduce any new dangers of congestion collapse."*

It is this reasoning that was embedded in the design of BIC (Figure 8 in [6]) and Compound TCP ("By employing the delay-based component, CTCP can gracefully reduce the sending rate when the link is fully utilized. In this way, a CTCP flow will not cause more self-induced packet losses than a standard TCP flow, and therefore maintains fairness to other competing regular TCP flows.") [7], two of the three mechanisms that this paper focuses on. The third mechanism, CUBIC, the successor of BIC, implicitly includes this reasoning, as it was shown to be more TCP-friendly than BIC in all circumstances [8, 10].

## II. THE PROBLEM

Directly after the sentence quoted above, RFC 3649 continues as follows:

> *"However, if relative fairness between HighSpeed TCP connections is to be preserved, then in our view any*

[1] This work was funded by the Tyrolean Science Fund.

*modification to the TCP response function should be addressed in the IETF, rather than made as ad hoc decisions by individual implementors or TCP senders."*

The IETF failed to follow this advice in time. While TCP gradually evolved within the IETF, no drastic departure from its original congestion control mechanism was made. In the meantime, link capacities have grown, pronouncing the deficiencies of TCP and creating a demand for better congestion control. As a result, operating system developers have in the meantime made "ad hoc decisions": starting in kernel version 2.6.7, which was released in June 2004, BIC TCP was included in Linux, and enabled as the default mechanism - approximately three months after a major press release related to this mechanism. There are indications that upcoming kernels will use CUBIC, and Microsoft Windows Vista will use CTCP [7, 11].

Despite the fact that none of these mechanisms can be expected to endanger the stability of the Internet, the impact of one mechanism onto another is important even when there is no significant congestion, as it allows us to learn about the performance of real systems that are connected to the Internet today (or will be connected to it in the near future). In the next sections, we will therefore look at some simulations that involve BIC, CUBIC, CTCP and standard TCP. In order to give more meaning to these mechanisms, we would like to ask the reader to think "Linux" when reading "BIC", "possibly future Linux" when reading "CUBIC", "possibly Microsoft Windows Vista" when reading "CTCP", and think about any other operating system when reading "TCP". Note that our intention is to merely discuss the relative fairness of these mechanisms when they share a bottleneck; for an in-depth study of these and other congestion control schemes, please refer to [10, 12].

## III. SIMULATIONS AND DISCUSSION

Fig. 1 shows the dumbbell network configuration that is used for the simulation with ns-2 [13].

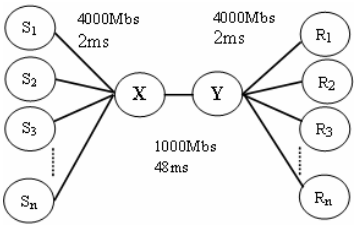


Fig. 1. *Dumbbell Topology*

The dumbbell topology is widely used for network simulations. Its aim is to represent a set of connections sharing the same bottleneck. We have used it to analyze how concurrent heterogeneous flows may affect the throughput of one another.

For the simulation the bottleneck capacity is 1Gbps and the bottleneck delay is set to 48ms. Drop Tail routers are used. The buffer size of the bottleneck link is set to 100% of Bandwidth-Delay Product. The packet size is set to 1500 bytes. The simulation runs for 1200 seconds. The ns-2 TCP-Linux implementation is used for the simulation [9].

Fig. 2 shows results of the NS simulation of 20 BIC, 20 CUBIC, 20 CTCP and 20 NewReno flows. We did not use TCP SACK because the Linux TCP implementation for ns-2 that we used did not include it, yet can be expected to more closely model the behavior of a real Linux implementation [9], and we believe that the difference between SACK and NewReno would not have a major impact on our results as multiple losses from a single sender window are unlikely in our simulation scenario.

We observe that the CWND curves get smooth after about 50 seconds of simulation. The BIC flows get a major part of the total bottleneck capacity by stealing it from CUBIC, CTCP and NewReno flows. The BIC flows stay aggressive throughout the simulation. We observe that BIC and CUBIC flows in the network immediately reduce the congestion windows of CTCP and NewReno flows. It is interesting to note that in the presence of BIC and CUBIC flows the CTCP flows do not behave any better than the standard TCP flows and in fact both CTCP and NewReno behave almost identically. We question whether this behavior will be acceptable for the future Internet with the release of Microsoft's Windows Vista which might be using CTCP when Linux is using BIC or CUBIC.

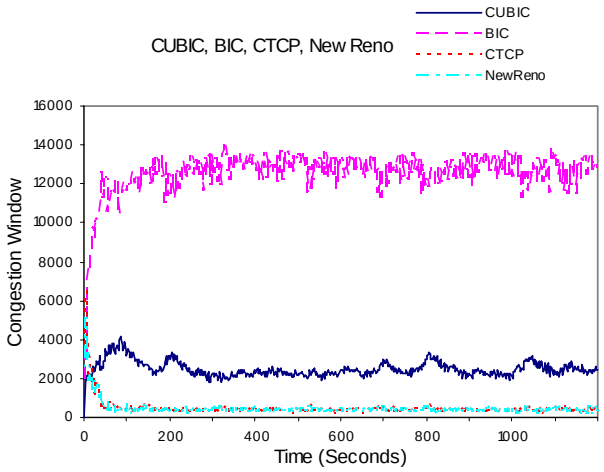


Fig. 2. *Congestion window curves for CUBIC, BIC, CTCP and NewReno flows. A congestion window curve represents the sum of 20 flows of a type*

Table 1 shows the average transfer rate as well as the average bottleneck link utilization per mechanism over the simulation time. From the numerical results it is clear that there are serious fairness problems due to the interaction of the heterogeneous flows. The BIC flows behave aggressively as they get a major share of the bottleneck capacity. As CUBIC is designed to be less aggressive and more TCP-friendly than BIC, the link utilization of CUBIC flows is almost five times less than BIC flows. The link capacity utilization of CTCP and NewReno flows is almost identical.

Table 1: Average transfer rates and average link utilization

| Mechanism | Average Transfer Rate (Mbs) | Average Link Utilization (%) |
|---|---|---|
| CUBIC | 151.11 | 15.11 |
| BIC | 764.42 | 76.42 |
| CTCP | 27.80 | 2.78 |
| NewReno | 26.59 | 2.66 |

## IV. RELATED WORK

Due to the wealth of in-depth performance studies of high-speed TCP mechanisms that were published, we shed some light on the "aggression" (the potential to adversely interact with other flows) of some of these mechanisms by providing an overview of related work instead of describing our own simulation results.

The studies that we considered generally fit in two categories. In one category, flows of each high-speed TCP were separately tested without mixing them up with flows of any other high-speed TCP variant. In the other category, flows of two high-speed TCP variants were tested against each other. In most cases of both categories flows were also tested with regular TCP flows. A dumbbell network topology was used for simulations.

### *A. One High-speed Mechanism*

In [14] experiments were done to measure the average throughputs as the propagation delay of the first flow was held constant and that of the second flow was varied from 16ms to 320ms. Measurements were taken for a range of link bandwidths and propagation delays of the first flow; the queue was sized as a constant proportion of the bandwidth-delay product. The measurements were restricted to long-lived TCP flows operating in drop-tail environments. An evaluation of the performance of STCP, HSTCP, BIC TCP, FAST TCP and HTCP was presented. The tests showed that STCP and FAST TCP exhibit substantial unfairness.

In [15] each experiment was run for 300 seconds. CTCP, HSTCP and Standard TCP were evaluated. Four regular TCP, four HSTCP flows and four CTCP flows were run separately. When packet loss was high (> 0.01), all three protocols behaved exactly the same. However, with the decrease of packet loss rate, HSTCP and CTCP could use the bandwidth more effectively. CTCP had slightly higher throughput than HSTCP. According to the authors, the reasons were twofold: 1) CTCP's response function is slightly more aggressive than HSTCP in a moderate window range; and 2) CTCP introduces much less self-induced loss due to its delay-based nature.

It was shown that CTCP and HSTCP could effectively recover from packet losses caused by burst background traffic and achieved high link utilization.

To check TCP friendliness, eight regular TCP flows were run as baseline. Then, four flows were replaced with the high speed protocols and the experiments were repeated under the same conditions. The throughputs of regular TCP flows were compared with and without the presence of high speed protocols. Bandwidth stolen from TCP flows was calculated in each experiment. The results showed that HSTCP caused around 60% throughput reduction of regular TCP, while the throughput reduction caused by CTCP was around 10%.

In [16] the performance of FAST TCP was compared with Reno, HSTCP, STCP, and BIC TCP using their default parameters. In the experiments the bottleneck capacity was 800Mbps and the maximum buffer size was 2000 packets. The results showed that FAST had the best performance of all the protocols for three evaluation criteria of fairness, stability and responsiveness. It had the second best overall throughput after BIC TCP. STCP showed worse intra protocol fairness compared with TCP Reno, while both BIC TCP and HSTCP achieved comparable intra-protocol fairness to Reno. HSTCP, BIC TCP and STCP showed increased oscillations compared with Reno and the oscillations became worse as the number of sources increased.

Different fairness issues were also seen in [17] where the effect of reverse traffic on various TCP stacks was studied. TCP NewReno, Sack, BIC TCP, HSTCP, HTCP, STCP, FAST TCP and Westwood+ TCP were analyzed. The bottleneck capacity was set to 250 Mbps. The bottleneck buffer was set to the bandwidth-delay product. The packet size was 1500 bytes. When the RTT was 40ms, and when reverse traffic was active, FAST TCP yielded the best link utilization of 90% whereas HTCP yielded the worst link utilization of 78%. When the RTT was 160ms, and when the reverse traffic was active, the highest link utilization was achieved by SACK TCP whereas FAST TCP achieved the lowest link utilization which was about 60% of the available bandwidth. Among other protocols, BIC obtained 92% link utilization, followed by STCP, Westwood+, HSTCP and HTCP in the order. It was mentioned that, except for FAST TCP, all new TCP mechanisms provided goodputs comparable to those of standard TCP SACK but with larger packet drop rates.

Another scenario with reverse traffic + background web traffic (with standard TCP) for 40 ms RTT and 160 ms RTT was studied. It was noticed that, in the presence of background Web traffic, the congestion windows of HTCP, HSTCP, BIC and STCP dropped to one several times because of timeout expirations. For an RTT of 160 ms, the congestion windows of the two forward connections of STCP were significantly different, which showed that the fairness was poor. SACK and Westwood+ were not affected by reverse traffic and timeout events were quite infrequent. FAST TCP flows were not affected by timeout events; however it was stated that they did not share the bandwidth fairly. For FAST, when reverse traffic was active, the two congestion windows converged to the same value and the congestion windows oscillated frequently.

In [18] the high-speed TCP variants BIC TCP, CUBIC, FAST, HSTCP, HTCP and STCP were evaluated in the presence of background traffic. The bandwidth of the bottleneck link was configured to be 400 Mbps. The buffer size of the bottleneck link was fixed to the maximum of 2 Mbytes. The long-lived and short-lived background flows were generated by TCP and their transmission rates were elastic to the amount of traffic in the network to the extent limited by the maximum allowed by 64KB receiver buffer size. When one high-speed TCP flow was run with one flow of TCP-SACK both having same RTTs, then even with background traffic the utilization of HTCP, FAST and STCP with RTTs of 160 and 324ms was significantly lower than that of the other protocols. It was mentioned that the problem was related to the inherent protocol behavior of HTCP in the way that HTCP ties its window reduction to the estimated buffer size of the network. The results showed STCP as simply too aggressive. For FAST, its behavior became less predictable in the presence of background traffic due to noise in the RTT estimation.

For TCP friendliness experiments were run with one high-speed TCP flow and one regular TCP flow with the same RTT (varied from 16ms to 324ms) over the same bottleneck link with and without background traffic. TCP-friendliness was measured with Jain's fairness index [19]. It

was observed that HTCP had the best TCP-friendliness in very low RTT networks with or without background traffic. However, as RTT increased beyond 16ms, HTCP's fairness to TCP dropped rapidly in both cases. In general it was noted that all TCP variants except FAST improved their TCP-friendliness when background traffic was added to the experiments. This was mostly because of two reasons. First, increased background traffic took away bandwidth from high speed TCP variants, so they became less aggressive as their average window sizes became smaller than without background traffic. The other reason was that with background traffic, randomness in packet losses increased. HSTCP, CUBIC and BIC improved their TCP fairness quite considerably with background traffic. It was observed that FAST showed the best TCP friendliness in high RTT networks.

In [20] the buffer size was set equal to 200-500 packets as are suggested values for buffers in Cisco Systems router [21]. According to the authors TCP-Libra showed more friendliness to TCP NewReno than Vegas, Cubic, and Hybla coexisting with TCP NewReno.

In [22] the Performance of STCP, HSTCP, BIC TCP, FAST TCP and HTCP was evaluated. The ratio of throughputs of two flows under symmetric conditions (same propagation delay, shared bottleneck link, same congestion control algorithm) was measured, as the path propagation delay was varied from 16ms to 320ms. Results were shown for 10 Mbps and 250 Mbps bottleneck bandwidths. The bottleneck queue size was 20% BDP. It was observed that, while Standard TCP and HTCP were essentially fair (the competing flows achieve, to within 5%, the same average throughput) under these conditions, STCP and FAST TCP were notably unfair. HSTCP and BIC TCP were also mentioned to exhibit significant unfairness but to a smaller degree than Scalable-TCP and FAST TCP.

### *B. Mixed High-speed Mechanisms*

In [23] two different protocols were tested at a time and the need of a standard for high-speed TCPs was stressed. An evaluation study of competing high-speed TCP flows was done where one flow entered a network in which another high-speed flow had already reached its maximum data rate. The fairest result would be for the existing flow to give half of its bandwidth to the new flow in order to allow both flows to evenly share the link. The results showed that in most cases this did not happen, but rather one high-speed flow dominated the other. HSTCP, STCP, FAST, HTCP, BIC TCP and CUBIC were studied in a network with a 622 Mbps bottleneck and 100 ms RTT. The maximum router queue buffer length of 100% of the BDP, 20% of the BDP and 40 packets were used.

For a 20% BDP router queue buffer, Intra protocol fairness suffered when competing flows were started at different times. STCP was too aggressive in obtaining throughput from other high-speed flows and most of the high-speed protocols were not fair when competing with other high-speed protocols. To ensure that the TCP window size was not a limiting factor, each TCP connection had a maximum window size of 67000 segments, which was about 64 MB. Each simulation was run for 500 seconds. For each protocol a set of six experiments was run with a maximum queue buffer size. Within each set, flow 1 used the same protocol and flow 2 used a different one of the six protocols. 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 provided the best link utilization, but may be realistic for real networks. With a 20% BDP queue buffer length, all experiments had a total link utilization of 99%-100%.

For the remainder of the experiments a queue buffer length of 20% BDP was used. Of all the results, the pairing of HTCP and BIC TCP when HTCP started first gave the best fairness results. When the order of the protocols was reversed, the behavior was much less fair, with BIC TCP controlling most of the throughput. 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. Of all the experiments, the experiments involving one STCP flow showed very poor fairness results. It was shown that STCP was quite aggressive in obtaining and keeping bandwidth even when competing with another STCP flow. FAST improved its intra-protocol fairness performance when both flows started at the same time. The starting time of flows also had an affect on BIC TCP. When BIC TCP and either HSTCP or HTCP started at the same time, BIC TCP increased its window aggressively and did not let other flows fairly share the bandwidth.

In [24] only two different protocols were tested at a time. In the experiments flows were started at the same time. New Reno TCP was compared with Parallel TCP Reno (P-TCP), STCP, FAST TCP, HSTCP, HighSpeed TCP Low Priority (HSTCP-LP), BIC TCP and HTCP. The analysis compared and reported on the stacks in terms of achievable throughput, impact on RTT, intra and inter-protocol fairness and stability. Experiments were done for 20 minutes for small RTT network of 10 ms, medium RTT network of 70 ms and high RTT network of 170 seconds. The bottleneck capacity for most of the tests was set to 622 Mbps. For inter-protocol fairness, two different flows were sent on the link from two different machines. On the short RTT network, it was shown that P-TCP behaved very aggressive. Only BIC TCP was sufficiently aggressive to compete with P-TCP, but it appeared too aggressive for the other protocols. The results showed that STCP, which was very aggressive on the short RTT network, became quite gentle on the high RTT network. On the other hand, HTCP, which was gentle in the small and the medium RTT networks, became aggressive for the high RTT network. HSTCP was too gentle in the tests.

In [25] High-speed transport protocols – HSTCP, STCP, FAST, CUBIC, HTCP and UDT – were also evaluated by mixing two of the mechanisms at a time.

Bottleneck bandwidth of 1 Gbps was used. The default value of 127 packets was mainly used as the output buffer size. A large buffer size of 512 packets was also used to see the impact of buffer size. Every high speed transport protocol flow achieved a high throughput near to 1 Gbps.

In an experiment using buffer size of 127 packets and then buffer size of 512 packets, a large difference in their throughput characteristics of FAST and Scalable TCP flows was observed. The throughput of FAST flow was greatly improved by increasing the buffer size at the edge routers larger. When the buffer size was set to 127 packets, losses

of packets were observed with FAST, while no packet losses occurred with FAST when the buffer size was 512 packets. On the other hand, the throughput of Scalable TCP seemed unstable with a larger buffer size in edge routers. In one experiment Linux was adopted as the sender side OS and FreeBSD (ver.5.3) and Windows XP (SP2) were adopted as receiver side OS. It was observed that all protocols (except for standard TCP) achieved high throughput regardless of the kind of receiver side OS. That is the majority of users on the Internet were ready to fill up the bandwidth up to 1 Gbps as receivers if the sender had employed such the high-speed protocols. It was mentioned that while every high-speed transport protocol flow eventually achieved a high throughput near to 1 Gbps, the throughput in the case of the XP receiver increased more slowly compared with the cases of Linux and FreeBSD receivers. In addition, Standard TCP flows in the case of XP achieved higher throughput than in the cases of Linux and FreeBSD.

Co-existence of a high-speed transport protocol and a long-lived Standard TCP was also examined in [25]. It was observed that a high speed transport protocol flow starved the long-lived Standard TCP flow, and the performance of the high speed transport protocol was merely degraded. An experiment was done to examine the throughput characteristics when two of the high-speed transport protocol flows co-existed in the path. It was found that the link utilization degraded when different kinds of high-speed transport protocols flows coexisted.

Experiments were also done to examine the co-existence of short-lived TCP flows with a high-speed transport protocol flow. The performance of the high-speed transport protocol flow was considerably affected by the coexisting short-lived TCP flows, even though the amount of these flows was small. The larger the averaged file size of the short-lived TCP flow was, the larger the observed damage in the high-speed transport protocol flow became, except for UDT. The UDT flow was also affected by coexisting short-lived TCP flow, but its degree of degradation was relatively small. The results indicated that coexisting short-lived standard TCP flows could considerably damage the performance of high-speed TCP-based transport protocol flows although coexisting long-lived standard TCP flows could not so. Setting the buffer size at the bottleneck nodes larger could decrease the degradation of throughput of both high-speed TCP and short-lived TCP flows.

## V. Conclusion

Our simulation work indicates that, in the regime of little congestion, Linux (with kernel version 2.6.7 or higher) will perform better than Microsoft Windows if systems of both type share the same network. This is not to say that CTCP could not outperform BIC or CUBIC in isolation in some circumstances – it is the mixture that we are worried about, and that this paper addresses. Our review of related work has shown that this problem does not only concern BIC, CUBIC, CTCP and standard TCP – numerous fairness issues exist among many other high-speed TCP variants.

These results leave us with a number of open questions regarding the practical application of congestion control in current operating systems. For instance, should Linux make the choice of congestion control application dependent by using BIC for transfers of large files only and CUBIC for everything else? Will the more aggressive behavior (which translates into better performance for the user) of BIC and CUBIC (as opposed to standard TCP and CTCP) cause more users to switch from Microsoft Windows to Linux? Will people be disappointed with the network performance of Microsoft Windows Vista as they compare these systems in the same network and at the same time? And, most disturbingly, could such a development lead to some sort of an “arms race” for more aggressive congestion control mechanisms in the long run? We have seen that there is still headroom: STCP is an example of a mechanism which is more aggressive than most other protocols in many cases.

We believe that these are serious concerns, and that it is high time for the IETF to act. A new congestion control framework should be specified, and this framework should be beneficial for the Internet as a whole, and in line with incentives of users – that is, it should be advantageous for a single end user as well as an operating system designer to follow the recommendation. To this end, we believe that the new framework could either follow the conservative path of CTCP (which may be an overly prudent decision, as single users will see better performance with a more aggressive scheme when traffic is mixed) or specify a congestion control behavior that is more aggressive than standard TCP, yet designed in a way that preserves the stability of the Internet.

In our opinion, this new framework should replace the notion of TCP-friendliness. But what could such a framework look like? Could the recommendation simply be that (and note that this is a tongue-in-cheek statement) new mechanisms should, for instance, be “CUBIC-friendly”?

## References


[1] Jacobson, V., Karels, M. J., “Congestion Avoidance and Control”, In Proceedings of ACM SIGCOMM '88, Stanford, CA, August 1988.

[2] Sally Floyd, Kevin R. Fall, “Promoting the use of end-to-end congestion control in the internet,” IEEE/ACM Transactions on Networking, 7(4):458--472, 1999.

[3] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, L. Zhang, “Recommendations on Queue Management and Congestion Avoidance in the Internet,” April 1998, RFC 2309.

[4] E. Kohler, M. Handley, S. Floyd, “Datagram Congestion Control Protocol (DCCP),” March 2006, RFC 4340.

[5] Sally Floyd, “HighSpeed TCP for Large Congestion Windows,” December 2003. RFC 3649.

[6] L. Xu, K. Harfoush, and I. Rhee, “Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks,” In Proceedings of IEEE INFOCOM 2004, March 2004

[7] K. Tan, J. Song, Q. Zhang, and M. Sridharan, “Compound TCP: A Scalable and TCP-friendly Congestion Control for High-speed Networks”, in 4th International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet), 2006, Nara, Japan.

[8] I. Rhee, L. Xu, “CUBIC: A New TCP-Friendly High-Speed TCP Variant, “In Proceedings of Third International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet), February 2005, Lyon, France.

[9] David X. Wei and Pei Cao, “NS-2 TCP-Linux: An NS-2 TCP Implementation with Congestion Control Algorithms from Linux”, to appear in Proceedings of ValueTool'06 -- Workshop of NS-2, Oct, 2006

[10] Sangtae Ha, Long Le, Injong Rhee and Lisong Xu, “A Step Toward Realistic Evaluation of High-Speed TCP Variants,” in The 4th International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet), February 2006, Nara Japan.

[11] http://www.microsoft.com/technet/community/columns/cableguy/cg1105.mspx#EAD
[12] Li, Y.T., Leith, D. J., Even, B., “Evaluating the Performance of TCP Stacks for High-Speed Networks”, Proc. PFLDnet, Feb. 2006, Nara, Japan.
[13] The Network Simulator – ns-2. *URL: http://www.isi.edu/nsnam/ns/.*
[14] Yee-Ting Li, Douglas Leith, and Robert Shorten, “Experimental Evaluation of TCP Protocols for High-Speed Networks”, Technical report, Hamilton Institute, 2005.
[15] Kun Tan, Jingmin Song, Qian Zhang, “A Compound TCP Approach for Fast Long Distance Networks”, Microsoft Technical Report, 2005.
[16] C. Jin, D. Wei, S. Low, “FAST TCP: Motivation, Architecture, Algorithms, Performance”, In Proc IEEE Infocom 2004.
[17] S. Mascolo, F. Vacirca, "The effect of reverse traffic on TCP congestion control algorithms," in Proc. of Protocols for Fast Long-Distance Networks, Feb. 2006, Nara, Japan.
[18] Sangtae Ha, Yusung Kim, Long Le, Injong Rhee, Lisong Xu, “A Step toward Realistic Performance Evaluation of High-Speed TCP Variants”, PFLDnet 2006, Nara, Japan.
[19] D. Chiu and R. Jain. “Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks.” Journal of Computer Networks and ISDN, 17(1):1-14, 1989.
[20] G. Marfia, C. Palazzi, G. Pau, M. Gerla, M. Sanadidi and M. Roccetti, “TCP-Libra: Exploring RTT Fairness for TCP”, IEEE Journal on Selected Areas in Communications Special Issue on Non Linear Optimization of Communication Systems, Sept. 2005. A.K.A. UCLA Computer Science Department Technical Report # UCLA-CSD TR-050037
[21] C.S. Inc Buffer tuning for all Cisco routers – documented id: 15091. Available: http://www.cisco.com/warp/public/63/buffertuning.html.
[22] B.Even, Y.Li, D.J.Leith, “Evaluating the Performance of TCP Stacks for High-Speed Networks”, PFLDnet 2006, Nara, Japan.
[23] Michele C. Weigle, Pankaj Sharma, Jesse R. Freeman IV, “Performance of Competing High-Speed TCP Flows”, Networking 2006: 476-487.
[24] Hadrien Bullot, R. Les Cottrell, Richard Hughes-Jones, “Evaluation of Advanced TCP Stacks on Fast Long-Distance Production Networks”, J. Grid Comput. 1(4): 345-359 (2003).
[25] K. Kumazoe, K. Kouyama, Y. Hori, M. Tsuru, Y. Oie, “Can high-speed transport protocols be deployed on the Internet? : Evaluation through experiments on JGNII”, PFLDnet 2006, Nara, Japan.