Experimental Evaluation of TCP Implementations on Linux/Windows Platforms

Yue Zhou
School of Information and Engineering
Communication University of China
Beijing, China 100024
Email: masonzhy@gmail.com

Jinyao Yan
Computer and Network Center
Communication University of China
Beijing, China 100024
Email: jyan@cuc.edu.cn

***Abstract*—Although new TCP congestion control algorithms have been proposed for high-speed network in recent years, standard TCP is still the most widely used on the Internet. The performance of applications strongly depends on the behavior of TCP, especially the TCP congestion control algorithms. As different operating systems may implement standard TCP differently in detail, our objective is to investigate and compare the performance of standard TCP implementations on different operating systems, particularly on Windows 7 and Linux. In this paper, we introduce our experimental methodology and show some exemplary results in our experiments in terms of throughput, fairness and so forth. Surprisingly, we find that TCP implementation in Linux probes slightly less packet drop than Windows 7, however, it increases the congestion window slower than Windows 7. Moreover, intra-implementation flows or inter-implementation flows are generally fair in both Windows 7 and Linux. More differences in detail are also found in the paper.**

I. Introduction

TCP (Transmission Control Protocol) is an important protocol and widely used by most Internet services including HTTP (Hypertext Transfer Protocol), SMTP (Simple Mail Transfer Protocol), FTP (File Transfer Protocol) and Web TV. The congestion control algorithm strongly impacts on the performance of TCP, eventually on the performance of applications. And it has been remarkably successful especially since the standard additive-increase-multiplicative-decrease (AIMD) algorithm [1] deployed.

While recent studies on TCP almost focus on new congestion control algorithms for high-speed network by modifying the AIMD algorithm. For example, CUBIC [2], Compound-TCP [3], HighSpeed TCP [4] and Scalable-TCP [5]. However, the standard TCP implementations, including Reno, NewReno, and SACK, are still dominant on the Internet. Microsoft Windows use TCP Reno by default in the consumer-oriented operating systems (e.g., Windows XP, Windows Vista and Windows 7) while enabling Compound-TCP by default in Windows Server 2008. BSD-based systems such as Mac OS X use TCP SACK by default in the latest version. Linux use CUBIC by default in kernels since version 2.6.19, however TCP Reno is still available in all kernels. As shown in Fig 1, the current OS including Microsoft Windows, Linux and Mac OS X are holding more than 85% usage share.

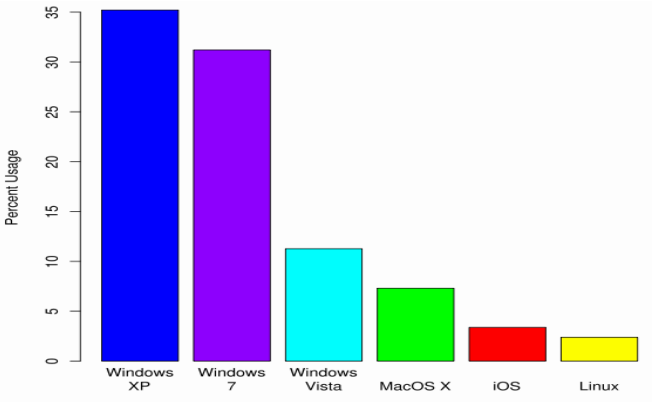

Fig. 1. Usage share of web client operating systems: August 2011. [6]

Previous works on TCP performance evaluation are most simulation-based such as [7], [8] and a few are single-OS-based such as [9], [10]. Each OS has a unique implementation in detail of standard TCP. Therefore, in this paper, our objective is to investigate and compare the performance of standard TCP implementations on different operating systems, in particular Windows 7 and Linux. We choose PlanetLab [11] as our testbed which provides a real Internet environment. We present our experimental methodology and show some exemplary results from our experiments. We evaluate TCP performance in terms of throughput, fairness and so forth. Based on extensive experimental data, we find that standard TCP implementation on Linux experience slightly lower packet drop rates than Windows 7. It is possibly due to the facts that 1) Linux TCP uses packet-based congestion window whereas Windows 7 TCP uses byte-based congestion window; 2) Linux provides a more precise clock than Windows 7. However, Linux TCP increases congestion window slower than Window 7 TCP. Moreover, both Windows 7 TCP and Linux TCP exhibit very nice fairness among two or four flows. But if flow number reached more than eight, it can be seen a noticeable decline in fairness.

The rest of this paper is organized as follows. In Section II we introduce our experimental methodology including per-

J. Yan is also affiliated with Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland.

978-1-4673-1544-9/12/$31.00 ©2012 IEEE

formance metrics and experimental setting. In Section III we analyze the results of conducted experiments and discuss some open questions and future work. Finally, conclusions are made in Section IV.

II. Experimental Methodology

In this section, we will first introduce our experimental method to evaluate the performance of both Linux and Windows 7 TCP implementations. And then we present experimental cases and setting in detail.

A. Metrics

In our experiments, we consider two primary metrics for evaluating the performance of standard TCP implementations: *throughput* and *fairness*.

1) Throughput: It is a clear purpose of most congestion control algorithms to achieve high throughput. So it is the first metric to evaluate TCP performance. We don't distinguish between throughput and goodput, but used the general term as [12] defined. We consider both aggregate behavior of all flows and individual behavior of each flow. In each test, we measured the average throughput aggregated from all flows. It is defined as Equation 1:

$$U := \sum_{i=1}^{n} \overline{x_i} \tag{1}$$

where $\overline{x_i}$ is the average throughput of the *i*th flow over the whole test, for totally *n* flows.

As the TCP congestion window size controls the throughput: AIMD algorithm increase the congestion window by roughly one segment per round-trip time (RTT) in the absence of congestion. To observe different implementations in more detail, we also measure the change of *cwnd* for each TCP flow. It is worth mentioning that the unit of congestion window is different between two operating systems: *Bytes* in Windows 7 and *Packets* in Linux. In order to show the difference clearly, we unify them in *Packets*.

2) Fairness: TCP fairness requires that a new protocol acquire no larger share of the network than a comparable TCP flow [12]. There are several definitions of fairness and it is usually divided into two parts: inter-protocol fairness and intra-protocol fairness. We exam two kinds of TCP fairness in our evaluation: inter-implementation fairness and intra-implementation fairness. Inter-implementation fairness compares the throughput of flows while senders use different operating systems. On the contrary, intra-implementation fairness is the throughput comparison of flows while senders use the exactly same TCP implementation and OS. We use Jain's fairness index [13] (see also in Equation 2) which provides a metric for max-min fairness.

$$\mathcal{J}(x_1, x_2, \ldots, x_n) := \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} x_i^2} \tag{2}$$

where x_i is the measured throughput of *i*th flow, for totally *n* flows.

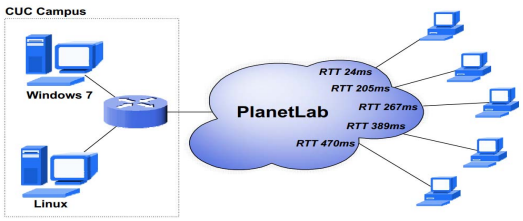

Fig. 2. Testing topology

TABLE I
Hardware Configuration and OS version of Senders

CPU	Intel Pentium E5500 2*2.8GHz
MEMORY	3 GBytes
NIC	Marvell Yukon 88E8057 PCI-E
Windows Version	Windows 7 Ultimate: 6.1.7600.16385
Linux Kernel Version	2.6.39.4

B. Experimental Cases

A series of test cases were designed in order to evaluate the performance of TCP implementations in real Internet environment. In each case, senders connect to a single receiver at the same time so that *n* flows shared bottleneck link and other network conditions such as packet loss rate, propagation delay and etc. In all test cases, the RTT ranges from 25 ms to 470 ms.

- Case 1: 1-flows on Windows 7 vs. 1-flows on Linux
  A single TCP flow is established on Windows 7 or Linux. After 60 seconds, the other one is established on a different sender (different operating system). This case shows the competition between different TCP implementations of Windows 7 and Linux.
- Case 2: *n*-flows on Windows 7
  A single TCP flow is established on Windows 7. After 60 seconds, the second TCP flow is established on the same sender. If *n* is greater than 2, then after the same time slot (60 seconds), another TCP flow is established. It requires each flow to run at least 400 seconds. This case shows the competition of Windows 7 TCP flows.
- Case 3: *n*-flows on Linux
  Similar to Case 2, *n* flows are established on Linux. It shows the competition of Linux TCP flows.

C. Experimental Setting

The topology used in our experiments is shown in Fig 2. We deploy two nodes as sender at Communication University of China (CUC) in Beijing with identical hardware as shown in Table I. However, one is running Windows 7 and the other is running Fedora Linux. Remote nodes as receivers are selected on PlanetLab [11] with different RTT which also running Fedora Linux. To measure TCP parameters such as *cwnd*, *ssthresh* and sample RTT, the function *getsockopt()* with *TCP_INFO* option is used in Linux while IP Helper API is used in Windows 7.

We list more settings in the following to conduct our experiments:

- TCP parameters are set to the same value in both operating systems. In particular, SACK [14] is enabled and MTU size is set to 1500 bytes.
- Both before and after each test, we check the route path using *traceroute* from senders to receivers. We find that the Internet path is relative stable during one test duration (10 minutes in average).
- Each individual test case is repeated at least five times, and we present the arithmetic mean of the results.

III. Results

In this section, we present some representative results in our intensive experiments. The results illustrate different behaviors of TCP implementations in terms of throughput, *cwnd* and fairness, and also some detailed problems and differences we observed.

A. Case 1: 1-flow on Windows 7 vs. 1-flow on Linux

The first case compared the performance of different standard TCP implementations under the same network conditions. Fig 4, Fig 5 and Fig 6 show the examples of measured *cwnd* for two flows with RTT 25 ms, 267 ms and 389 ms separately. One curve represents Windows 7 TCP flow and the other one represents Linux.

We find that Windows 7 often experiences a slightly higher packet drop rate than Linux. Meanwhile, the *cwnd* increasing phase is quite different between them. Fig. 3 shows the details of this difference. Increasing one packet of *cwnd* on Linux takes almost 80 ms which is approximately three times RTT (26.5 ms at the time being). In the meantime, Windows 7 roughly increases the *cwnd* by one packet per RTT. As a result, Windows 7 increases the *cwnd* faster than Linux. In addition, Linux sends packets and increases *cwnd* in the beginning of each round-trip. In contrast, Windows 7 sends packets and increases *cwnd* across each individual round-trip time, namely TCP pacing is applied in Windows 7.

As noted previously, Linux uses a packet-based *cwnd* whereas Windows 7 use byte-based *cwnd*. Linux sender compares the number of outstanding segments to *cwnd* when determining how much data to transmit. It always tracks the number of outstanding segments in units of full-sized packets, however, Windows 7 compare *cwnd* to the number of transmitted bytes no matter full sized or not. This will generate some differences if the sender transmits small packets (not full-sized). Another factor is that the clock for RTT measurement on Linux is 1 ms whereas 10 ms on Windows 7, which also leads some performance differences between them.

Table II summarizes the ratio of measured throughput and inter-implementation fairness calculated by Jain's index for this case. The results are the mean value of five repeated tests. The standard TCP implementations on Windows 7 and Linux show a good inter-implementation fairness and a similar average throughput during one test duration. As mentioned above, Windows 7 increases the *cwnd* faster than Linux, so it should be more aggressive. However, from Fig 4, Fig 5 and Fig 6, it can be seen that packet drop rate show a high correlations which leads different packet drop rate of them. Therefore, their throughput are comparable in our experiments.

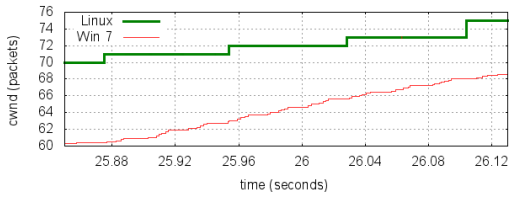

Fig. 3. The details of additive increasing *cwnd* on Windows 7 and Linux. The receiver is planetlab-1.sjtu.edu.cn, and RTT is 25 ms in average.

TABLE II
Throughput and Fairness under Linux/Windows 7 coexistence

RTT (ms)	OS	Throughput (Mbps)	Fairness
25	Windows 7	21.722	0.9992
	Linux	22.948	
205	Windows 7	2.242	0.9984
	Linux	2.072	
267	Windows 7	2.306	0.9988
	Linux	2.155	
389	Windows 7	1.354	0.9949
	Linux	1.452	
470	Windows 7	1.032	0.9975
	Linux	1.104	

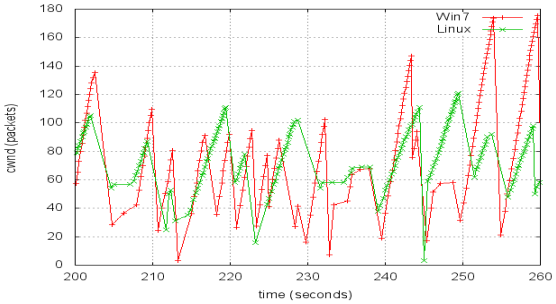

Fig. 4. The *cwnd* for two flows (one is on Windows 7 and the other is on Linux). The receiver is planetlab-1.sjtu.edu.cn, and RTT is 25 ms in average.

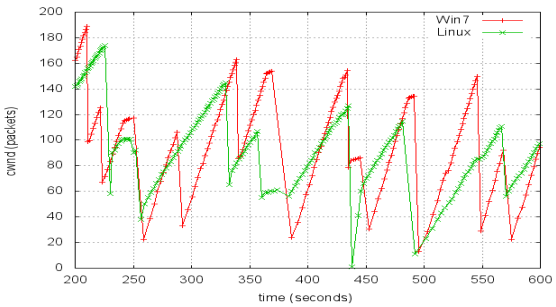

Fig. 5. The *cwnd* for two flows (one is on Windows 7 and the other is on Linux). The receiver is planetlab01.uncc.edu, and RTT is 267 ms in average.

B. Case 2: n-flows on Windows 7

In this case, we evaluate the performance of TCP implementation on Windows 7. We consider round-trip time in the range from 25 ms to 470 ms and perform tests for various number of flows. We only show figures of results with two

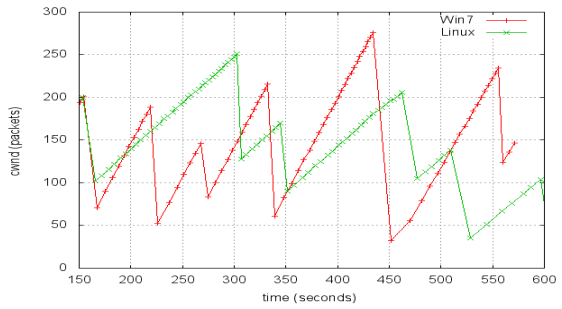

Fig. 6. The *cwnd* for two flows (one is on Windows 7 and the other is on Linux). The receiver is planetlab-1.ing.unimo.it, and RTT is 389 ms in average.

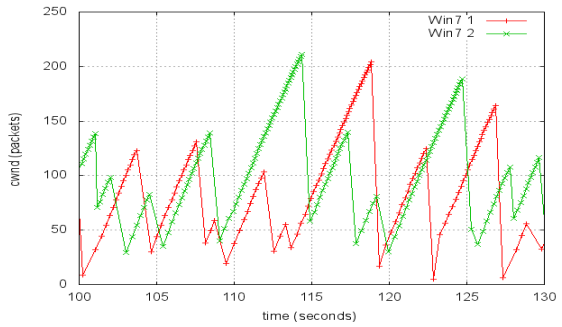

Fig. 7. The *cwnd* for two flows on Windows 7. The receiver is planetlab-1.sjtu.edu.cn, and RTT is 25 ms in average.

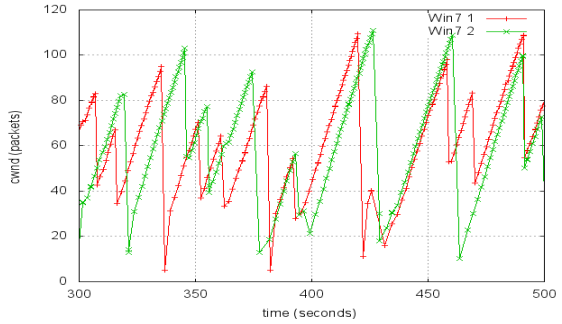

Fig. 8. The *cwnd* for two flows on Windows 7. The receiver is planetlab01.uncc.edu, and RTT is 267 ms in average.

flows for clarity reason. Fig 7, Fig 8, and Fig 9 show the examples of measured *cwnd* on Windows 7 with RTT 25 ms, 267 ms and 389 ms respectively. Table III summarizes the ratio of throughput fairness for the Windows 7 implementation. The results are the mean value of five repeated tests.

From Table III, we can see that the standard TCP implementation of Windows 7 TCP/IP stacks exhibits a very nice intra-implementation fairness and comparable throughput. But if the number of flow up to 8, the fairness begin to decline.

However, we unexpectedly find that standard TCP implementation on Windows 7 often detects a succession of packet loss events after each fast retransmission period, and then the *cwnd* and *ssthresh* decrease to very small values. We are still working on finding the root cause of this unexpected behavior.

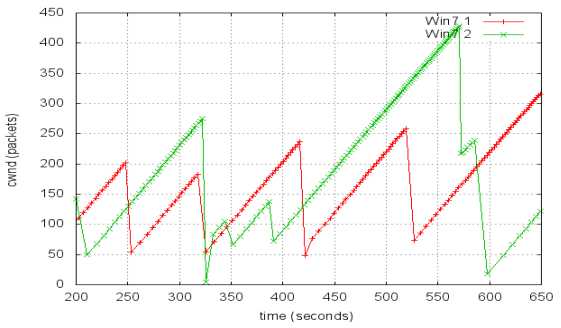

Fig. 9. The *cwnd* for two flows on Windows 7. The receiver is onelab7.iet.unipi.it, and RTT is 389 ms in average.

TABLE III
FAIRNESS WITH THE SAME RTT FOR WINDOWS 7 TCP

RTT (ms)	Fairness		
	2 Flows	4 Flows	8 Flows
25 ms	0.9999	0.9988	0.9947
205 ms	1	0.9687	0.8827
267 ms	1	0.9996	0.8834
389 ms	0.9998	0.9978	0.8989
470 ms	0.9999	0.9975	0.8563

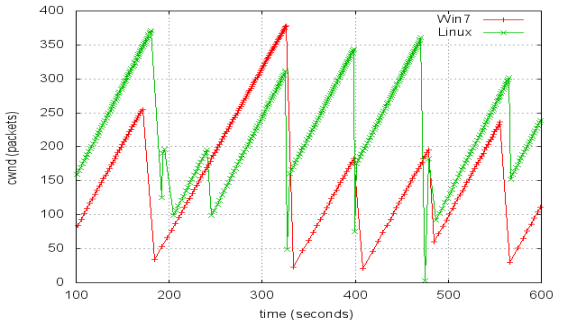

Fig. 10. The *cwnd* for two flows (one is on Windows 7 and the other is on Linux). The Linux sender was enabling TCP ABC. The receiver is onelab7.iet.unipi.it, and RTT is 389 ms in average.

C. Case 3: n-flows on Linux

In this case, we evaluate the performance of standard TCP implementation on Linux. We also consider RTT in the range from 25 ms to 470 ms, perform tests for various flows and show figures of results with two flows. Fig 11, Fig 12, and Fig 13 show the examples of measured *cwnd* on Linux with the same RTT compared to Windows 7. Table IV summarizes the ratio of throughput fairness (mean value of five repeated tests) for Linux TCP implementation.

We find that the standard TCP implementation of Linux also exhibits a quite good intra-implementation fairness compared to Windows 7. Linux TCP often increases *cwnd* by one full-sized segment every two or three RTTs instead of per RTT. The root cause is that Linux only increases *cwnd* by one packet after an entire MSS has been transmitted. If there are some small packets (not full-sized) have been transmitted, Linux takes rather longer time to increase *cwnd*. Fig. 10 describes Linux enabled TCP Appropriate Byte Counting (ABC) op-

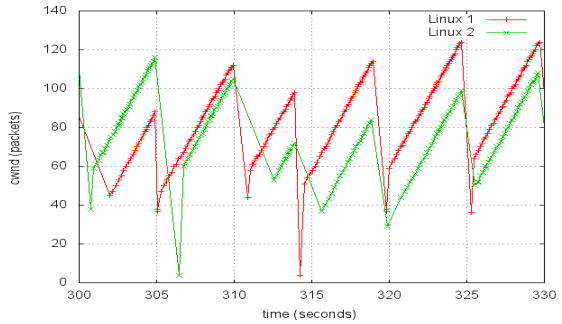

Fig. 11. The *cwnd* for two flows (top) on Linux. The receiver is planetlab-1.sjtu.edu.cn, and RTT is 25 ms in average.

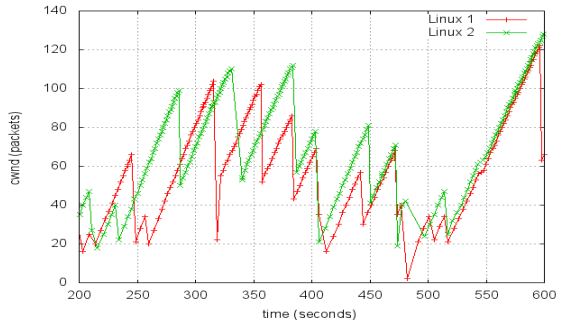

Fig. 12. The *cwnd* for two flows (top) on Linux. The receiver is planetlab01.uncc.edu, and RTT is 267 ms in average.

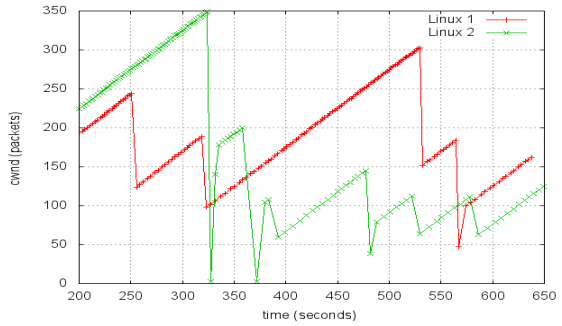

Fig. 13. The *cwnd* for two flows (top) on Linux. The receiver is onelab7.iet.unipi.it, and RTT is 389 ms in average.

TABLE IV
FAIRNESS WITH THE SAME RTT UNDER LINUX TCP

RTT (ms)	Fairness		
	2 Flows	4 Flows	8 Flows
25 ms	0.9987	0.9976	0.9966
205 ms	1	0.9911	0.8332
267 ms	0.995	0.992	0.9181
389 ms	0.9972	0.9953	0.8523
470 ms	0.9983	0.9965	0.8402

tion [15] which could get a more appropriate *cwnd* growth. However, by default, TCP ABC has been disabled since Linux kernel 2.16.18 because it unfairly penalizes some applications that do small writes (See kernel ChangeLog-2.6.18).

IV. CONCLUSIONS AND FUTURE WORK

In this paper, we present our experimental results which evaluating the performance of standard TCP implementations on Linux and Windows platforms. We measured the performance through a series of experiments in the real Internet environment. We find that standard TCP implementations on Windows 7 and Linux perform differently in detail. An important reason is that Linux uses packet-based *cwnd* and a more precise clock whereas Windows 7 uses byte-based *cwnd*. Surprisingly, we find that Linux TCP implementation detect slightly less packet drops than Windows 7 implementation. However, Linux increases *cwnd* slower than Windows 7 by default configuration. Moreover, both Windows 7 and Linux TCP flows are fair to intra-implementation flows and inter-implementation flows in general. We are currently working on the impact of TCP performance for applications such as Web TV [16].

ACKNOWLEDGMENT

This work was supported partly by NSFC (National Natural Science Foundation of China) under grant No. 60970127, Science and Technology Key Project of Ministry of Education (No. 109029) and the Program for New Century Excellent Talents in University (NCET-09-0709).

REFERENCES

[1] V. Jacobson, "Congestion avoidance and control," in *Symposium proceedings on Communications architectures and protocols*, ser. SIGCOMM '88. New York, NY, USA: ACM, 1988, pp. 314–329.
[2] I. Rhee and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," in *Proceedings of the International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet '05)*, Feb. 2005.
[3] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound TCP approach for high-speed and long distance networks," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, april 2006, pp. 1 –12.
[4] S. Floyd, "Highspeed TCP for large congestion windows," RFC 3649 (Experimental), Dec. 2003.
[5] T. Kelly, "Scalable tcp: Improving performance in highspeed wide area networks," *SIGCOMM Comput. Commun. Rev.*, vol. 33, pp. 83–91, April 2003.
[6] "Usage share of OS," http://en.wikipedia.org/wiki/OS_usage_share.
[7] K. Fall and S. Floyd, "Simulation-based comparisons of tahoe, reno and sack tcp," *SIGCOMM Comput. Commun. Rev.*, vol. 26, pp. 5–21, July 1996.
[8] S. Gelle, "Performance of high-speed tcp protocols over ns-2 tcp linux," Masters Project Final Report, Old Dominion Universit, 2008.
[9] Y.-T. Li, "Evaluation of TCP Congestion Control Algorithms on the Windows Vista Platform," 2006, SLAC-TN-06-005.
[10] Y.-T. Li, D. Leith, and R. N. Shorten, "Experimental evaluation of TCP protocols for high-speed networks," *IEEE/ACM Trans. Netw.*, vol. 15, pp. 1109–1122, October 2007.
[11] "Planelab," http://www.planet-lab.org/.
[12] S. Floyd, "Metrics for the evaluation of congestion control mechanisms," RFC 5166, Mar. 2008.
[13] W. H. R. Jain, D. Chiu, "A quantitative measure of fairness and discrimination for resource allocation in shared computer system," *DEC Research Report*, vol. TR-301, 1984.
[14] F. S. Mathis M., Mahdavi J. and R. A., "Tcp selective acknowledgment options," RFC 2018, Oct. 1996.
[15] M. Allman, "TCP congestion control with appropriate byte counting," RFC 3465 (Experimental), Feb. 2003.
[16] J. Yan, W. Muehlbauer, and B. Plattner, "Analytical framework for improving the quality of streaming over TCP," *IEEE Transactions on Multimedia*, 2012, to appear.