This paper was presented as part of the main technical program at IEEE INFOCOM 2011

TCP-FIT: An Improved TCP Congestion Control Algorithm and its Performance

Jingyuan Wang, Jiangtao Wen, Jun Zhang and Yuxing Han
Key Laboratory of Pervasive Computing, Ministry of Education
Tsinghua National Laboratory for Information Science and Technology
Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
Email: wangjingyuan06@mails.tsinghua.edu.cn, jtwen@tsinghua.edu.cn

***Abstract*—The Transport Control Protocol (TCP) has been widely used by wired and wireless Internet applications such as FTP, email and HTTP. Numerous congestion algorithms have been proposed to improve the performance of TCP in various scenarios, especially for high bandwidth-delay product (BDP) and wireless networks. Although different algorithms may achieve different performance improvements under different network conditions, designing a congestion algorithm that performs well across a wide spectrum of network conditions remains a great challenge. In this paper, we propose a novel congestion control algorithm, named TCP-FIT, which could perform gracefully in both wireless and high BDP networks. The algorithm was inspired by parallel TCP, but with the important distinctions that only one TCP connection with one congestion window is established for each TCP session, and that no modifications to other layers (e.g. the application layer) of the end-to-end system need to be made. Extensive experimental results obtained using both network simulators as well as over "live" wired line, WiFi and 3G networks at different geographical locations and at different times of the day are presented. The performance of the algorithm shown in the experiment results is significantly improved as compared to other state-of-the-art algorithms, while maintaining good fairness.**

I. Introduction

The Transmission Control Protocol (TCP) [1] is a reliable transport layer protocol that is widely used on the Internet. Congestion control algorithm [2] is an integral module of TCP that directly determines the performance of the protocol. Standard congestion control algorithms such as TCP Reno [3] and TCP NewReno [4], achieved great success for several decades but are found to perform poorly over wireless and/or high Bandwidth Delay Product (BDP) links. To improve TCP performance over wireless and high BDP networks, many TCP variants have been proposed, including TCP Westwood [5], [6], TCP Veno [7] for wireless applications and Compound TCP [8], TCP CUBIC [9], FAST TCP [10] and [11] for high BDP networks. Although these algorithms have achieved success in their respective target applications, designing a TCP congestion control algorithm that perform gracefully in **both** wireless and high BDP networks is still a great challenge. On the other hand however, with the deployment of highspeed wireless networks, such as LTE, WiMAX, as well as high bandwidth, real time applications such as multimedia over TCP/HTTP, it is required for the TCP congestion algorithm to handle both wireless connections with random radio related losses as well as congestion-introduced issues which is typical for wired high BDP networks.

In this paper, a novel TCP congestion control algorithm for the heterogeneous networks which contain high BDP links and wireless links, named TCP-FIT, is described. The algorithm was inspired by parallel TCP, but with the important distinctions that only one TCP connection with one congestion window is established for each TCP session, and that no modifications to other layers (e.g. the application layer) of the end-to-end system need to be made. Extensive experimental results obtained using both network simulators as well as over "live" wired line, WiFi and 3G networks at different geographical locations and at different times of the day are presented. It is shown that the performance of the algorithm is significantly improved as compared to other state-of-the-art algorithms, while maintaining good fairness.

The paper is organized as the following. Section II gives an overview of existing TCP congestion control algorithms and the motivation for the TCP-FIT algorithm. Section III describes the TCP-FIT algorithm in detail and the throughput model of TCP-FIT is introduced in Section IV. Section V provides theoretical analysis of the network capacity utilization and fairness performance of TCP-FIT. Emulation-based experiment results and performance measured over live networks are given in Section VI. Section VII concludes the paper.

II. Background and Motivations

The function of TCP congestion control algorithm is to adjust the rate with which the protocol sends packets into the network using a congestion control window. A good congestion algorithm can fully utilize the bandwidth while avoiding over-driving the network and thereby creating packet losses. Since the congestion control mechanism has been introduced in TCP by BSD UNIX, many TCP congestion control algorithms are proposed. On a high level, existing TCP congestion algorithms can be classified into three main categories based on the input of the control mechanism - namely Loss-based TCP, Delay-based TCP and Hybrid TCP.

Loss-based TCP includes the original TCP Reno, TCP BIC[12], TCP CUBIC, High Speed TCP[13], Scalable TCP[14], etc. Among these Loss-based TCP variants, TCP Reno and TCP CUBIC are widely deployed as standard TCP algorithms and default TCP of Linux respectively. Using

978-1-4244-9921-2/11/$26.00 ©2011 IEEE

packet loss as the symptom for network congestion, Loss-based TCP reduces the congestion control window when packet losses occur and increases the window otherwise. A basic assumption in the design of Loss-based TCP congestion control is that packet losses are caused by over-driving the network *only*, which is no longer valid when the algorithm is applied to wireless networks. Packet losses introduced via random physical layer artifacts (e.g. multi-path, interferences) which is typical for wireless networks will cause the congestion control algorithm to aggressively lower the congestion control window. On the other hand, in a high BDP network, Loss-based TCPs require a very low (in the order 10^{-7} or lower [10]) random packet loss rate to fully utilize network capacity. This requirement is far from the reality of the network condition nowadays.

Delay-based TCP including TCP Vegas [15] and FAST TCP [10] uses the queuing delay as the symptom of congestion. The queuing delay is defined as the difference between the RTT and the propagation delay, i.e. time actually required for a packet to be transmitted from the sender to the receiver. Delay-based TCPs are more resilient to transient changes of network conditions such as random packet losses and are also suitable for high BDP networks [16]. The down side of the approach on the other hand is that, because increasing in queuing delay doesn't necessarily immediately lead to packet loss (due to buffers), when Delay-based TCP stations share the same bottleneck with Loss-based TCP stations, between the time when delay starts to increase and packet loss occurs, the window for the Delay-based TCP will decrease while that for the Loss-based TCP will not, leading to bandwidth "starvation" for the Delay-based stations.

Hybrid TCP uses both packet loss and delay as inputs control and includes TCP variants for wireless environments such as Veno and TCP Westwood, as well as TCP variants for high speed links such as Compound TCP [8], TCP-Illinois [17], H-TCP [18] and Fusion TCP [19]. Among these algorithms, Compound TCP has been widely deployed in the Microsoft Windows Vista operating system. Although the performance of these TCP variants are good for the application scenarios they were originally designed for, for the emerging new generation of high bandwidth wireless networks such as LTE and WiMAX, as well as for applications over heterogeneous networks combining segments of high BDP and wireless links, it becomes difficult for existing TCP congestion control algorithms to perform well.

The TCP-FIT was motivated by various algorithms such as GridFTP [20] and E-MulTCP [21]. These algorithms can utilize available bandwidth more by establishing multiple actual parallel TCP connections for the same application [22]. Various studies have shown that if an appropriate number of parallel TCP sessions is chosen, the performance for such algorithms can be very good in wireless and large BDP networks [20] [21] [23]. Such parallel TCP algorithms typically operate outside of the scope of TCP congestion control, and may often require support from the application layer.

MulTCP is proposed in [24] to implement parallel TCP in the transport layer. In MulTCP, N virtual TCP Reno sessions are simulated in a single TCP session. Fig. 1 shows the simulation results when a MulTCP application with different values of N shared a network with a TCP Reno session. The bandwidth of the network was 10 Mbps with 1% random packet loss and 20 ms propagation delay. Fig. 1(a) shows the aggregated throughput of the MulTCP application and the Reno session. It is clear that when N is sufficiently large, network bandwidth can be fully utilized.

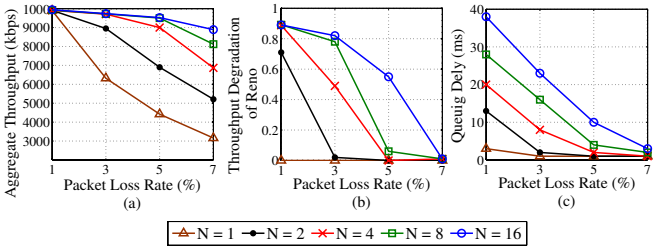


Fig. 1. Performance and fairness of MulTCP over wireless link

However, the performance of a congestion control algorithm needs to be evaluated by more parameters than throughput. If we define *Throughput Degradation* as the ratio of the throughput of a TCP Reno session when it shares the same network with MulTCP and the throughput for a TCP Reno session when it is the only connection over a network, we will have the results in Fig. 1(b). As can be seen from the figure, when N becomes larger, throughput degradation becomes more and more severe, leading to over 50% performance loss of the Reno session in many cases which is unacceptable in many real world applications. It is clear therefore, an intelligent and adaptive algorithm that chooses an appropriate value of N is critical to MulTCP in real applications. Some N adaption algorithms for parallel TCPs such as [25] have been proposed, but are mostly application layer algorithms. Designing a good algorithm suitable for integration with MulTCP on the transport layer remains an open issue.

Yet another observation of the behavior of MulTCP is given in Fig. 1(c), which shows its queuing delay. Comparing Fig. 1(b) and Fig. 1(c), we can see that there seems to be a strong correlation between the queuing delay of a MulTCP session and the throughput degradation of the competing Reno session on the same network. Using this correlation as a cue, we can adapt the values of N such that network bandwidth can be more utilized, while at the same time, the queuing delay is kept at an acceptable level, so that the session remains fair.

Based on these insights, we proposed a novel TCP congestion control algorithm called TCP-FIT. In contrast to other parallel TCP techniques, TCP-FIT is a fully compliant TCP congestion control algorithm requesting no modifications to other layers in the protocol stack and/or any application software.

III. The TCP-FIT Algorithm

In TCP-FIT, the Additive-Increase/Multiplicative-Decrease (AIMD) mechanism is employed to directly adjust the conges-

tion control window w. The AIMD combines linear growth of the congestion control window w with an exponential reduction when a packet loss event takes place. In general, the w adjustment equation of the AIMD can be expressed as

$$Each\ \ RTT:\ w \leftarrow w + A,$$
$$Each\ \ Loss:\ w \leftarrow w - B \cdot w,$$

where A and B are increasing and decreasing factors of AIMD respectively. In the TCP Reno, $A = 1$ and $B = 1/2$. To achieve N times throughput of the TCP Reno, the MulTCP sets $A = N$ and $B = 1/2N$. It will be shown in the analysis of Section IV, that given the same network conditions, the standard MulTCP with the parament N can not guarantee that the congestion window is exactly N times that of a single Reno session. Therefore, in the TCP-FIT, w is decreased by a factor of $B = 2/(3N + 1)$ instead of $B = 1/2N$ in the MultTCP when a packet loss occurs. This helps keep the throughput of TCP-FIT flows exactly N times of the TCP Reno at the same network condition. For TCP-FIT, the congestion control window w adjustment equation can be expressed as,

$$\begin{aligned} &Each\ \ RTT:\ w \leftarrow w + N, \\ &Each\ \ Loss:\ w \leftarrow w - \frac{2}{3N+1}w. \end{aligned} \tag{1}$$

In TCP-FIT, the value N is a dynamic parameter that is updated after each packet loss event. For the i-th updating period, N is updated by

$$N_i = \begin{cases} N_{i-1} + 1 & ,\text{if}\ \ Q < \alpha \cdot \dfrac{\bar{w}}{N_{i-1}}, \\ N_{i-1} & ,\text{if}\ \ Q = \alpha \cdot \dfrac{\bar{w}}{N_{i-1}}, \\ \max\{1, N_{i-1} - 1\} & ,\text{if}\ \ Q > \alpha \cdot \dfrac{\bar{w}}{N_{i-1}}, \end{cases} \tag{2}$$

where N_i is the value of N in the i-th updating period, $\bar{w}$ is the average window size of the i-th updating period and α is a parameter and $0 < \alpha < 1$. Q is an estimate of the number of in-flight packets of the TCP-FIT session that are currently queued in the network buffer, and is calculated in a way that is similar to existing delay-based TCP and hybrid TCP variants [15]

$$Q = (\overline{rtt} - rtt_{min})\frac{\bar{w}}{\overline{rtt}}, \tag{3}$$

where $\overline{rtt}$ is the average RTT window size of the current updating period, rtt_{min} is the minimal observed RTT value used as a reasonable estimate of the propagation delay of the network. $(\overline{rtt} - rtt_{min})$ represents the estimated value of the queuing delay. Since a TCP session sends w packets during a RTT, $\bar{w}/\overline{rtt}$ could be considered as an estimate of packet transmission rate of the current TCP session.

Since w of a TCP-FIT session is N times of a Reno session, it is straightforward to justify that the purpose of adjusting N of (2) is letting the TCP-FIT session worked at a steady state where Q of the TCP-FIT session is proportional to a Reno session. We discuss how to set the proportionality factor α in Sections V. Combining (3) with (2), we get at the steady state, the average value of N is

$$E[N] = \alpha \cdot \frac{\overline{rtt}}{\overline{rtt} - rtt_{min}}. \tag{4}$$

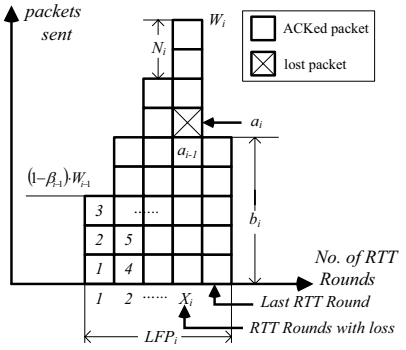


Fig. 2. Packets sent during a Loss Free Period

TABLE I
NOTATION OF LFP

Parameter	Definition
LFP_i	i-th Loss Free Period
W_i	The window size at the end of the LFP_i
N_i	The number of virtual sessions in LFP_i
a_i	The first packet lost in LFP_i
X_i	The RTT round where loss a_i occurs
b_i	The number of packets sent in the last RTT round
β_i	The window decrease factor.

IV. THROUGHPUT MODEL OF TCP-FIT

In this section, we introduce an approximate stable state throughput model for TCP-FIT. The model follows Padhye's well-known Reno throughput model [26], and assumes that all loss events are detected by triple-duplicate ACKs and not consider the slow start and fast retransmission process.

As illustrated in Fig. 2, we define a Loss Free Period (LFP) as a period between two consecutive packet losses indicated by triple-duplicate ACKs. The notations used in this section are summarized in Table I. The evolution of TCP-FIT congestion control window after the slow-start phase can be viewed as a series of statistically i.i.d. LFPs. Each LFP is comprised of a number of RTT rounds. The congestion window W_i of LFP_i is increased by N_i after each RTT and decreased by a factor of $(1 - \beta_i)$ after each packet loss. Here $\beta_i = \frac{1}{2N_i}$ in MulTCP and $\beta_i = \frac{2}{3N_i+1}$ in TCP-FIT.

For LFP_i of duration A_i, we define Y_i as the number of packets sent. Then the expected steady-state TCP-FIT throughput is

$$T = \frac{E[Y]}{E[A]}. \tag{5}$$

As shown in Fig. 2, a total of $Y_i = a_i + W_i - 1$ packets are sent in $X_i + 1$ rounds. Following the assumption in [26] that

$E[a] = 1/g$, where g is end-to-end packet loss rate of the TCP-FIT session, the expectation of Y can be calculated as

$$E[Y] = E[a] + E[W] - 1 = \frac{1-g}{g} + E[W]. \tag{6}$$

Because the value of the congestion window in LFP_i is book-ended by $(1-\beta_{i-1})W_{i-1}$ and W_i, with an increment of N_i after each RTT in LFP_i, we have

$$\begin{aligned} Y_i &= \sum_{k=0}^{X_i-1} ((1-\beta_{i-1})W_{i-1} + kN_i) + b_i \\ &= (1-\beta_{i-1})W_{i-1}X_i + \frac{X_iN_i}{2}(X_i-1) + b_i, \end{aligned} \tag{7}$$

and

$$W_i = (1-\beta_{i-1})W_{i-1} + N_iX_i. \tag{8}$$

From (6) and (7) we have

$$\begin{aligned} &\frac{1-g}{g} + E[W] \\ &= E[X]\left((1-E[\beta])E[W] + \frac{E[N](E[X]-1)}{2}\right) + E[b], \end{aligned} \tag{9}$$

while from (8) we have

$$E[X] = \frac{E[\beta]E[W]}{E[N]}. \tag{10}$$

Similar to [26], we assume that b_i's are uniformly distributed between 1 and W_i and therefore $E[b] = E[W]/2$. Then from (9) and (10) we have

$$E[W] = \frac{\frac{E[\beta]+1}{2} + \sqrt{(\frac{E[\beta]+1}{2})^2 + 4\frac{1-g}{g}\frac{2E[\beta]-E[\beta]^2}{2E[N]}}}{(2E[\beta]-E[\beta]^2)/E[N]}, \tag{11}$$

and

$$E[X] = \frac{\frac{E[\beta]+1}{2} + \sqrt{(\frac{E[\beta]+1}{2})^2 + 4\frac{1-g}{g}\frac{2E[\beta]-E[\beta]^2}{2E[N]}}}{2-E[\beta]}. \tag{12}$$

Let $r_{i,j}$ be the randomly distributed value of the j-th RTT of LFP_i, with the duration of LFP_i $A_i = \sum_{j=1}^{X_i+1} r_{i,j}$. Assuming as in [26] that $r_{i,j}$ are independent of the size of the congestion window, and therefore independent of j, we have

$$E[A] = (E[X]+1)E[r]. \tag{13}$$

From (12) and (13) we have

$$E[A] = (\frac{\frac{E[\beta]+1}{2} + \sqrt{(\frac{E[\beta]+1}{2})^2 + 4\frac{1-g}{g}\frac{2E[\beta]-E[\beta]^2}{2E[N]}}}{2-E[\beta]} + 1)E[r] \tag{14}$$

Combining (5), (6), (11) and (14) we have

$$\begin{aligned} T &= \frac{\frac{1-g}{g} + E[W]}{E[A]} \\ &= \frac{\frac{1-g}{g} + \frac{\frac{E[\beta]+1}{2} + \sqrt{(\frac{E[\beta]+1}{2})^2 + 4\frac{1-g}{g}\frac{2E[\beta]-E[\beta]^2}{2E[N]}}}{(2E[\beta]-E[\beta]^2)/E[N]}}{(\frac{\frac{E[\beta]+1}{2} + \sqrt{(\frac{E[\beta]+1}{2})^2 + 4\frac{1-g}{g}\frac{2E[\beta]-E[\beta]^2}{2E[N]}}}{2-E[\beta]} + 1)E[r]}. \end{aligned} \tag{15}$$

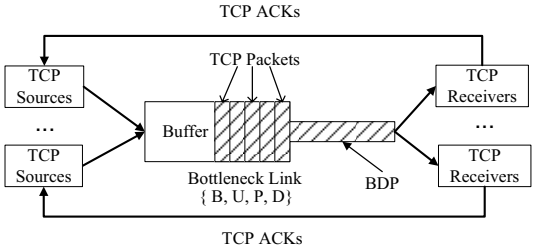


Fig. 3. Topology of the network model

When g is sufficiently small,

$$T = \frac{1}{E[r]}\sqrt{\frac{(2-E[\beta])E[N]}{2gE[\beta]}} + o(1/\sqrt{g}). \tag{16}$$

Let $E[r] = D + q$, where D is the average propagation delay and q is the average of queuing delay. Then from (4)

$$E[N] = \max\left\{1, \ \frac{\alpha(D+q)}{q}\right\}. \tag{17}$$

Let $g = P + p$, where P corresponds to the inherent losses of the link and p for losses caused by congestion. Consider that $\beta_i = \frac{2}{3N_i+1}$,

$$T = \frac{E[N]}{D+q}\sqrt{\frac{3}{2(P+p)}} + o(1/\sqrt{P+p}). \tag{18}$$

Compared with Reno's throughput in Table II [26], (18) is $E[N]$ times of throughput of Reno. Combining (17) and (18), the throughput model of TCP-FIT is approximately expressed as

$$T = \max\left\{\frac{1}{D+q}\sqrt{\frac{3}{2(P+p)}}, \frac{\alpha}{q}\sqrt{\frac{3}{2(P+p)}}\right\}. \tag{19}$$

V. Efficiency of TCP-FIT

In this section, we use a simple but widely adopted network scenario to analyze three aspects of the performance of TCP-FIT: network utilization, rtt-fairness and inter-fairness.

As illustrated in Fig. 3, we assume that the network K includes one bottleneck link which has a bandwidth limit of B, buffer size of U, and round-trip propagation delay D, an inherent random packet loss rate of P and several non-bottleneck links with unlimited bandwidth and different round-trip propagation delays. In the network model, n TCP sessions share the bottleneck link at the same time but may have different non-bottleneck routing paths. These TCP sessions may use different TCP congestion control algorithms. When these TCP sessions send packets through the network, we assume there are on average M packets traversing the bottleneck. The relationship between M, the congestion packet loss rate p, and the queuing delay q can be approximated using:

$$\begin{cases} p = 0, \ q = 0, & \text{if } 0 \le M < B \cdot D \quad (a) \\ p = 0, \ q > 0, & \text{if } B \cdot D \le M \le B \cdot D + U \quad (b) \\ p > 0, \ q \to \infty, & \text{if } B \cdot D + U < M \quad (c) \end{cases} \tag{20}$$

In the state (a) of (20), M is smaller than the link bandwidth-delay product (BDP), no packets are queued in the bottleneck buffer and no congestion packet loss occurs. If M is higher than BDP but smaller than the BDP plus bottleneck buffer size U, which is state (b), the bottleneck bandwidth is fully utilized and packets begin to queue in the buffer. As a result, queuing delay q begins to increase but no packets are dropped due to network congestion. In state (c), when M is higher than $B \cdot D + U$, the bottleneck buffer will overflow and some packets will be lost due to congestion, and queuing delay is effectively infinite.

Obviously, the best state of operation is (b) when the bandwidth resource of the bottleneck is fully utilized with no congestion related packet losses. The function of the TCP congestion control algorithm is to adjust the packet sending rate so that the corresponding session operates in this state while using only a fair share of the resources.

We list the throughput models of several typical TCP variants in Table II using the results of [26] [8] [27] [28] [29] [10] and the analysis in the previous section. All of these models follow the same assumptions of (19) based on.

A. Network Utilization

Theorem 1: The bottleneck link of a network K depicted as in Fig. 3 with a TCP-FIT session works in state (b) of (20).

Proof. It is obvious that the throughput of TCP-FIT $T < B$.

- Suppose the bottleneck is in state (a), where $q = 0$. From (19) the throughput of TCP-FIT $T \to \infty$. This contradicts with $T < B$.
- If the bottleneck is in state (c), then $q \to \infty$. From (19) and throughput models in Table II we get $T \to 0$ and therefore $M \to 0$, which means that the bottleneck is in state (a). ■

Theorem 1 shows that TCP-FIT can fully utilize the network capacity. Similar conclusions can be proved for TCP Vegas and FAST but not for other TCP variants. We use the throughput model of TCP Reno in Table II [26] as an example. Since $p \geq 0$ and $q \geq 0$, there is an upper bound for the throughput of a Reno session for any given P and D that is not a function of B:

$$T = \frac{1}{D+q}\sqrt{\frac{3}{2(P+p)}} \leq \frac{1}{D}\sqrt{\frac{3}{2P}}.$$

Similar upper bounds for the throughputs of different TCP algorithms except for TCP-FIT and Vegas/FAST are given in the third column of Table II. Because these upper bounds are independent of B, when the bottleneck link state of network K is sufficiently bad (i.e. having a high packet loss rate P and/or a very long propagation delay), or if the bandwidth B of bottleneck is sufficiently high, the aggregated throughput of the TCP sessions in K will be lower than the total bottleneck bandwidth B. Theoretically, therefore, except for TCP-FIT and Vegas/FAST TCP, other TCP algorithms designed for wireless/large-BDP networks could only mitigate but not completely solve the bandwidth utility problem in wireless/large-BDP environments. At least theoretically, compared with these TCP variants, TCP-FIT has an advantage in its capability of achieving higher throughputs.

B. Fairness

RTT-fairness: By definition in [30], RTT-fairness means that "flows with similar packet drop rates but different round-trip times would receive roughly the same throughput".

We again use TCP Reno as an example. Since different TCP sessions sharing the same bottleneck in K may traverse different non-bottleneck routes, their end-to-end propagation RTTs could be different. Using the throughput model of TCP Reno as an example, we can find the ratio of the throughputs for two sessions i and j

$$\eta = \frac{T_i}{T_j} = \frac{D_j + q}{D_i + q}. \tag{21}$$

If $D_i \neq D_j$, the session with a longer propagation delay will have a low throughput and therefore at a disadvantage. That leads to RTT-unfairness.

As shown in the last column of Table II, the TCP-FIT could achieve theoretical RTT-fairness when $E[N] > 1$. When $E[N] = 1$, the TCP-FIT has same RTT-fairness property with the Reno. A TCP congestion control algorithm can't be perfect with respect to both RTT-fairness and inter-fairness because the TCP Reno is not RTT-fair.

Inter-fairness: Inter-fairness, or TCP friendliness, refers to fairness between new TCP variants and standard TCP congestion control algorithm, the TCP Reno. We used the Bandwidth Stolen Rate (BSR) [8] to quantify the inter-fairness of TCP congestion control algorithms.

Let T^* be the aggregated throughput of m TCP Reno flows when they compete with another k TCP Reno flows. Let T' be the aggregated throughput of m TCP Reno flows when they compete with another k new TCP variant flows in the same network topology, then the BSR is defined as

$$BSR = \frac{T^* - T'}{T^*}. \tag{22}$$

To get a low BSR, the number of in-flight packets that are queued in bottleneck buffer for k TCP-FIT sessions must not be significantly higher than that for k Reno sessions. Recall that for TCP-FIT, the number of in-flight packets $Q = \alpha\frac{w}{N}$, this means that α should be set to the ratio between the number of in-flight packets that are queued in network buffer and the value of the congestion window for a Reno session to achieve inter-fairness. In a practical implementation, one can approximate this theoretical α value with the mean of each TCP-FIT session's $(rtt_{max} - rtt_{min})/rtt_{max}$, where rtt_{max} and rtt_{min} are the maximal and minimal end-to-end RTTs observed for the session. Experiment results show that both the throughput and fairness of the implementation when using this approximation are good.

VI. Experimental Results

In our experiments, we embedded TCP-FIT into the Linux kernel (v2.6.31) of the server. For comparisons, we used the

TABLE II
TCP MODELS

TCP Variants	Throughput Models	Upper Bounds	$\eta = T_i(D_i)/T_j(D_j)$
TCP-FIT	$T = \max\left\{\frac{1}{D+q}\sqrt{\frac{3}{2(P+p)}}, \frac{\alpha}{q}\sqrt{\frac{3}{2(P+p)}}\right\}$	$\times$	$\eta = 1$ (when $E[N] > 1$)
Reno	$T = \frac{1}{D+q}\sqrt{\frac{3}{2(P+p)}}$ [26]	$T \le \frac{1}{D}\sqrt{\frac{3}{2P}}$	$\eta = \frac{D_j+q}{D_i+q}$
CTCP	$T = \frac{1}{D+q}\frac{\Lambda}{(P+p)^{\frac{1}{2-k}}}$, $\Lambda = \frac{(\frac{1-(1-\beta')^{2-k}}{2-k})^{\frac{1-k}{2-k}}}{\alpha'^{\frac{1}{2-k}}(1-(1-\beta')^{1-k})}$ where, α', β' and k are preset parameters. [8]	$T \le \frac{1}{D}\frac{\Lambda}{P^{\frac{1}{2-k}}}$	$\eta \le [\frac{D_j+q}{D_i+q}]^2$ [8]
Veno	$T = \frac{1}{D+q}\sqrt{\frac{1+\gamma'}{2(1-\gamma')(P+p)}}$, where $\frac{1}{2} \le \gamma' \le \frac{4}{5}$. [27]	$T \le \frac{1}{D}\sqrt{\frac{1+\gamma'}{2(1-\gamma')P}}$	$\eta = \frac{(D_j+q)\sqrt{(1+\gamma_i')(2(1-\gamma_j')(P+p))}}{(D_i+q)\sqrt{(1+\gamma_j')(2(1-\gamma_i')(P+p))}}$
Westwood	$T = \frac{1}{\sqrt{(D+q)((P+p)D+q)}}\sqrt{\frac{(1-(P+p))}{P+p}}$ [28]	$T \le \frac{1}{DP}\sqrt{1-P}$	$\eta = \sqrt{\frac{(D_j+q)((P+p)D_j)+q}{(D_i+q)((P+p)D_i)+q}}$
Vegas/FAST	$T = \alpha''/q$, where α'' is preset parameter. [29] [10]	$\times$	$\eta = 1$

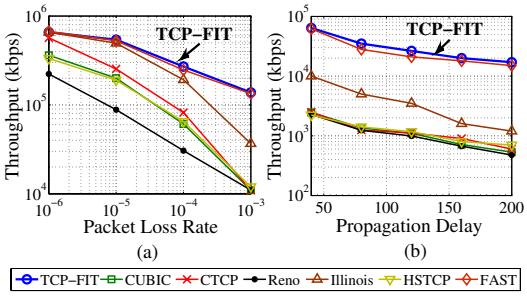


Fig. 4. Performance of the different TCP variants over a high speed link

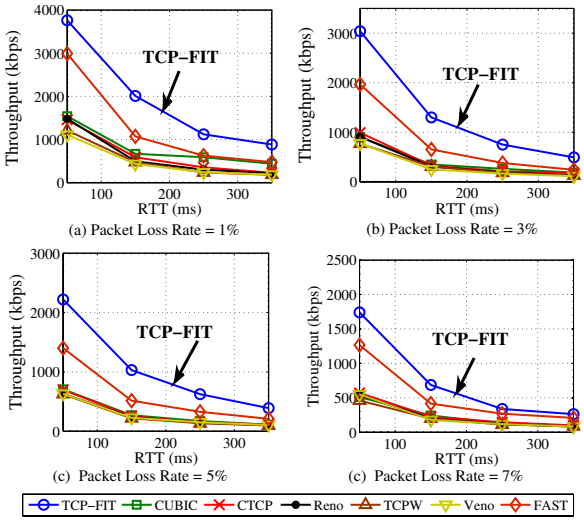


Fig. 5. Performance of different TCP variants over a wirless link.

Compound TCP for Linux from CalTech [31] and implemented the FAST TCP algorithm based on the ns-2 code of FAST TCP [32]. Other TCP variants are available on the Linux kernel v2.6.31, including:

- TCP Reno and TCP CUBIC;
- TCP Westwood (TCPW) and TCP Veno, as benchmark of optimized TCP congestion control algorithms for wireless links;
- Highspeed TCP (HSTCP) and TCP Illinois as benchmark of optimized TCP congestion control algorithms for high BDP links.

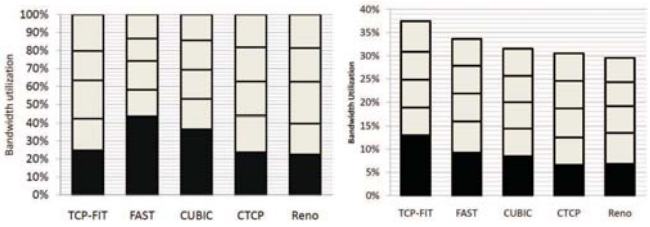
A. Emulation-Based Experiments

In our experiments, we first conducted comparisons between different TCP algorithms using network emulators. In the experiments, a TCP server and a TCP client were connected through the emulator, which injected random packet losses and delays into the connection and capped the network bandwidth to a selected value.

We first compared the performance of the different TCP algorithms for high BDP networks. In the experiments, we compared TCP-FIT with Reno, CUBIC, CTCP as well as HSTCP, Illinois and FAST for high BDP networks. Following [33], the value of α'' of FAST was set to 200 for Gbps links and 20 for 100 Mbps links.

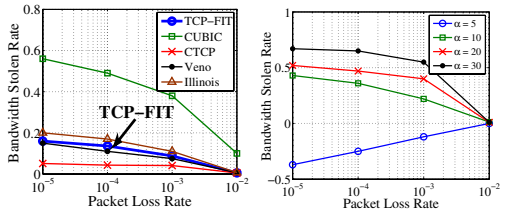
Fig. 4(a) shows the resulted throughput comparison for a 1 Gbps link. We set the propagation delay to 20 ms, and varied the packet loss rate from 10^{-3} to 10^{-6} by a dummynet [34] emulator. As can be seen from the figure, TCP-FIT achieved similar throughput as FAST TCP, which was much higher than other TCP algorithms. In Fig. 4(b), we set the bandwidth to 100 Mbps with a packet loss rate of 1% using a Linktropy [35] hardware network emulator. We varied the propagation delay from 40 ms to 200 ms. Again, TCP-FIT achieved higher throughput than other TCP variants.

To compare the performances of the algorithm in the presence of wireless losses, we compared TCP-FIT with Reno, CUBIC, CTCP and as well as TCP Westwood and Veno, which were originally designed with wireless networks in mind. Although FAST was not originally designed specifically for wireless links, we also included it in the tests using fine-tuned α'' values found by extensive trial and error. The link



(a) Packet Loss Rate = 0% (b) Packet Loss Rate = 5%

Fig. 6. Bandwidth occupation among competing TCP sessions



(a) BSR of different TCP variants (b) BSR of FAST with different α''

Fig. 7. Bandwidth Stolen Rate comparison of different TCP variants

TABLE III
THROUGHPUT RATIO WITH DIFFERENT PROPAGATION DELAY

RTT ratio	1	1/2	1/4	1/8
TCP-FIT	1	1.17	1.55	1.9
FAST	1	1.03	1.45	1.93
CUBIC	1	1.34	1.64	2.14
CTCP	1	1.37	2.17	2.83
Reno	1	1.84	3	5.19

TABLE IV
EXPERIMENTAL ENVIRONMENTS

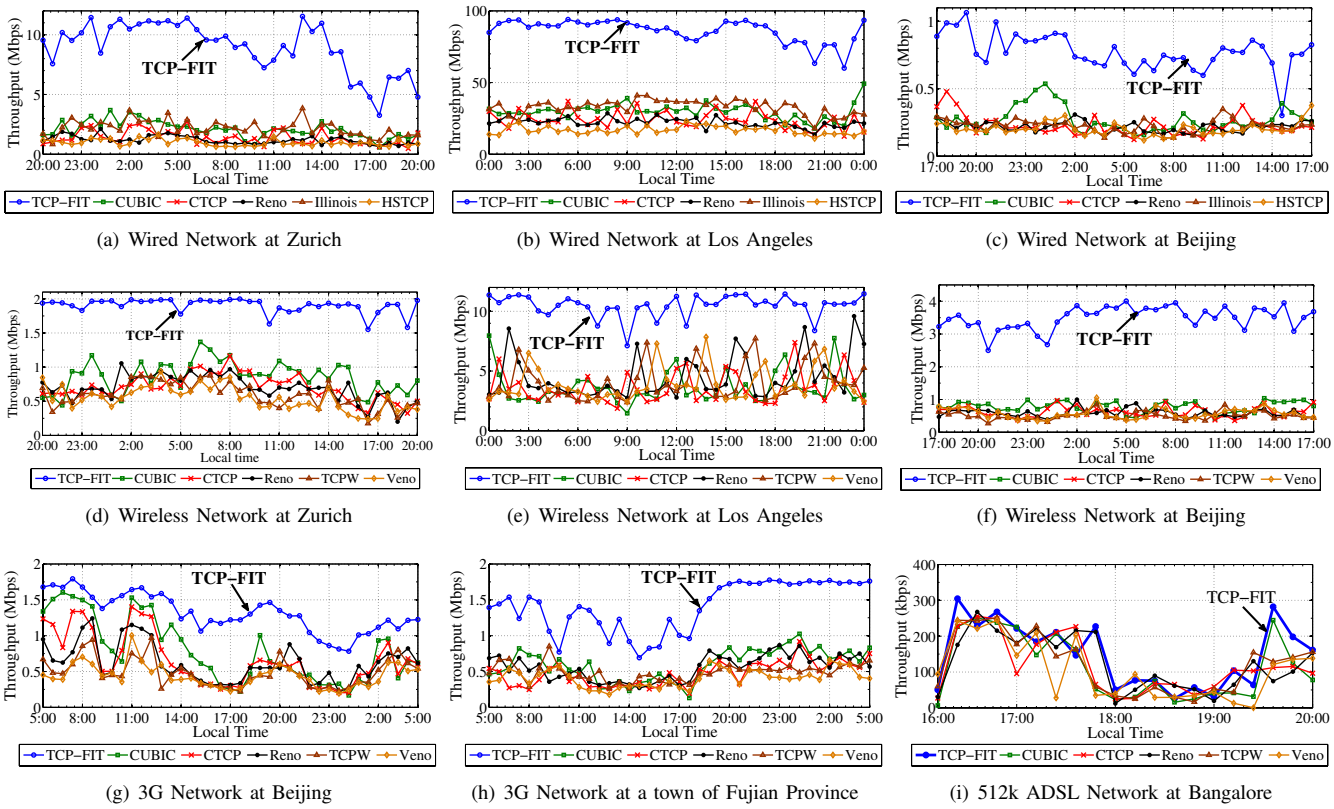
Figures	Location	Network	Client OS
Fig. 8(a)	Zurich	Ethernet	Win Vista
Fig. 8(b)	LA	Ethernet	Win Vista
Fig. 8(c)	Beijing	2M ADSL	Win XP
Fig. 8(d)	Zurich	WIFI	MAC OS
Fig. 8(e)	LA	WIFI	Win Vista
Fig. 8(f)	Beijing	WIFI	Linux
Fig. 8(g)	Beijing	CDMA 2000	Win XP
Fig. 8(h)	Fujian	CDMA 2000	Win Vista
Fig. 8(i)	Bangalore	512k ADSL	Win Vista

bandwidth was set to 4 Mbps on a Linktropy emulator. In Fig. 5, the packet loss rate was set from 1% to 7%. In each experiment, we varied the propagation delay of the link from 50 ms to 350 ms. To simulate the random delay fluctuation of wireless link, Gaussian distributed delay noise were generated by Linktropy emulator. The standard deviation of delay noise was 50% of the mean RTT. As can be seen from the figures, TCP-FIT achieved much higher throughput than other TCP congestion control algorithms, including FAST. Compared with FAST, which doesn't react to packet loss, TCP-FIT are more robust to random delay fluctuation of wireless link in the experiments.

Fig. 6 and Fig. 7 are the results from inter-fairness experiments. In Fig. 6(a), one connection of a certain TCP algorithm is tested when competing with four TCP Reno sessions. The combined bandwidth was set to 10 Mbps, with a propagation delay of 10 ms and no random packet losses. As shown in Fig. 6(a), both TCP-FIT and Compound TCP occupied about 20% of the total bandwidth, i.e. same as when the algorithm is replaced with a 5th Reno session, or the theoretical "fair share". FAST TCP and TCP CUBIC on the other hand, occupied up to 35% bandwidth, which means these algorithms might be overly aggressive and not inter-fair/TCP-friendly. Similar experiments were conducted for Fig. 6(b), with the total network bandwidth still set to 10 Mbps but with 5% packet loss, and Gaussian distributed propagation delay with a mean of 100 ms, and 80 ms standard deviation. In this case, as a result of the worsened network conditions, the TCP sessions combined were only able to use less than 40% of the the network bandwidth. TCP-FIT in this case was able to pick up some of the capacity that the other Reno sessions left unused, and it is important to also note that the percentages (i.e. between 5% and 10%) of the bandwidth that the other Reno sessions used were still comparable to their respective numbers when 5 Reno sessions were used, indicating that the additional throughput that TCP-FIT was able to "grab" did not come at the expense of the Reno sessions.

Fig. 7(a) shows the Bandwidth Stolen Rate. According to the definition of BSR in (22), in the experiments, we first ran 20 Reno sessions over a 50 Mbps, 100 ms delay link and then replaced 10 Reno sessions with 10 sessions of a different TCP variant. The ideal value of BSR is of course 0. It is shown in Fig. 7(a) that the BSR of TCP-FIT is a little higher than Compound TCP, which is well-known to have extremely good inter fairness characteristics, but much lower than TCP CUBIC. Compared with other wireless and high speed network optimized algorithms, such as Veno and Illinois, the inter-fairness property of TCP-FIT is at a reasonable level. BSRs for FAST under the same link conditions are shown in Fig. 7(b). The inter-fairness of FAST depends on the selection of parameters α''.

Table III compares the RTT-fairness of TCP algorithms. In the experiment, 20 TCP sessions were divided into two groups of 10 to compete for a 50 Mbps link. One of the groups had a shorter delay of 50 ms. The other group had longer delays that varied among 50 ms, 100 ms, 200 ms, and 400 ms. The packet loss rate of the bottleneck link was 0.1%. Table. III summarizes the throughput ratios between sessions with different propagation delays. The ideal ratio should be 1. As shown in Table. III , the RTT-fairness property of TCP-FIT is similar to FAST TCP and CUBIC, and better than other algorithms.



(a) Wired Network at Zurich (b) Wired Network at Los Angeles (c) Wired Network at Beijing

(d) Wireless Network at Zurich (e) Wireless Network at Los Angeles (f) Wireless Network at Beijing

(g) 3G Network at Beijing (h) 3G Network at a town of Fujian Province (i) 512k ADSL Network at Bangalore

Fig. 8. Throughput Variation at Different Locations

B. Performance Measured over Live Networks

Finally, to test the performance of TCP-FIT over live, deployed, real-world networks, we put a server in Orange County, California, US, on the Internet and tested TCP-FIT using clients located in 5 different cities/towns in 4 countries (China, Switzerland, USA and India) on 3 continents. At each location, we tested a combination of wired line connections, Wi-Fi and whenever possible, 3G wireless networks. The location, network and OS of the test clients are listed in Table IV. In each experiment, a script was used to automatically and periodically cycle through different TCP algorithms on the server over long durations of time (4-24 hours), while the client collected throughput information and other useful data. The period for changing the TCP algorithms was set to about 5-10 minutes, so that 1) the algorithms tested were able to reach close to steady-state performances; 2) the period is consistent with the durations of a reasonably large percentage of the TCP based sessions on the Internet (e.g. YouTube streaming of a single piece of content, synchronizing emails, refreshing one web page, etc.). In the performance measure, TCP-FIT are compared with CUBIC, CTCP, Reno in all cases. HSTCP and Illinois are compared for wired networks and TCPW, Veno for wireless.

Our results are summarized in Table V and Fig (8). Throughout the experiments, TCP-FIT achieved speedup ratios up to 5.8x compared with average throughput of other TCP congestion control algorithms.

TABLE V
AVERAGE THROUGHPUT (MBIT/S)

Network	Wired Network			WIFI			CDMA 200	
Location	ZRH	LAX	PEK	ZRH	LAX	PEK	City	Town
FIT	8.9	86	0.8	1.9	10.3	3.4	1.3	1.4
CUBIC	2.1	29	0.3	0.9	3.5	0.8	0.8	0.4
CTCP	1.4	23	0.2	0.7	3.5	0.6	0.7	0.6
Reno	1.2	22	0.2	0.7	4.5	0.6	0.7	0.6
TCPW	x	x	x	0.6	4.1	0.5	0.5	0.4
Veno	x	x	x	0.6	3.8	0.6	0.6	0.5
Illinois	2.1	32	0.22	x	x	x	x	x
HSTCP	1.0	16	0.2	x	x	x	x	x
Speedup	5.8	3.5	3.4	2.7	2.6	5.7	2.0	2.8
BSR	0.12	0.07	0.2	0.07	0.09	0.05	0	0

The throughput of TCP-FIT was not always higher than other TCP algorithms. In Fig. 8(i), when clients accessed the server through a low speed ADSL network with large bandwidth variations, TCP-FIT had no obvious advantage compared with other TCP variants during a 4-hour period. This was probably due to the fact that, compared with other TCP algorithms such as FAST which use very advanced network condition estimation algorithms, the corresponding modules in FIT were still relatively simplistic. For networks

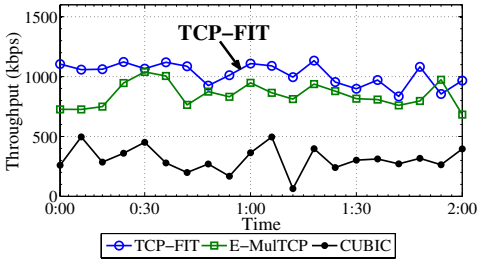


Fig. 9. Throughput comparison between TCP-FIT and E-MulTCP over live 3G network in Beijing.

with large variations, this might lead to TCP-FIT performance degradations as the settings of key algorithm parameters such as Q depends on such estimates.

To confirm that the performance gains of TCP-FIT did not come from grabbing bandwidth of other TCP sessions, we also measured the Bandwidth Stolen Rate of different TCP variants. The results are listed in the last row of Table V. As shown in the table, the BSR for TCP-FIT remained low.

Finally, Fig. 9 shows a comparison between TCP-FIT and E-MulTCP as measured over China Telecom's 3G network in Beijing. Although strictly speaking E-MulTCP is not a TCP congestion algorithm per se, TCP-FIT was still able to achieve an average of about 27% improvement.

VII. Conclusions

In this paper, we describe a novel TCP congestion control algoithm for application in both high BDP and wireless networks. The algorithm was motivated by parallel TCP techniques, but is fully TCP compliant and requires no changes to other layers of the protocol stack and/or application software. Theoretical and emulation results as well as performances measured using live real world networks over 3 continents show significant performance improvements in throughput, fairness or robustness (e.g. against network RTT variations) over state-of-the-art TCP variants such as TCP Reno, CUBIC, CTCP, Westwood, Illinois and FAST TCP.

References

[1] J. Postel, "Transmission control protocol," *RFC 793*, September 1981.

[2] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM Computer Communication Review*, vol. 25, no. 1, pp. 157–187, January 1995.

[3] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control," *RFC 2581*, April 1999.

[4] S. Floyd and T. Henderson, "The new reno modification to TCPs fast recovery algorithm," *RFC 2582*, April 1999.

[5] S. Mascolo, C. Casetti, M. Gerla, M. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proceedings of ACM MobiCom'01*, July 2001, pp. 287–297.

[6] R. Ferorelli, L. Grieco, S. Mascolo, G. Piscitelli, and P. Camarda, "Live Internet measurements using Westwood+ TCP congestion control," in *Proceedings of IEEE GLOBECOM'02*, vol. 3, November 2002, pp. 2583–2587.

[7] C. Fu and S. Liew, "TCP Veno: TCP enhancement for transmission over wireless access networks," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 2, pp. 216–228, February 2003.

[8] 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*, April 2006, pp. 1–12.

[9] 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, February 2008.

[10] D. Wei, C. Jin, S. Low, and S. Hegde, "FAST TCP: motivation, architecture, algorithms, performance," *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1246–1259, 2006.

[11] S. Gorinsky, M. Georg, M. Podlesny, and C. Jechlitschek, "A theory of load adjustments and its implications for congestion control," *Journal of Internet Engineering*, vol. 1, no. 2, October 2007.

[12] L. Xu, K. Harfoush, and I. Rhee, "Binary Increase Congestion Control (BIC) for fast, long distance networks," in *Proceedings of IEEE INFOCOM'04*, March 2004, pp. 2514–2524.

[13] S. Floyd, "HighSpeed TCP for large congestion windows," *RFC 3649*, December 2003.

[14] T. Kelly, "Scalable TCP: improving performance in highspeed wide area networks," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 2, pp. 83–91, April 2003.

[15] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: new techniques for congestion detection and avoidance," in *Proceedings of ACM SIGCOMM'02*. ACM, August 31-September 02 1994, pp. 24–35.

[16] S. H. Low, L. L. Peterson, and L. Wang, "Understanding TCP Vegas: a duality model," *Journal of the ACM*, vol. 49, no. 2, pp. 207–235, 2002.

[17] S. Liu, T. Basar, and R. Srikant, "TCP-Illinois: A loss-and delay-based congestion control algorithm for high-speed networks," *Elsevier Performance Evaluation*, vol. 65, no. 6-7, pp. 417–440, 2008.

[18] R. Shorten and D. Leith, "H-TCP: TCP for high-speed and long-distance networks," in *Proceedings of PFLDnet'04*, February 2004, pp. 95–101.

[19] K. Kaneko, T. Fujikawa, Z. Su, and J. Katto, "TCP-Fusion: A hybrid congestion control algorithm for high-speed networkss," in *Proceedings of PFLDnet'07*, February 2007, pp. 322–329.

[20] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The globus striped GridFTP framework and server," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, 2005, p. 54.

[21] M. Chen and A. Zakhor, "Flow control over wireless network and application layer implementation," in *Proceedings of IEEE INFOCOM'06*, March 2006, pp. 103–113.

[22] E. Altman, D. Barman, B. Tuffin, and M. Vojnovic, "Parallel TCP sockets: Simple model, throughput and validation," in *Proceedings of the IEEE INFOCOM'06*, March 2006, pp. 1–12.

[23] R. Chakravorty, S. Katti, J. Crowcroft, and I. Pratt, "Flow aggregation for enhanced TCP over wide-area wireless," in *Proceedings of the IEEE INFOCOM'03*, vol. 3, March 2003, pp. 1754–1764.

[24] J. Crowcroft and P. Oechslin, "Differentiated end-to-end Internet services using a weighted proportional fair sharing TCP," *ACM SIGCOMM Computer Communication Review*, vol. 28, no. 3, pp. 53–69, July 1998.

[25] M. Chen and A. Zakhor, "Rate control for streaming video over wireless," *IEEE Wireless Communications*, vol. 12, no. 4, pp. 32–41, 2005.

[26] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," in *Proceedings of the ACM SIGCOMM'98*, 1998, pp. 303–314.

[27] B. Zhou, C. Fu, D. Chiu, C. Lau, and L. Ngoh, "A simple throughput model for TCP Veno," in *Proceedings of IEEE ICC'06*, vol. 12, 2006, pp. 5395–5400.

[28] L. Grieco and S. Mascolo, "Mathematical analysis of Westwood+ TCP congestion control," in *Proceedings of IEE Control Theory and Applications*, vol. 152, no. 1, 2005, pp. 35–42.

[29] C. Samios and M. Vernon, "Modeling the throughput of TCP Vegas," in *Proceedings of ACM SIGMETRICS'03*, June 2003, pp. 71–81.

[30] S. Floyd and M. Allman, "Comments on the usefulness of simple best-effort traffic," *RFC 5290*, July 2008.

[31] L. Andrew. Compound TCP in Linux. [Online]. Available: http://netlab.caltech.edu/lachlan/ctcp

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

[33] D. X. W. Cheng Jin and S. H. Low. (2003, December) FAST TCP for high-speed long-distance networks. [Online]. Available: http://netlab.caltech.edu/pub/papers/draft-jwl-tcp-fast-01.txt

[34] L. Rizzo. Dummynet. [Online]. Available: http://info.iet.unipi.it/~luigi/ip_dummynet/

[35] Linktropy Mini2 WAN Emulator. [Online]. Available: http://www.apposite-tech.com/products/mini2.html