CUBIC-FIT: A high performance and tcp CUBIC friendly congestion control algorithm

Article *in* IEEE Communications Letters · August 2013

DOI: 10.1109/LCOMM.2013.060513.130664

6 authors, including:

Jingyuan Wang
Beihang University (BUAA)
65 PUBLICATIONS **1,675** CITATIONS

SEE PROFILE

Chao Li
Beihang University (BUAA)
67 PUBLICATIONS **637** CITATIONS

SEE PROFILE

Zhang Xiong
Nanjing Agricultural University
209 PUBLICATIONS **2,977** CITATIONS

SEE PROFILE

CUBIC-FIT: A High Performance and TCP CUBIC Friendly Congestion Control Algorithm

Jingyuan Wang, Jiangtao Wen, *Fellow, IEEE*, Yuxing Han, Jun Zhang, Chao Li and Zhang Xiong

***Abstract*—With the recent popularity of Linux-based HTTP servers, TCP CUBIC, the default Linux congestion control algorithm, is close to the new de facto standard algorithm for the Internet congestion control. A need for new congestion control technologies for the TCP CUBIC-dominated Internet is therefore emerging. To cater to this trend, this paper proposes a novel TCP congestion control algorithm, CUBIC-FIT. Compared with TCP CUBIC and other state-of-the-art TCP algorithms, CUBIC-FIT can improve performance over a large range of network conditions and maintain graceful fairness with the widely deployed TCP CUBIC servers. Analytical and experimental studies over emulators and real networks validate the proposed algorithm and its performance.**

***Index Terms*—Congestion control, friendliness, TCP CUBIC**

I. Introduction

The standard TCP congestion control algorithm, TCP Reno, achieved great success for several decades and was widely used as the default algorithm for many mainstream Operating Systems. Because of its widespread use, maintaining friendliness with Reno is an important consideration when designing a new congestion control algorithm. Researchers have proposed many high-performance and Reno-friendly congestion control algorithms, including Compound TCP [1] and TCP-FIT [2] to improve TCP Reno over different network conditions. However, the standard place of TCP Reno across the Internet is changing as the popularity of Linux-based web servers increases. TCP CUBIC [3], the default algorithm of Linux kernels 2.6.19 and above, is becoming the new de facto standard for the Internet congestion control. P. Yang et al. [4] measured the congestion control algorithms of the 5,000 most popular web servers. CUBIC had been adopted by 30.15% of these servers. TCP BIC, the default algorithm of Linux kernels 2.6.18 to 2.6.19, was used by 14.36% of these servers (this set of servers is likely to adopt CUBIC in the future as kernels are updated). Only 16.85% to 25.58% of these servers still used the Reno algorithm. These findings demonstrate that Reno no longer controls a majority of the Internet TCP traffic. New congestion control technologies for the CUBIC-dominated Internet are required.

There are two challenges for the CUBIC algorithm and CUBIC-dominated networks. The first challenge is *performance over wireless links*. CUBIC is designed for large BDP networks but not wireless networks. CUBIC uses packet losses as network congestion symptoms, so random physical and MAC layer artifacts (e.g. multi-path, interferences [5]) introduced packet losses can cause performance degradation of TCP CUBIC. The second challenge is *CUBIC friendliness*. TCP flows in CUBIC-dominated networks should stay friendly with other CUBIC flows. However, most existing algorithms are designed to Reno-dominated networks and maintain friendliness with Reno but not CUBIC, which introduces serious fairness problems [6].

In the present paper we propose a new TCP variant, called CUBIC-FIT. CUBIC-FIT introduces the idea of delay-based TCP into the TCP CUBIC algorithm framework. CUBIC-FIT can improve throughput performance over wireless networks more than plain CUBIC and maintain graceful friendliness with widely deployed plain CUBIC servers. Theoretical analysis, as well as experiments over emulators and the PlanetLab testbed, is performed to characterize and validate the performance of our proposed approach.

II. The CUBIC-FIT Algorithm

CUBIC-FIT is based on the TCP CUBIC algorithm framework. In TCP CUBIC algorithm, packet losses are treated as a symptom of network congestion. When a packet loss event is detected, TCP CUBIC use a variant w_{max} to records the current congestion control window size as a network capacity estimate, and sets the window size w_{cubic} as $(1-b)\cdot w_{max}$ to recover the network from congestion, where $b \in (0,1)$ is the window decrease factor of the CUBIC sender. When there are no packet losses, TCP CUBIC uses w_{max} as a parameter and probes the network capacity using a cubic function:

$$w_{cubic} := C(t-I)^3 + w_{max}, \quad I = \sqrt[3]{(w_{max}b)/C}. \qquad (1)$$

In Eq. (1), C is a preset scaling parameter, and t is the elapsed time from the last packet loss. Because CUBIC flows reduce their window when packet loss is detected, random packet losses introduced by wireless links severely harm throughputs.

Manuscript received mm dd, yyyy. The associate editor coordinating the review of this letter and approving it for publication was F. Theoleyre.

J. Wang, C. Li and Z. Xiong are with the State Key Laboratory of Software Development Environment, Beihang University, 100191 China, and Research Institute of Beihang University in Shenzhen, Sehnzhen, 518057 China. Email: {jywang, licc, xiongz}@buaa.edu.cn

J. Wen and J. Zhang are with the Department of Computer Science, Tsinghua University, Beijing, 100084 China. Email: jtwen@tsinghua.edu.cn, junzhang10@mails.tsinghua.edu.cn.

Y. Han is with the Flora Production Inc., Santa Clara, CA 95054 US. Email: ehan@floraproduction.com.

The work was supported by the National Natural Science Foundation of China (Grant No. 61202426, 61272350), the National Science Fund for Distinguished Young Scholars of China (Grant No. 61125102), the State Key Program of National Natural Science of China (Grant No. 61133008), and the State Key Laboratory of Software Development Environment (SKLSDE-2012ZX-20).

Digital Object Identifier XX.XXXX/LCOMM.XXXX.XX.XXXXXX

To improve the CUBIC throughput performance over wireless links, CUBIC-FIT uses the idea of delay-based TCP to extend the CUBIC algorithm framework. CUBIC-FIT simulates behavior of multiple CUBIC flows in a single TCP connection to fully utilize network capacity, and uses the end-to-end queuing delay to adjust the simulated CUBIC flow number. As shown in Eq. (1), the CUBIC framework can tune two parameters, C and b. According to [3], the relationship between the transmission rate of a CUBIC flow, T_{cubic}, and the two parameters for giving averaged packet loss rate, PLR, and average round-trip time, RTT, is

$$T_{cubic} = \frac{1}{RTT}\sqrt[4]{\frac{C(4-b)}{4b}\left(\frac{RTT}{PLR}\right)^3}. \quad (2)$$

In the setting of plain CUBIC, $C = 0.4$ and $b = 0.2$. Therefore the function (2) is

$$T_{cubic}^{plain} = \frac{1}{RTT}\sqrt[4]{1.9\left(\frac{RTT}{PLR}\right)^3}. \quad (3)$$

Because CUBIC-FIT simulates N plain CUBIC flows using only one window (e.g., not running N independent CUBIC windows in parallel and summing them), the throughput function of CUBIC-FIT should be

$$T_{cubic}^{fit} = N \cdot T_{cubic}^{plain} = \frac{N}{RTT}\sqrt[4]{1.9\left(\frac{RTT}{PLR}\right)^3}. \quad (4)$$

If we implement the throughput function (4) under the CUBIC algorithm framework defined in (1), the CUBIC-FIT parameters C and b should satisfy $\frac{C(4-b)}{4b} = 1.9N^4$. Letting $C = 0.4N^3$, we obtain $b = \frac{4}{19N+1}$, the window growth function for CUBIC-FIT becomes

$$w_{cubic}^{fit} = 0.4(Nt - I)^3 + w_{max}, \quad I = \sqrt[3]{\frac{10w_{max}}{19N+1}}, \quad (5)$$

and the behaviors of N plain CUBIC flows can be emulated in a single connection.

N is an important parameter to control aggression in CUBIC-FIT. N values that are too small may cause network capacity under-utilization, but N values that are too large may cause starvation in other TCP flows. CUBIC-FIT uses end-to-end RTT to periodically updates parameter N. The new N at period $t+1$ is calculated as [2]

$$N_{t+1} := \max\left\{1,\ N_t + 1 - \frac{RTT_t - RTT_{min}}{\alpha \cdot RTT_t}N_t\right\}, \quad (6)$$

where RTT_t is the average RTT of period t, and RTT_{min} is the observed minimal RTT. The parameter $\alpha \in (0,1)$ is discussed in Section III. In the implementation, N is updated for every 500 ms. Plugging (6) into (4), the throughput of CUBIC-FIT for given average PLR and RTT is

$$T_{cubic}^{fit} = \max\left\{\frac{1}{RTT},\ \frac{\alpha}{q}\right\}\sqrt[4]{1.9\left(\frac{RTT}{PLR}\right)^3}, \quad (7)$$

where $q = RTT - RTT_{min}$ represents the network queuing delay. According to Eqs. (6) and (7), a low queuing delay q leads to an increase in N to improve performance if networks are not fully utilized. Conversely, CUBIC-FIT decreases N to maintain friendliness with plain CUBIC flows when network suffers long queuing delays informed congestion.

III. Discussion

This section analyzes network utilization efficiency and the CUBIC friendliness of CUBIC-FIT using a simple but widely adopted network model [2]. In the network model, S TCP flows indexed by $s \in \{1, ..., S\}$ share a bottleneck link at the same time. The bottleneck link has a bandwidth limitation B, a round-trip propagation delay D and a random packet loss rate P. When the S TCP flows send packets through the bottleneck link, the relationships between the bottleneck link queuing delay q, the congestion packet loss rate p and the aggregate throughput of S TCP flows, $T_s = \sum_{s\in S} T_s$, are given by

$$T_S = \sum_{s\in S} T_s \begin{cases} \le B, \text{ if } p = 0,\ q = 0, & (a) \\ = B, \text{ if } p = f(q),\ q > 0, & (b) \end{cases} \quad (8)$$

where T_s is the throughput of flow s, and the queue management algorithm in the bottleneck determines $f(\cdot)$. We assume that $f(0) = 0$ and $f(\cdot)$ is a non-decreasing function of q.

Theorem 1: If S flows contain a CUBIC-FIT flow, the bottleneck link operates in state (b) of Eq. (8).

Proof: The aggregate throughput in the network satisfies $\sum_{s\in S} T_s \le B$. Suppose the bottleneck link operates in state (a) with $q = 0$. From Eq. (7), the throughput of CUBIC-FIT $T_{cubic}^{fit} \to \infty$. This contradicts $\sum_{s\in S} T_s \le B$. The bottleneck is thus always operating in state (b). ■

Theorem 1 shows that CUBIC-FIT can ensure that the network bottleneck capacity is fully utilized. Similar conclusions cannot be proven for plain CUBIC, according to Eq. (3). At least theoretically, CUBIC-FIT has a network utilization efficiency advantage compared with plain CUBIC.

We next analyze the fairness of CUBIC-FIT with plain CUBIC, i.e., CUBIC friendliness. We use *Bandwidth Stolen Rate* [1] as the metric of CUBIC friendliness. Let T_0 be the average throughput of S plain CUBIC flows sharing the bottleneck link defined by (8). Let T' be the average throughput of $S-1$ plain CUBIC flows when they share the bottleneck with a CUBIC-FIT flow. The *Bandwidth Stolen Rate* of the CUBIC-FIT flow is defined as $\Phi = (T_0 - T')/T_0$. As Φ becomes closer to 0, the CUBIC-FIT flow is friendlier to plain CUBIC flows. We discuss CUBIC-FIT's *Bandwidth Stolen Rate* in two different network conditions:

1) S CUBIC flows let the bottleneck run in (b) of Eq. (8);
2) S CUBIC flows let the bottleneck run in (a) of Eq. (8).

For network condition 1), we have the following theorem.

Theorem 2: Let q_0 be the queuing delay of the bottleneck when S plain CUBIC flows share the bottleneck, and let q' be the queuing delay when $S-1$ plain CUBIC flows compete with a CUBIC-FIT flow. The CUBIC-FIT flow is CUBIC friendly when its $\alpha \le \frac{q_0}{D+q_0}$.

Proof: We prove Theorem 2 by demonstrating that $q_0 = q'$ by contradiction, which is obtained using the throughput function of plain CUBIC (3) and CUBIC-FIT (7). In Theorem 2, if $q_0 < q'$, then $p_0 = f(q_0) \le p' = f(q')$.

According to the CUBIC throughput function (3), we obtain $T_0 > T' \Rightarrow B-(S-1)T_0 < B-(S-1)T'$. Hence,

$$\frac{1}{D+q_0}\sqrt[4]{\left(\frac{D+q_0}{P+p_0}\right)^3} < \frac{\alpha}{q'}\sqrt[4]{\left(\frac{D+q'}{P+p'}\right)^3}. \quad (9)$$

Because $\alpha \le \frac{q_0}{D+q_0}$, inequality (9) is false under the conditions $q_0 < q'$ and $p_0 \le p'$. Therefore, $q_0 \ge q'$.

If $q_0 > q'$, then $p_0 = f(q_0) \ge p' = f(q')$. From function (3), then, $T_0 < T' \Rightarrow B-(S-1)T_0 > B-(S-1)T'$,

$$\frac{1}{D+q_0}\sqrt[4]{\left(\frac{D+q_0}{P+p_0}\right)^3} > \max\left\{\frac{1}{D+q'}, \frac{\alpha}{q'}\right\}\sqrt[4]{\left(\frac{D+q'}{P+p'}\right)^3},$$

which is false under $q_0 > q'$, $p_0 \ge p'$. Therefore $q_0 = q'$, $T_0 = T'$ and $\Phi = 0$. CUBIC-FIT is plain CUBIC friendly. ■

Based on Theorem 2, we use the average $(RTT_{max} - RTT_{min})/RTT_{max}$ of a CUBIC-FIT server to approximate the $\frac{q_0}{D+q_0}$, where RTT_{max} and RTT_{min}, respectively, are the maximal and minimal RTT samples of a TCP flow. In practical implementation, α is set by

$$\alpha := \min\left\{\frac{1}{10}, \frac{RTT_{max}-RTT_{min}}{2RTT_{max}}\right\}.$$

Experimental results show that both the throughput and plain CUBIC friendliness of the implementation are good when adopting this approximation.

Next, we investigate the plain CUBIC friendliness of CUBIC-FIT under condition 2). With random packet losses, networks may run under this condition. In condition 2), $p_0 = 0$, and $q_0 = 0$, and therefore the average throughput loss of the plain CUBIC flows is

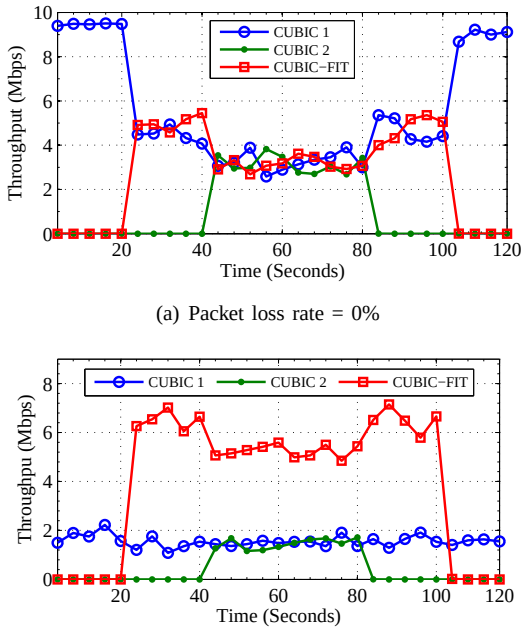
$$T_0 - T' = \frac{1}{D}\sqrt[4]{\frac{1.9D^3}{P^3}} - \frac{1}{D+q'}\sqrt[4]{1.9\left(\frac{D+q'}{P+p'}\right)^3}.$$

In this state, $T_0 - T'$ is caused by the queuing delay and congestion loss introduced by CUBIC-FIT saturating the network bottleneck, which should not be considered as CUBIC-FIT "stealing" bandwidth from plain CUBIC but rather as a reasonable cost of utilizing the extra bottleneck bandwidth.

IV. Experimental Results

To illustrate the CUBIC-FIT features, we present several experiments in this section. In the experimental setups, Linux-based TCP servers and clients are connected through a Linktropy hardware network emulator [7], which injects random packet losses and delays (packet losses and delays do not include congestion losses and queuing delays caused by network congestion) into the connection and caps the network bandwidth at a selected value. CUBIC-FIT is embedded in the Linux kernel of TCP servers. Other TCP algorithms are available by the Linux kernel.

We first investigate the performance and friendliness of CUBIC-FIT in a sequentially starting scenario. In this scenario, a CUBIC-FIT flow shares a 10 Mbps bottleneck link with two plain CUBIC flows. The propagation delay of the bottleneck link is set to 100 ms, and the network buffer is half of the network BDP. Two CUBIC-FIT flows and a plain CUBIC flow sequentially start and end every 20s. Fig. 3(a) shows the throughput variation of the plain CUBIC and CUBIC-FIT flows when the packet loss rate of the link is 0%. As shown in the figure, the CUBIC-FIT flow fairly shares the bandwidth with the other two plain CUBIC flows from 40 s to 80 s. Parameter N of the CUBIC-FIT flow converges to 1 when the random packet loss rate is 0%; thus, the CUBIC-FIT flow throughput is the same as that of the plain CUBIC flows and can fairly share bandwidth with plain CUBIC. Fig. 3(b) shows the flow throughput variations when the link has a 1% packet loss rate. As shown in the figure, the plain CUBIC flows cannot fully utilize the network bandwidth because of the harm of random losses, but although the CUBIC-FIT flow picks up the remained bandwidth. When the CUBIC-FIT flow appeared in 20 s to 100 s, the CUBIC flow throughputs has no reduction, which means that CUBIC-FIT also keeps graceful friendliness with plain CUBIC flows over lossy networks. As shown in Fig. 1, CUBIC-FIT can maintain friendliness with plain CUBIC flows and fully utilize the network capacity over both lossy and loss-free networks.



(a) Packet loss rate = 0%

(b) Packet loss rate = 1%

Fig. 1. Throughput variation of sequentially starting experiments

TABLE I
Jain's Index of different TCP with CUBIC flows (PLR = 0%)

	CUBIC-FIT	CUBIC	Reno	TCP-FIT	Compound
10 ms	0.999	0.999	0.948	0.975	0.936
50 ms	0.999	0.999	0.698	0.884	0.682
100 ms	0.999	0.999	0.655	0.633	0.615
150 ms	0.999	0.999	0.590	0.571	0.588
200 ms	0.999	0.999	0.560	0.551	0.555

Table I and Fig. 2 shows the results of the expanded experiments. In the experiments, the CUBIC-FIT throughput and fairness are compared with those of plain CUBIC, TCP-FIT and Reno. A TCP flow that uses one of the tested algorithms

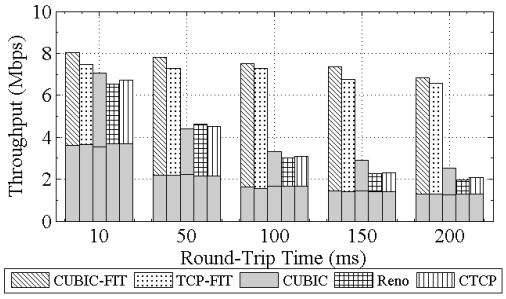
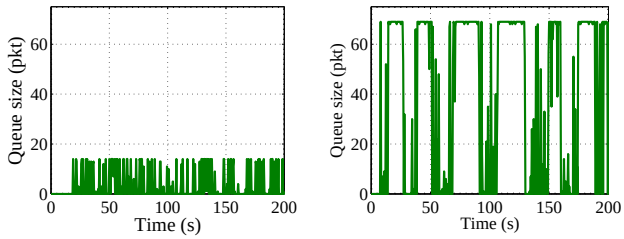


Fig. 2. Bandwidth occupation between competing TCP flows (PLR = 1%)



(a) Buffer size = 70 packets (b) Buffer size = 15 packets

Fig. 3. Network stability of CUBIC-FIT with different buffer size

shares a 10 Mbps bottleneck with a plain CUBIC flow. The propagation delay of the bottleneck is varied from 10 ms to 200 ms, and 5 exponential distributed UDP flows that have an aggregate rate of 10% bottleneck bandwidth are generated as background traffic. The UDP flows just make the experiment closer to the real, and don't influence the conclusion of the experiments. Table I lists the Jain fairness index of the tested algorithm and plain CUBIC flows when the bottleneck packet loss rate is 0%. Table I shows that the Jain indexes of CUBIC-FIT are close to 1, indicating CUBIC-FIT can fairly share the bottleneck plain CUBIC. However, the Jain indexes of Reno, Compound, and TCP-FIT are lower than CUBIC-FIT and plain CUBIC, indicating that the fairness feature of these algorithms are not suitable for CUBIC-dominated networks. Fig. 2 shows the bandwidth occupation of different algorithms when the bottleneck packet loss rate is 1%. CUBIC-FIT in this setup can pick up the bandwidth that the plain CUBIC left unused. The plain CUBIC flows under the same propagation delay have similar throughput, which indicates the improved CUBIC-FIT throughput does not come at the expense of the plain CUBIC.

Next, we test the stability of CUBIC-FIT with different buffer size over ns-2 using a similar network setup given by G. Rauba et. al. [8]. In the experiment, 15 CUBIC-FIT shares a bottleneck link with 150ms RTT, 480Mbps bandwidth, and buffer of either 70 or 15 packets. When the buffer size is 70, the window behavior of CUBIC-FIT flows become synchronized. The synchronized flows lead an oscillatory queue length in network buffer, which is shown in Fig. 3(a). As shown in the Fig. 3(b), the flows synchronization does not happen for a small buffer with 15 packets. The experiment results are very similar to the behavior of TCP Reno in [8] with same network setup. These results imply that the stability of CUBIC-FIT with different buffer sizes is similar to the standard TCP Reno algorithm, which is acceptable for existing networks.

TABLE II
THE AVERAGE THROUGHPUT OF THE CUBIC AND TEST SERVERS

	CUBIC-FIT	TCP-FIT	CUBIC	Reno	CTCP
CUBIC SRV	20.32	24.62	20.45	24.76	25.02
Test SRV.	32.24	25.59	21.49	13.45	15.45
Aggregate	52.56	50.21	41.94	38.20	40.47

Finally, we test CUBIC-FIT on the PlanetLab testbed. PlanetLab is a group of computers available as a testbed for computer networking research [9]. In the experiment, 154 PlanetLab nodes located in 85 cities of 27 countries are used to download video clips from two HTTP servers located in San Diego, CA, USA. These nodes covered 128 ISPs, representing the condition of the current Internet. The two HTTP servers are located closely to each other to ensure a high rate of router overlap from servers to the PlanetLab nodes. In the experiment, in the two HTTP servers, one server that always uses plain CUBIC called CUBIC Server and the other server, called Test Server, switches its algorithm among CUBIC-FIT, TCP-FIT, Reno, plain CUBIC and Compound TCP (CTCP). Table II lists the average aggregative throughputs for the CUBIC and Test Servers. As shown in the figure, the CUBIC-FIT throughput is higher than that of the other TCP variants, i.e., CUBIC-FIT improved the throughput performance of plain CUBIC and other algorithms. As Table II shows, the average CUBIC Server throughput when the Test Server uses CUBIC-FIT is close to that obtained when using plain CUBIC, indicating that the CUBIC-FIT throughput gain is not caused by stealing bandwidth from the CUBIC Server. The proposed CUBIC-FIT algorithm can improve performance and maintain friendliness with plain CUBIC flows over wide area networks.

V. CONCLUSION

In this paper, we have presented a TCP congestion control algorithm that has high network utilization efficiency and can maintain friendliness with the TCP CUBIC algorithm. Initial results indicate that this idea holds merit.

REFERENCES

[1] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound TCP approach for high-speed and long distance networks," in *Proceedings of IEEE INFOCOM'06*, 2006, pp. 1–12.
[2] J. Wang, J. Wen, J. Zhang, and Y. Han, "TCP-FIT: An improved TCP congestion control algorithm and its performance," in *Proceedings of IEEE INFOCOM '11*, 2011, pp. 2894–2902.
[3] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.
[4] P. Yang, W. Luo, L. Xu, J. Deogun, and Y. Lu, "TCP congestion avoidance algorithm identification," in *Proceedings of IEEE ICDCS '11*, 2011, pp. 310–321.
[5] B. Wang, H. Li, and J. Cao, "An efficient MAC scheme for secure network coding with probabilistic detection," *Frontiers of Computer Science*, vol. 6, no. 4, pp. 429–441, 2012.
[6] K. Munir, M. Welzl, and D. Damjanovic, "Linux beats Windows! – or the worrying evolution of TCP in common operating systems," in *Proceedings of PFLDNet 2007*, February 2007.
[7] Linktropy Mini2 WAN Emulator, http://www.apposite-tech.com.
[8] G. Raina, D. Towsley, and D. Wischik, "Part II: control theory for buffer sizing," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 3, pp. 79–82, Jul. 2005.
[9] Planetlab, http://www.planet-lab.org/.