Journal of Network and Computer Applications 71 (2016) 167–180

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

TCP-FIT: An improved TCP algorithm for heterogeneous networks☆

Jingyuan Wang [a,*], Jiangtao Wen [b], Jun Zhang [b], Zhang Xiong [a,c], Yuxing Han [d]

[a] State Key Laboratory of Software Development Environment, School of Computer Science and Engineering, Beihang University, Beijing 100191, China
[b] Department of Computer Science, Tsinghua University, Beijing 100084, China
[c] Research Institute of Beihang University in Shenzhen, Shenzhen 518057, China
[d] TCPEngines Inc., Santa Clara, CA 95054, US

A R T I C L E I N F O

Article history:
Received 1 September 2015
Received in revised form
11 March 2016
Accepted 29 March 2016
Available online 1 April 2016

Keywords:
TCP
Congestion control
Wireless
Large bandwidth delay product

A B S T R A C T

Wireless networks and large bandwidth delay product (BDP) networks are two types of challenging environments for TCP congestion control. Many congestion control algorithms have been proposed to improve the performance of TCPs in these two environments. Although these improved algorithms can achieve remarkable performance enhancements for either of the two environments, designing a congestion algorithm that performs well over heterogeneous networks that contain both wireless and large BDP links remains a great challenge. In this study, we propose a novel congestion avoidance algorithm called TCP-FIT, which can perform excellently over both wireless and large BDP links while maintaining good fairness with the standard TCP Reno algorithm. To evaluate the proposed algorithm, theoretical analysis is presented for *equilibrium*, *network utilization*, *stability*, *TCP friendliness*, *RTT fairness*, and *responsiveness*. A series of experimental results obtained using network emulators and over the "live" Internet are also presented to demonstrate the significant performance and fairness improvements of TCP-FIT compared with other state-of-the-art algorithms.

1. Introduction

The Transmission Control Protocol (TCP) is a reliable transport layer protocol that is widely used on the Internet. Congestion control is an integral module of TCP that directly determines the performance of the protocol. The standard TCP congestion control framework includes four algorithms: "congestion avoidance", "slow start", "fast retransmit" and "fast recovery", among which congestion avoidance plays a crucial role in the performance of long-term TCP flows because it governs the steady-state behavior of the TCP congestion control window. TCP Reno (Jacobson, 1995), the de facto standard TCP algorithm for Internet congestion control, uses the additive increase and multiplicative decrease (AIMD) algorithm in its congestion avoidance, which achieved great success for several decades but has been found to perform poorly over wireless (Tian et al., 2005; Ghaffari, 2015) and large bandwidth delay product (BDP) links (Floyd, 2003; Kushwaha and Gupta, 2014). To improve the performance of TCP Reno over wireless and large BDP links, a substantial number of new TCP variants have been proposed, including TCP Westwood (Mascolo et al., 2001), and TCP Veno (Fu and Liew, 2003) for wireless applications and HighSpeed TCP (Floyd, 2003), Compound TCP (Tan et al., 2006), TCP CUBIC (Ha et al., 2008), FAST TCP (Wei et al., 2006) and HCC (Xu et al., 2011) for large BDP links.

Congestion control algorithms for high-speed large BDP links and high-loss wireless links are commonly considered as two separate research topics requiring different design philosophies and methodologies, leading to various algorithms focusing on only one of the two types of networks but not both. For example, TCP Veno and TCP Westwood can enhance TCP performance over high-loss wireless links but cannot fully adapt to the rapid growth of BDPs in emerging high-speed long-delay networks. HighSpeed TCP, Compound TCP, and TCP CUBIC achieve remarkable throughput improvements in large BDP links but cannot handle random packet losses of wireless links. However, with the deployment of high-speed long-distance networks, such as intercontinental optical networks and advanced high-speed wireless networks such as 4G LTE, the Internet has become increasingly heterogeneous. Therefore, designing a TCP congestion avoidance algorithm that can perform excellently over *both* large BDP and wireless connections becomes essential for current Internet congestion control.

In this paper, a novel TCP congestion avoidance algorithm, named TCP-FIT, for both large BDP and wireless links is proposed. Theoretical analysis from the *equilibrium, network utilization, stability, TCP friendliness, RTT fairness*, and *responsiveness* perspectives and extensive experimental results obtained using both network emulators as well as over the PlanetLab WAN testbeds show that

☆This paper is an extent version of Wang et al. (2011).
* Corresponding author.
E-mail address: jywang@buaa.edu.cn (J. Wang).

http://dx.doi.org/10.1016/j.jnca.2016.03.020
1084-8045/© 2016 Elsevier Ltd. All rights reserved.

TCP-FIT significantly improved TCP throughput over large BDP, packet loss, and RTT fluctuation environments compared with other state-of-the-art algorithms while maintaining good TCP friendliness, RTT fairness and responsiveness.

The rest of this paper is organized as follows. Section 2 gives an overview of existing TCP variants. Section 3 describes the TCP-FIT algorithm in detail. The throughput model of TCP-FIT is introduced in Section 4, and Section 5 provides theoretical analysis. Performance measured over network emulators and PlanetLab is given in Sections 6 and 7. Section 8 concludes the paper.

2. Related studies

Since the introduction of the first widely used TCP congestion control algorithm in Jacobson (1995), many TCP congestion control algorithms have been proposed. Table 1 contains a list of such algorithms widely found in mainstream operating systems, i.e., Windows and Linux. At a high level, these algorithms can be classified into three categories based on their input signals: namely, loss-based TCP, delay-based TCP and hybrid TCP.

Loss-based TCP includes TCP Reno, TCP Bic (Xu et al., 2004), TCP CUBIC (Ha et al., 2008), HighSpeed TCP (Floyd, 2003), Scalable TCP (Kelly, 2003), and TCP Hybla (Caini and Firrincieli, 2004). TCP Reno adopts an additive increase and multiplicative decrease (AIMD) congestion control window adjustment algorithm. TCP Bic adopts the binary window adjustment method, and TCP CUBIC improves it using a CUBIC function. HighSpeed TCP proposes a generalized AIMD where the linear increase factor and multiplicative decrease factor are adjusted by a convex function of the current congestion window size. Scalable TCP proposes using multiplicative increase and multiplicative decrease (MIMD) to meet features of high-speed networks. TCP Hybla scales the window increment rule to ensure fairness among the flows with different RTTs. Using 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 of loss-based TCP is that packet losses are caused by over-driving the network *only*, which is no longer valid for wireless networks. Random physical layer artifacts (e.g., multipath, interferences) introduced packet losses that have nothing to do with network congestion but will also cause loss-based TCP to aggressively reduce the window (Ahmed et al., 2015). This defect limits the application of loss-based TCP over emerged wireless access networks (Karrer et al., 2007).

Delay-based TCP algorithms include TCP Vegas (Bowker et al., 1994), FAST TCP (Wei et al., 2006) and ICTCP (Wu et al., 2013), which use the queuing delay of network links as the symptom for congestion. TCP Vegas is the first delay-based version proposed by Bowker et al. (1994). FAST TCP is a high-speed version of TCP Vegas that was designed through an equilibrium analysis. ICTCP adopts delay as a congestion symptom to avoid TCP incast in datacenter networks, which is other challenge network environment caused by small network buffer (Yu et al., 2015; Luo et al., 2015). The queuing delay is defined as a transmission delay increase of TCP sessions caused by in-flight packets queuing in network buffers. Delay-based TCP is more resilient to the wireless random packet losses and has potential in datacenter networks (Lee et al., 2012; Wang et al., 2015). The downside of this approach is the fairness issue. Because an increase in the queuing delay will not necessarily immediately lead to packet loss, when delay-based TCP shares the same bottlenecks with loss-based TCP, between the times when the delay starts to increase and the when packet loss occurs, the congestion control window for delay-based TCP will decrease while that for loss-based TCP flows will not, leading to bandwidth “starvation” for the delay-based flows (Wang et al., 2014).

Hybrid TCPs such as TCP Veno (Fu and Liew, 2003) and TCP Westwood (Mascolo et al., 2001), and DWTCP (Wang et al., 2013a) for wireless links and high-speed links such as Compound TCP (Tan et al., 2006) and H-TCP (Shorten and Leith, 2004) use both packet loss and queuing delay as inputs. TCP Veno adopts queuing delay as an index to adjust AIMD parameters for different random packet loss rates. TCP Westwood uses queuing delay to measure network bandwidth and thus avoid the impact of wireless packet losses to TCP congestion control. Compound TCP uses queuing delay to achieve high throughput and uses packet losses to maintain fairness with standard TCP Reno. H-TCP adopts delay jitters to adjust the parameters of AIMD. The performance of these TCP variants are good for the application scenarios for which they were originally designed. However, an emerging generation of high-bandwidth wireless networks and heterogeneous networks that contain segments of both large BDP and wireless links still presents challenges to such algorithms.

Parallel TCP is yet another research area of TCP congestion control. The core idea of parallel TCP is to build multiple TCP sessions that are controlled jointly to fully utilize available bandwidth. As reported in Chakravorty et al. (2003), Chen (2006), and Chen and Zakhor (2006), parallel TCP algorithms can achieve very good bandwidth utilization in large BDP and/or wireless networks. However, the deployment of parallel TCP requires monitoring of the context information of multiple TCP sessions and modifications to the application layer software. In contrast, MulTCP (Crowcroft and Oechslin, 1998) was designed to implement parallel TCP in the transport layer by simulating a fixed, preset number of parallel TCP flows in a single session. By fixing the number of parallel TCP flows, MulTCP encounters performance issues when the network bandwidth varies over time or if the presets are not appropriate for the end-to-end network. ECN also are adopted to improved TCP performance over data center networks (Alizadeh et al., 2011; Jingyuan et al., 2014).

Based on these insights, we proposed a novel hybrid TCP congestion control algorithm named TCP-FIT, which improved MulTCP by adopting an adaptive parallel flow number setting algorithm. Compared with existing hybrid TCP and MulTCP variants, TCP-FIT performs well in both wireless and large BDP networks. We explain the motivation of TCP-FIT in the next section.

Table 1
TCP Congestion control algorithm types.

Inputs	TCP Variants	Environment
Packet loss (Loss-based TCP)	TCP Reno (Jacobson, 1995)	Low speed wired
	TCP Bic (Xu et al., 2004)	Large BDP
	TCP CUBIC (Ha et al., 2008)	Large BDP
	HighSpeed TCP (Floyd, 2003)	Large BDP
	Scalable TCP (Kelly, 2003)	Large BDP
	TCP Hybla (Caini and Firrincieli, 2004)	Long delay
Queuing delay (Delay-based TCP)	TCP Vegas (Bowker et al., 1994)	Low speed wired
	FAST TCP (Wei et al., 2006)	Large BDP
Packet loss and queuing delay (Hybrid TCP)	Compound TCP (Tan et al., 2006)	Large BDP and TCP friendliness
	H-TCP (Shorten and Leith, 2004)	Large BDP
	TCP Veno (Fu and Liew, 2003)	Wireless
	TCP Westwood (Mascolo et al., 2001)	Wireless

3. The TCP-FIT algorithm

3.1. Motivation

We define a Packet Loss Event (PLE) as a series of packet losses that starts when the first packet loss is detected by three duplicate

ACKs in a window (which triggers the TCP Fast Recovery mechanism), and ends when the last lost packet recovered using Fast Recovery in the same window. The standard AIMD algorithm for adjusting the window w is

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

Here, we ignore losses that are detected by timeouts because w in timeout is not controlled by Congestion Avoidance. To simulate N TCP Reno flows in one session, MulTCP (Crowcroft and Oechslin, 1998) modifies the AIMD algorithm of Reno as

$$\begin{aligned}&\textit{Each} \quad \textit{RTT}: \quad w \leftarrow w + N,\\ &\textit{Each} \quad \textit{PLE}: \quad w \leftarrow w - \frac{w}{2N},\end{aligned} \tag{2}$$

which is approximatively equivalent to increasing w of N flows by one after each Round-Trip Time (RTT), and reducing w of one flow by half after a PLE.

We investigated the throughput and fairness of MulTCP in a simple scenario, in which a TCP Reno flow and a MulTCP flow shared a bottleneck link with a 10 Mbps bandwidth, 100 ms RTT and 1% random packet loss rate. Fig. 1(a) plots the aggregate throughput of the two flows when the N of MulTCP was set to different values. The figure shows that the Reno flow could not fully utilize the network bandwidth owing to random packet losses, and MulTCP could fully utilize the network bandwidth when N was set sufficiently large. However, the performance of a TCP algorithm must be evaluated by more metrics than the throughput alone. MulTCP flows with an excessively large value of N could cause throughput degradation of the TCP Reno flow. The TCP Reno throughput became increasingly lower while N became increasingly higher. The line with the cycle marker in Fig. 1(b) shows the throughput degradation rate of the Reno flow. When N was less than 10, the network capacity was not fully utilized. In this condition, the increased simulated TCP flows of MulTCP picked up the idle network capacity and did not occupy the bandwidth of the TCP Reno flow. Conversely, when N became larger than 10, the network bandwidth was fully utilized, and the network capacity should be equally divided by N simulated flows of MulTCP and a TCP Reno flow. In this condition, the throughput of the TCP Reno flow should be equal to the $1/(N+1)$ network capacity, and the throughput of the TCP Reno flow decreased with increasing parameter N i.e., number of simulated MulTCP flows. There was a throughput degradation of more than 50% in TCP Reno when N was larger than 10, which is unacceptable in most applications.

The results shown in Fig. 1(b) shed light on an algorithm for appropriately setting the value of N to remedy the fairness issues created by an inappropriately chosen N. The line in Fig. 1(b) with the box marker demonstrates the length of the in-flight packet queue of the MulTCP flow in the bottleneck buffer. We can see that there is a strong correlation between the queue length and the throughput degradation of the Reno flow. It is possible, therefore, to adjust N dynamically to fully utilize the network bandwidth while keeping the queuing delay at a reasonable level to preserve fairness. This experiment inspired us to propose the TCP-FIT algorithm.

3.2. The TCP-FIT algorithm

In Congestion Avoidance of TCP-FIT, the congestion control window is adjusted by

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

where N is a dynamic parameter that can be adjusted by TCP-FIT adaptively. According to the analysis in Section 4, the theoretical steady-state window size of (3) is N times of Reno flows for a given packet loss rate and RTT.

The target queue length of TCP-FIT flows are proportional to the average window size of TCP Reno flows, i.e.,

$$E[Q] = \alpha \cdot E[w_{reno}], \tag{4}$$

where $E[Q]$ is the queue length expectation of TCP-FIT flows, and $E[w_{reno}]$ is the window size expectation of TCP Reno flows. Both $E[Q]$ and $E[w_{reno}]$ are for the same packet loss rate and RTT. $\alpha \in (0, 1)$ in (4) is a dynamic parameter, which is closely related with the TCP Reno fairness of TCP-FIT, and we discuss α in Section 5.

Because the theoretical window size of TCP-FIT flows is N times of TCP Reno flows (according to the throughput model of TCP-FIT in Section 4), (4) can be expressed as

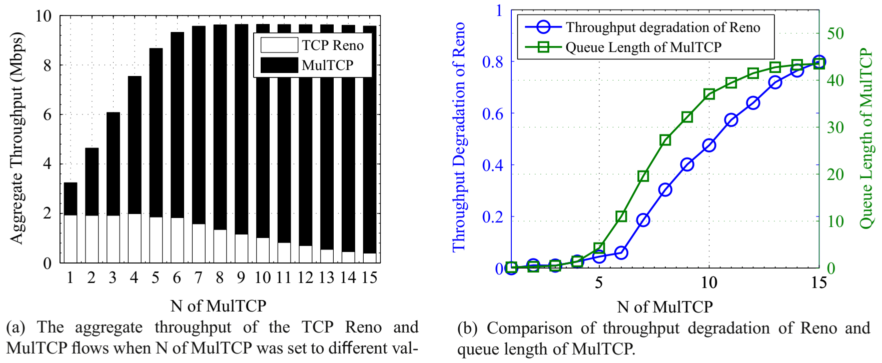
$$E[Q] = \alpha \cdot \frac{E[w]}{E[N]}, \tag{5}$$

where $E[w]$ and $E[N]$ are expectation of w and N for TCP-FIT flows respectively. To achieve the queue length in (5), TCP-FIT periodically updates the parameter N using

$$N(t+1) \leftarrow \max\left\{1, N(t) + 1 - \frac{Q(t)}{\alpha \cdot w_i} N(t)\right\}. \tag{6}$$

where $N(t)$, $w(t)$ and $Q(t)$ are the parameter N, window size and queue length of TCP-FIT in the t-th updating period, respectively. The setting of N updating period length is discussed in Section 5.

In order to facilitate implementations of the algorithm, TCP-FIT



(a) The aggregate throughput of the TCP Reno and MulTCP flows when N of MulTCP was set to different values.

(b) Comparison of throughput degradation of Reno and queue length of MulTCP.

Fig. 1. Performance and fairness of MulTCP. A TCP Reno flow and a MulTCP flow shared a bottleneck link with a 10 Mbps bandwidth, 100 ms RTT and 1% random packet loss rate.

uses

$$Q(t) = (RTT(t) - RTT_{\min})\cdot\frac{w(t)}{RTT(t)} \tag{7}$$

to estimate in-flight packet queue length, where $RTT(t)$ is average RTT of the t-th N updating period, and $RTT_{\min}$ is the minimal observed RTT used as an approximation of network propagation delay. Plugging (7) into (6), we have

$$N(t+1) \leftarrow \max\left\{1, N(t) + 1 - \frac{RTT(t) - RTT_{\min}}{\alpha\cdot RTT(t)}N(t)\right\}, \tag{8}$$

which is used for updating N in practical implementations.

4. Throughput model of TCP-FIT

In this section, we introduce an approximate throughput model for TCP-FIT following the notations of Padhye's TCP Reno throughput model (Padhye et al., 1998). In the model, we represent the AIMD algorithms in a general form, i.e.,

$$\begin{aligned} &Each \quad RTT: \; w \leftarrow w + a, \\ &Each \quad Loss: \; w \leftarrow w - b\cdot w. \end{aligned} \tag{9}$$

Obviously $a=N$, $b = \frac{2}{3N+1}$ for TCP-FIT, and $a=1$, $b = 1/2$ for TCP Reno. Fig. 2 illustrates the evolution of w when w is controlled by the generic AIMD algorithm in (9). The notations in Fig. 2 are summarized in Table 2.

We define the period between two consecutive PLE as a Loss Free Period (LFP). For the i-th LFP of duration A_i, the number of packets sent is denoted as Y_i. Then, the expected steady-state TCP packet throughput is

$$T_* = \frac{E[Y]}{E[A]}. \tag{10}$$

Because the congestion control window of LFP_i is increased by a_i for each RTT and decreased by a factor of $(1 - b_i)$ after a packet loss event, we have

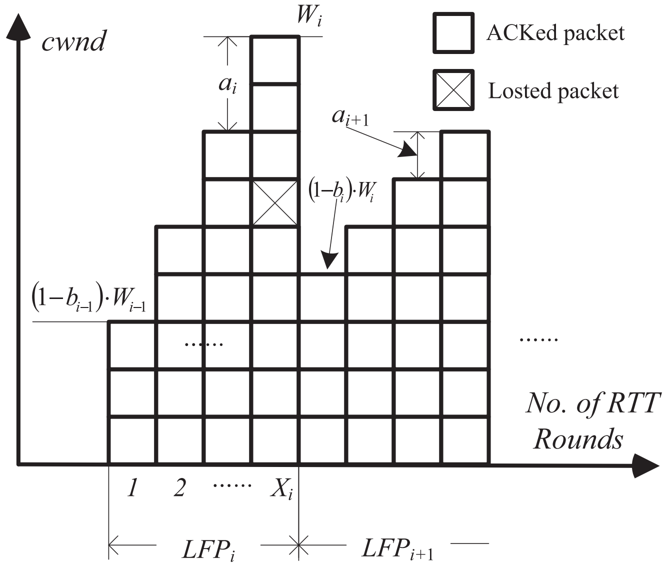


Fig. 2. A Padhye's TCP Reno throughput model analysis of the evolution of w of the generic AIMD algorithm.

Table 2
Notation of LFP.

Variables	Definition
LFP_i	i-th Loss Free Period
W_i	The w size at the end of the LFP_i
a_i	The increasing factor of the LFP_i
b_i	The decreasing factor of the LFP_i
X_i	The RTT round of the LFP_i where loss occurs

$$\begin{aligned} Y_i &= \sum_{k=0}^{X_i-1}((1 - b_{i-1})W_{i-1} + ka_i) \\ &= (1 - b_{i-1})W_{i-1}X_i + \frac{X_i a_i}{2}(X_i - 1). \end{aligned} \tag{11}$$

The congestion control window at the end of LFP_i is

$$W_i = (1 - b_{i-1})W_{i-1} + a_i X_i. \tag{12}$$

Merging (11) and (12), we have

$$Y_i = X_i\left(\frac{(1 - b_{i-1})W_{i-1} + W_i - a_i}{2}\right). \tag{13}$$

Assumption as in Padhye et al. (1998) that $E[Y] = \frac{1}{PLR}$, where PLR is the average packet loss event rate, then from (13), we have

$$\frac{1}{PLR} = E[X]\left(\frac{(2 - E[b])E[W] - E[a]}{2}\right). \tag{14}$$

According to (12), the expectation of X_i is

$$E[X] = \frac{E[b]E[W]}{E[a]}. \tag{15}$$

Plugging (15) into (14), we can solve $E[X]$ as

$$E[X] = \frac{E[b]}{2(2 - E[b])} + \sqrt{\left(\frac{E[b]}{2(2 - E[b])}\right)^2 + \frac{2E[b]}{PLR\cdot E[a](2 - E[b])}}. \tag{16}$$

Assuming that the PLR is sufficiently small, (16) can be expressed as

$$E[X] = \sqrt{\frac{2E[b]}{PLR\cdot E[a](2 - E[b])}} + o\left(\frac{1}{\sqrt{PLR}}\right). \tag{17}$$

Let $r_{i,j}$ be the randomly distributed value of the j-th RTT of LFP_i, with $A_i = \sum_{j=1}^{X_i} r_{i,j}$. Following assumptions in Padhye et al. (1998) that the values of the $r_{i,j}$ are independent of the congestion control window size, and therefore independent of j, we have

$$E[A] = E[X]E[r] = E[X]\cdot RTT, \tag{18}$$

Combining (10), (17) and (18),

$$\begin{aligned} T_* &= \frac{E[Y]}{E[A]} = \frac{1}{PLR\cdot E[A]} \\ &= \frac{1}{PLR\cdot RTT\left(\sqrt{\frac{2E[b]}{PLR\cdot E[a](2 - E[b])}} + o\left(\frac{1}{\sqrt{PLR}}\right)\right)}. \end{aligned} \tag{19}$$

Because $o(\frac{1}{\sqrt{PLR}})$ is an infinitesimal of higher order for $\frac{1}{\sqrt{PLR}}$, we can approximately write (19) as

$$T_* \approx \frac{1}{RTT}\sqrt{\frac{E[a](2 - E[b])}{2PLR\cdot E[b]}}. \tag{20}$$

According to the algorithm of TCP-FIT, we have

$$E[a] = E[N], \quad E[b] = \frac{2}{3E[N] - 1}. \tag{21}$$

Plugging (21) into (20), we have an approximate throughput model for TCP-FIT,

$$T_{fit} = \frac{E[N]}{RTT}\sqrt{\frac{3}{2PLR}}. \tag{22}$$

According to the algorithm of TCP Reno, we have

$$E[a] = 1E[b] = \frac{1}{2}. \tag{23}$$

Plugging (23) into (20), we have,

$$T_{reno} = \frac{1}{RTT}\sqrt{\frac{3}{2PLR}}. \tag{24}$$

Comparing (22) with (24), the throughput of TCP-FIT flows is $E[N]$ times that of TCP Reno flows with the same *PLR* and *RTT*.

According to (8), the expectation of N can be approximately expressed as

$$E[N] = \max\left\{1, \alpha\frac{RTT}{RTT - RTT_{\min}}\right\}, \tag{25}$$

and then we have a full expression of TCP-FIT' throughput model, i.e.,

$$T_{fit} = \max\left\{\frac{1}{RTT}, \frac{\alpha}{q}\right\}\cdot\sqrt{\frac{3}{2PLR}}, \tag{26}$$

where $q = RTT - RTT_{\min}$.

5. Efficiency of TCP-FIT

5.1. Network model

We study the efficiency of TCP-FIT in a widely used network model (Tang et al., 2007). The network model consist of L links, denoted by $l \in \{1, \ldots, L\}$, which are shared by S TCP sessions, denoted by $s \in \{1, \ldots, S\}$. The routing matrix $\mathbf{R}$ of the model is an $L \times S$ matrix, where

$$R_{l,s} = \begin{cases} 1 & \text{session } s \text{ traverse link } l \\ 0 & \text{otherwise.} \end{cases}$$

The notations used in the network model are summarized in Table 3. Among the notations, D_l and P_l are inherent to the link and are independent of congestion. Oppositely, q_l and p_l have a closely related with network congestion. We assume that q_l and B_l of the link l satisfies

Table 3
Notation of the network model.

Notations	Definition
B_l	Bandwidth of the link l
D_l	Propagation delay of the link l
P_l	Random packet loss rate of the link l
q_l	Queuing delay of the link l
p_l	Congestion packet loss rate of the link l
T_s	Throughput of TCP the session s
D_s	Propagation delay of the session s
P_s	Random packet loss rate of the session s
q_s	Queuing delay of the session s
p_s	Congestion packet loss rate of the session s
v_l	Queuing managing function of l, i.e., $p_l = v_l(q_l)$
m_l	Congestion measure price of l, i.e., $x_l = m_l(p_l, q_l)$

$$\sum_{R_{l,s}=1} T_s \begin{cases} = B_l & \text{if } q_l > 0(a) \\ \le B_l & \text{if } q_l = 0(b) \end{cases}, \tag{27}$$

The congestion packet loss $p_l = v(q_l)$ is a non-decreasing function of queuing delay q_l, and $v(0) = 0$. This assumption corresponds with queuing management algorithms of most switches and routers.

For the sake of discussion, we further define the end-to-end congestion packet loss rate p_s and the queuing delay q_s of the TCP session s as

$$p_s = \sum_{l\in L} R_{l,s}\cdot p_l, \quad q_s = \sum_{l\in L} R_{l,s}\cdot q_l.$$

Then, the end-to-end propagation RTT as well as the random packet loss of session s are

$$P_s = \sum_{l\in L} R_{l,s}\cdot P_l, \quad D_s = \sum_{l\in L} R_{l,s}\cdot D_l.$$

5.2. Existence of equilibrium

We first analyze equilibrium of TCP-FIT flows in the network model defined in Section 5.1. We assume that each link l has a congestion measure price x_l, which is a function of p_l and q_l, and define $x_l = m_l(p_l, q_l)$. The function $x_l = m_l(p_l, q_l)$ satisfies

1. $x_l = m(p_l, q_l)$ is a strictly increasing function of p_l and/or q_l;
2. $m(0, 0) = 0$.

For a given network (L, S, R), the throughput of session s is a function of the network state vector $x = (x_1, \ldots, x_l)$, i.e.,

$$T_s = f_s(x). \tag{28}$$

Let $\mathbf{X} = diag(x_l)$, $\mathbf{T} = (T_1, T_2, \ldots, T_S)^T$ and $\mathbf{B} = (B_1, B_2, \ldots, B_l)^T$. As defined in Tang et al. (2007), the network model reaches *equilibrium* when

$$\mathbf{RT} \le \mathbf{B}, \ \mathbf{X}(\mathbf{RT} - \mathbf{B}) = 0. \tag{29}$$

Theorem 1. *Equilibrium defined in (29) exists if following assumptions hold.*

A1. *$\forall\, s$, $T_s = f_s(x_l, x_{-l})$ is a non-increasing function of x_l for any fixed x_{-l}, where $x_{-l} = (x_1, \ldots, x_{l-1}, x_{l+1}, \ldots, x_L)^T$;*
A2. *$\forall\, \epsilon > 0$, there exists a $x_{\max}$ such that if $x_l > x_{\max}$ for l, then $T_s(x_l) < \epsilon$, $\forall\, s$ with $R_{l,s} = 1$.*

Theorem 1 is proved in the appendixes.

Theorem 2. *There exists equilibrium in the network model defined in Section 5.1 when the network contains TCP-FIT flows.*

Proof 1. We verify whether the assumptions A1 and A2 hold.

1. We first verify the assumption A1. We express the throughput function (28) of TCP-FIT as

$$\begin{aligned} T_s &= f_s((x_1, \ldots, x_l)) = f_s((m_l(p_1, q_1), \ldots, m_l(p_l, q_l))) \\ &= \max\left\{\frac{1}{D_s + q_s}\sqrt{\frac{3}{2(P_s + p_s)}}, \frac{\alpha}{q_s}\sqrt{\frac{3}{2(P_s + p_s)}}\right\}. \end{aligned} \tag{30}$$

Because $x_l = m(p_l, q_l)$ is a strictly increasing function, an increase in x_l is equivalent to an increase in p_l and/or q_l. Then, because $m(0, 0) = 0$ and (30), we have that

- T_s is a strictly decreasing function of x_l for any fixed x_{-l}, when $R_{l,s} = 1$.
- T_s is a constant for any fixed x_{-l}, when $R_{l,s} = 0$.

Therefore, T_s is a non-increasing function with regard to x_l for any fixed x_{-l}. The assumption A1 holds.

2. We then verify the assumption A2. $T_s = f_s(x_l)$ of the TCP-FIT sessions s is a strictly decreasing function of x_l when $R_{l,s} = 1$. Letting $T_s = \epsilon$, $x_{\max} = f_s^{-1}(T_s)$. $x_l > x_{\max} \Rightarrow x_s(p_l) < x_s = \epsilon$. Therefore, the assumption A2 holds.

Combining (1) and (2), equilibrium in (29) exists according to Theorem 1.□

5.3. Network utilization

Theorem 3. *When a network that contains TCP-FIT flows reaches its equilibrium, we have $L' = \{l \in L | \sum_s R_{l,s} T_s = B_l\} \neq \varnothing$ for the network.*

Proof 2. Assuming that the network reach an equilibrium point where $L' = \varnothing$, then $p_s = 0$ and $q_s = 0$ for all sessions in S according to (27). From the throughput function of TCP-FIT in (30), we have $x_s = \infty$. This contradicts with $\mathbf{RT} \le \mathbf{B}$. Therefore, equilibrium reached by network models including TCP-FIT flows satisfy $L' = \{l \in L | \sum_s R_{l,s} T_s = B_l\} \neq \varnothing$. □

The set $L' = \{l \in L | \sum_s R_{l,s} T_s = B_l\}$ is referred to as the *active constraint set* of a network, which contains all bottleneck links. Theorem 3 implies that TCP-FIT can ensure networks work in a state where all of its bottleneck links are saturated.

We list the throughput functions of several widely used TCP algorithms in Table 4 using the results of Padhye et al. (1998), Ha et al. (2008), Tan et al. (2006), Zhou et al. (2006), Grieco and Mascolo (2005), Samios and Vernon (2003), and Wei et al. (2006). The conclusions of Theorem 3 are valid for TCP Vegas and FAST TCP but not for the other TCPs. Using the throughput model of TCP Reno as an example, because $p_s \ge 0$ and $q_s \ge 0$, there is an upper bound for the throughput of the TCP Reno session s with given P_s and D_s, i.e.,

$$T_s = \frac{1}{D_s + q_s}\sqrt{\frac{3}{2(P_s + p_s)}} \le \frac{1}{D_s}\sqrt{\frac{3}{2P_s}} = T_s^{\max}.$$

For a network that only contains TCP Reno sessions, if throughput of every link $l \in L$ satisfies

$$\sum_{s\in S} R_{l,s} T_s \le \sum_{s\in S} R_{l,s} T_s^{\max} = \sum_{s\in S} R_{l,s} \frac{1}{D_s}\sqrt{\frac{3}{2P}} < B_l, \tag{31}$$

its active constraint set will be $\varnothing$, indicating that network bandwidth is not fully utilized.

Remark 1. The inequality (31) depends only on the P, D and B of the links and is independent of the congestion state of network. If the random packet loss rate P is sufficiently high, such as wireless links, and/or the bandwidth B and the propagation delay D are sufficiently large, such as large BDP networks, the throughput of TCP Reno can be significantly lower than the available bandwidths of the network. Similar upper bounds for the throughput models of different TCP algorithms are listed in Table 4. As shown in the table, except for TCP-FIT and Vegas/FAST, all of the algorithms have a throughput upper bound that is a function of P, D and B, and as such, could only mitigate but not completely solve the bandwidth utility problem of TCP over wireless/large-BDP links. Whereas for TCP-FIT and Vegas/FAST, such theoretical limitation does not exist. Therefore, at least in theory, TCP-FIT can work well over both wireless and large BDP links.

Although Vegas/FAST also can handle wireless and large BDP links in theory, they have very serious fairness problems with the widely deployed Loss-based TCP variants (Tan et al., 2006). According to the analysis in Section 5.5, TCP-FIT can overcome this problem.

5.4. Stability

The stability analysis of TCP-FIT follows the proofs of stability theorem of FAST TCP in Wei et al. (2006). According to our analysis, a network with TCP-FIT flows is locally asymptotically stable.

We assume that a group of TCP-FIT flows $s \in S$ pass through a bottleneck link $\tilde{l}$ which has a bandwidth $\tilde{B}$ and follows model assumptions of (27). We define the queuing delay and congestion packet loss rate of $\tilde{l}$ respectively are $\tilde{q}$ and $\tilde{p}$. For the sake of dissection, we normalize the window size of s-th TCP-FIT flow at the t-th update period as

$$u_s(t) = \frac{w_s(t)}{D_s} = \frac{N_s(t)}{D_s}\sqrt{\frac{3}{2(P_s + \tilde{p})}}. \tag{32}$$

Then, from (8), the evolution of TCP-FIT becomes

Table 4
TCP models.

TCP variants	Throughput models	Upper Bounds	$\eta = T_i(D_i)/T_j(D_j)$
TCP-FIT	$T = \max\{\frac{1}{D+q}\sqrt{\frac{3}{2(P+p)}}, \frac{\alpha}{q}\sqrt{\frac{3}{2(P+p)}}\}$	×	$\eta = 1$, when $E[N] > 1$
Reno	$T = \frac{1}{D+q}\sqrt{\frac{3}{2(P+p)}}$ (Padhye et al., 1998)	$\frac{1}{D}\sqrt{\frac{3}{2P}}$	$\eta = \frac{D_j+q}{D_i+q}$
CUBIC	$T = 1.174\sqrt[4]{\frac{1}{(D+q)(P+p)^3}}$ (Ha et al., 2008)	$1.174\sqrt[4]{\frac{1}{DP^3}}$	$\eta = \sqrt[4]{\frac{D_j+q}{D_i+q}}$
Compound	$T = \frac{1}{D+q}\frac{\Lambda}{(P+p)^{\frac{1}{2-k}}}$, $\Lambda = \frac{(\frac{1-(1-\beta')^{2-k}}{2-k})^{\frac{1-k}{2-k}}}{\alpha'^{\frac{1}{2-k}}(1-(1-\beta')^{1-k})}$ where, α', β' and k are preset parameters (Tan et al., 2006)	$\frac{1}{D}\frac{\Lambda}{P^{\frac{1}{2-k}}}$	$\eta \le [\frac{D_j+q}{D_i+q}]^2$ (Tan et al., 2006)
Veno	$T = \frac{1}{D+q}\sqrt{\frac{1+\gamma'}{2(1-\gamma')(P+p)}}$, where $\frac{1}{2} \le \gamma' \le \frac{4}{5}$ (Zhou et al., 2006)	$\frac{1}{D}\sqrt{\frac{1+\gamma'}{2(1-\gamma')P}}$	$\eta = \frac{(D_j+q)\sqrt{(1+\gamma_i')(2(1-\gamma_j')(P+p))}}{(D_i+q)\sqrt{(1+\gamma_j')(2(1-\gamma_i')(P+p))}}$
Westwood	$T = \frac{1}{\sqrt{(D+q)((P+p)D+q)}}\sqrt{\frac{(1-(P+p))}{P+p}}$ (Grieco and Mascolo, 2005)	$\frac{1}{DP}\sqrt{1-P}$	$\eta = \sqrt{\frac{(D_j+q)((P+p)D_j)+q}{(D_i+q)((P+p)D_i)+q}}$
Vegas/FAST	$T = \gamma/q$, where γ is preset parameter (Samios and Vernon, 2003; Wei et al., 2006)	×	$\eta = 1$

$$u_s(t+1) = \left(1 - \frac{\tilde{q}(t)}{D_s + \tilde{q}(t)}\right) u_s(t) + \hat{\alpha}_s, \tag{33}$$

where $\hat{\alpha}_s = \frac{\alpha_s}{D_s}\sqrt{\frac{3}{2(P_s + \bar{p})}}$. Let $\hat{\alpha} = \sum_{R_{\tilde{l},s}} \hat{\alpha}_s$.

Lemma 1. *$\forall\, \epsilon > 0$, we have*

$$\frac{\hat{\alpha}}{\tilde{B}} \min_s D_s - \epsilon < \tilde{q}(t) < \frac{\hat{\alpha}}{\tilde{B}} \max_s D_s + \epsilon,$$

for all sufficiently large t.

The proof of Lemma 1 is given in the appendixes. Without loss of generality, we assume that

$$\tilde{q}(t) > \frac{\hat{\alpha}}{2\tilde{B}} D_s \quad \text{for all } t \ge 0.$$

This statement implies that, for all $t \ge 0$, equality (a) in (27) holds. Therefore, for all $u \in \Re_+^N$ and $\tilde{q} \in \Re_+$, let

$$G(u, \tilde{q}) = \sum_{R\tilde{l},s} \frac{D_s u_s}{D_s + \tilde{q}} - \tilde{B} = 0. \tag{34}$$

Because $u = \{u_1, \ldots, u_s\}$ are strictly decreasing functions of $\tilde{q}$, given any $u \in \Re_+^N$, there is a unique $\tilde{q} \in \Re_+$ that satisfies (34). An important implication of Lemma 1 is that we can restrict the space of u to a subset in $\Re_+^N$:

$$\mathcal{U} = \left\{ u \in \Re_+^N \mid \text{the unique } \tilde{q}(u) \text{ defined implicitly by (34)} \right.$$
$$\left. \text{is greater than } \frac{\hat{\alpha}}{2\tilde{B}} \cdot D_s \right\}. \tag{35}$$

The key feature of $\mathcal{U}$ is that, for all $u \in \mathcal{U}$, $\tilde{q}(u)$ is lower bounded uniformly in u. Define $F: \mathcal{U} \to \mathcal{U}$

$$F_s(u) = \left(1 - \frac{\tilde{q}(u)}{D_s + \tilde{q}(u)}\right) u_s + \hat{\alpha}_s, \tag{36}$$

where $\tilde{q}(u)$ is implicitly defined by (34). Then, the evolution (33) of the normalized window size is $u(t+1) = F(u(t))$.

According to Theorem 3 of Wei et al. (2006), for all $\forall\, u \in \mathcal{U}$, the spectral radius of $\partial F/\partial u$ is strictly less than 1, which implies that, for the neighborhood of the unique fixed point u^* defined by $u^* = F(u^*)$, given any initial normalized window $u(0)$ in this neighborhood, $u(t+1) = F(u(t))$ converges to u^*. In other words, the TCP-FIT congestion control system consist of (27) and (33) is locally asymptotically stable.

5.5. TCP Reno friendliness

TCP Reno friendliness refers to fairness between TCP-FIT and the standard TCP Reno algorithm. We analyze TCP Reno friendliness of TCP-FIT in a scenario where M TCP flows share a bottleneck link. Let T_M be the average throughput of the M TCP flows using the TCP Reno algorithm. Then, we replace K of the M Reno flows with TCP-FIT flows, and let T_N be the average throughput of the rest $N = M - K$ TCP Reno flows. Thus, the throughput decrement of the TCP Reno flows caused by TCP-FIT is $T_\Delta = T_M - T_N$. Obviously, the closer T_Δ is to 0, the more Reno friendly TCP-FIT is.

We also assume that the bottleneck link by using (27) to control its buffer queues. Based on this assumption, we discuss the Reno friendliness of TCP-FIT in two conditions:

1. The M TCP Reno flows let the bottleneck run in the state (a) of (27), i.e, $M \times T_M = B$, and $q_M > 0$;
2. The M TCP Reno flows let the bottleneck run in the state (b) of (27), i.e, $M \times T_M \le B$, and q_M=0.

where q_M is the queuing delay of the bottleneck link when all of the M flows use TCP Reno.

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

Theorem 4. *If α setting of TCP-FIT satisfies*

$$\alpha \le \frac{q_M}{D + q_M}, \tag{37}$$

$T_\Delta = 0$, i.e., TCP-FIT is TCP friendly.

Proof 3. Because $M \times T_M = B$, and $M = N + K$, we have

$$T_M = \frac{1}{D + q_M}\sqrt{\frac{3}{2(P + p_M)}} = \frac{1}{N}\left(B - \frac{K}{D + q_M}\sqrt{\frac{3}{2(P + p_M)}}\right), \tag{38}$$

where p_M is the congestion packet loss rate of the bottleneck link when all of the flows use the Reno algorithm. Moreover, according to Theorem 2, a bottleneck link passed by TCP-FIT flows is fully saturated, so we have

$$T_N = \frac{1}{D + q_N}\sqrt{\frac{3}{2(P + p_N)}}$$
$$= \frac{1}{N}\left(B - K \cdot \max\left\{\frac{1}{D + q_N}, \frac{\alpha}{q_N}\right\}\sqrt{\frac{3}{2(P + p_N)}}\right), \tag{39}$$

where q_N, p_N are the queuing delay and congestion packet loss rare when we replace K of the TCP Reno flows with TCP-FIT flows.

We first suppose $T_M > T_N$, then from (38) and (39), we have

$$\frac{1}{D + q_M}\sqrt{\frac{3}{2(D + p_M)}} < \max\left\{\frac{1}{D + q_N}, \frac{\alpha}{q_N}\right\}\sqrt{\frac{3}{2(D + p_N)}} \Rightarrow \frac{1}{D + q_M}\sqrt{\frac{3}{2(D + p_M)}}$$
$$< \frac{\alpha}{q_N}\sqrt{\frac{3}{2(D + p_N)}}.$$

Because $\alpha \le \frac{q_M}{D + q_M}$,

$$\frac{1}{D + q_M}\sqrt{\frac{3}{2(D + p_M)}} < \frac{q_M}{q_N(D + q_M)}\sqrt{\frac{3}{2(D + p_N)}},$$

which implies that $q_M > q_N$. Because $p = v(q)$ is a non-decreasing function, $p_M = v(q_M) \ge p_N = v(q_N)$. Plugging $q_M > q_N$ and $p_M \ge p_N$ into the first row of (38) and (39), we obtain

$$T_M = \frac{1}{D + q_M}\sqrt{\frac{3}{2(D + p_M)}} < T_N = \frac{1}{D + q_N}\sqrt{\frac{3}{2(D + p_N)}},$$

which contradicts $T_M > T_N$.

Next, we suppose $T_M < T_N$. From (38) and (39), we have

$$\frac{1}{D + q_M}\sqrt{\frac{3}{2(D + p_M)}} > \max\left\{\frac{1}{D + q_N}, \frac{\alpha}{q_N}\right\}\sqrt{\frac{3}{2(D + p_N)}},$$

which implies that $q_M < q_N$ and $p_M \le p_N$. Plugging $q_M < q_N$ and $p_M \le p_N$ into the first row of (38) and (39) again, we obtain $T_M > T_N$, which contradicts $T_M < T_N$.

Therefore, $T_M = T_N \Rightarrow T_\Delta = 0$. TCP-FIT is TCP Reno friendly when $\alpha \le \frac{q_M}{D + q_M}$. □

From Theorem 4 we know that the premise of TCP-FIT maintaining TCP Reno friendliness is guaranteeing $\alpha \le \frac{q_M}{D + q_M}$. In the practical implementation, we approximately set α as

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

where $RTT_{\max}$ and $RTT_{\min}$ are the smoothed maximal and the minimal observed RTT of TCP-FIT sessions. The experimental results show that this approximate implementation performs well under a wide range of network conditions.

For the condition (2), where the TCP Reno flows cannot fully use the network bandwidth, p_M=0 and q_M=0, the throughput reduction of the TCP Reno flows when they share the same network with the TCP-FIT flows is

$$T_\Delta = \frac{1}{D}\sqrt{\frac{3}{2P}} - \frac{1}{D+q_N}\sqrt{\frac{3}{2(P+p_N)}}.$$

The throughput loss T_Δ is caused by the queuing delay and congestion loss incurred by TCP-FIT flows, and is a (usually) reasonable cost for more fully utilization of the aggregated bandwidth via the TCP-FIT algorithm.

5.6. RTT fairness

According to the definition of RTT fairness in Floyd and Allman (2008), "when the throughputs for TCP flows with similar packet loss rates but different RTTs are the same, the TCP algorithm is referred to as RTT fair". For TCP-FIT, we have the flowing theorem about its RTT fairness.

Theorem 5. *If $E[N] > 1$, TCP-FIT flows with the same α have the same throughput if they have the same bottleneck links and end-to-end random packet loss rate P.*

Proof 4. According to the throughput model of TCP-FIT in (26), when $E[N] > 1$, $T_{fit} = \frac{\alpha}{q}\sqrt{\frac{3}{2PLR}}$. Thus, for any two TCP-FIT flows, i and j traversing the same saturated bottleneck links and with identical values of α and end-to-end packet loss rate P, the ratio between their throughputs is

$$\eta_{fit} = \frac{T_i}{T_j} = \frac{q_j}{q_i}\sqrt{\frac{P+p_j}{P+p_i}}.$$

Because i and j pass the same bottleneck links, $p_i=p_j$ and $q_i=q_j$. Therefore, $T_i=T_j$. □

From Theorem 5 we know TCP-FIT is an RTT fair algorithm when $E[N] > 1$. It is worth noting that Theorem 5 only requires the TCP-FIT flows pass the same *bottleneck* links, rather than traversing exactly identical paths, including bottleneck and non bottleneck links. This implies that these flows could have different end-to-end RTTs.

On the other hand, TCP Reno and many other algorithms are not RTT fair. For two TCP Reno flows i and j, the throughput ratio is

$$\eta_{reno} = \frac{D_j+q}{D_i+q}.$$

If $D_i \neq D_j$, then the TCP flow with a longer propagation RTT will have a lower throughput and will therefore be at a disadvantage. Among the algorithms listed in Table 4, only TCP-vegas/FAST could achieve theoretical RTT fairness.

When $E[N] = 1$, TCP-FIT is same as TCP Reno, and therefore not theoretically RTT fair. A TCP algorithm cannot be both RTT fair and TCP Reno friendly due to the TCP Reno algorithm itself is not RTT fair. TCP-FIT achieves a balance between RTT fairness and TCP Reno friendliness.

5.7. Responsiveness

We analyze the responsiveness of TCP-FIT from two aspects: bandwidth probing and congestion back-off. The probing speed of TCP-FIT can be described by the derivative of the window growth function with regard to time t, i.e.,

$$I'_{fit}(t) = \left\lfloor \frac{t}{R} \right\rfloor \frac{1}{RTT}, \tag{40}$$

where R is the length of an N update period, which was set to $\max\{RTT, 500\text{ ms}\}$ in our implementation. Table 5 lists the $I'(t)$ of TCP-FIT, TCP Reno, TCP CUBIC and Slow Start. As shown in (40), the window growth function of TCP-FIT is a quadratic polynomial function, which grows slowly at the beginning and then becomes increasingly fast as the window size increases. According to the analysis in Floyd (2003) and Cai et al. (2007), quadratic polynomial functions is a scalable window size growth manner, because its slow growth beginning is very suitable to probe bandwidth of low speed links and the subsequent increasedly fast window growth is very suitable to saturate large BDP of high speed networks. TCP Reno uses a linear function with a 1/RTT gradient in its window growth function, which is too slow to saturate large BDP links. The Slow Start uses an exponential function, which probes the network bandwidth very quickly but is too aggressive for Congestion Avoidance.

For network scenarios that require ultra response speed, TCP-FIT can modify its N update function (8) as following from,

$$N_{i+1} \leftarrow \max\left\{1, N_i + \beta - \frac{\beta\cdot(RTT_i - RTT_{\min})}{\alpha\cdot RTT_i}N_i\right\}. \tag{41}$$

Magnify β of (41) can raise the increasing step length of N. The recommended setting of β is $5^{\lg BDP}$. Table 6 lists the response speed TCP-FIT (with and without increasing β), TCP Reno, TCP CUBIC and Slow Start over 1–100K packets BDP links. The responsiveness of TCP-FIT using the recommended β setting is satisfactory for most applications.

According to (3) and (8), both the window size back-off of TCP-FIT is in an exponential manner. The exponential back-off is widely used almost all TCP algorithms and has been found to be satisfactory for most applications.

Table 5
Bandwidth probing method of different TCPs.

	Reno	TCP-FIT	CUBIC	Slow Start
I′	$\frac{1}{RTT}$	$\left\lfloor \frac{t}{R} \right\rfloor \frac{1}{RTT}$	$3C(t-K)^2$	$2^{\frac{t}{RTT}}\ln 2$
Feature	Linear	Polynomial		Exp.

Table 6
The response speed of different algorithms.

BDP	1K packets	10K packets	100K packets
Equivalents	**100 mb/100 ms**	**1 Gb/100 ms**	**10 Gb/100 ms**
Default β	1	5	25
TCP-FIT with default β	9.8 s	14.3 s	20.3 s
TCP-FIT β=1	9.8 s	31.4 s	99.8 s
Reno	100 s	1000 s	10,000 s
CUBIC	8.9 s	19.3 s	51.5 s
Slow start	1 s	1.4 s	1.7 s

6. Experimental results

In this section, we tested the performance of TCP-FIT over large BDP, random packet loss, random RTT fluctuation, TCP Reno friendliness, RTT fairness and responsiveness scenarios. Because there are so many TCP algorithms and diverse algorithms have different fortes for different scenarios, benchmark selection becomes a very difficult task in our experiments. To find a reasonable benchmark, we tested all the algorithms listed in Table 1 and selected the algorithm with the best performance in the target scenario as the benchmark algorithm of the corresponding experiments. The specific benchmark algorithm selection of each experiment is summarized in Table 7. Moreover, because Section 5 shows that TCP Vegas and FAST have the same theoretical

Table 7
The setup of experiments.

Scenarios	**Benchmark**	**Testbed**
Large BDP	CUBIC, **FAST**	Dummynet
Packet loss	Westwood, **Vegas**	Linktropy
RTT fluctuation	Westwood, **Vegas**	Linktropy
TCP friendliness	**Reno**, CUBIC, Vegas	Linktropy
RTT fairness	**Hybla**, Vegas	Linktropy
Responsiveness	CUBIC, **FAST**	Dummynet

performance as TCP-FIT with regard to network utilization and RTT fairness, we also compared the performance of TCP-FIT with that of Vegas/FAST in all experiments. Specifically, based on the design goal of the two algorithms, FAST TCP was used in the experiments involving a high-speed network, i.e., large BDP networks and responsiveness-whereas Vegas was used in the other tests. In addition, the performance of TCP Reno is also demonstrated as a reference. In Table 1, the algorithm with the best performance under the corresponding experiment setup is reported in boldface.

We embedded TCP-FIT into the Linux kernel v2.6.31 and implemented FAST TCP based on the ns-2 code of FAST TCP (Cui and Andrew, 2010). Other TCP variants are available in Linux kernels. Except where noted, β was set to 1 to investigate the performance of TCP-FIT with the most conservative setup. Other parameters followed the recommendation in Section 5. We used the dummynet (Rizzo, 2010) software network emulator as the testbed in the $\geq$1 Gbps link experiments, which supports $\geq$1 Gbps bandwidth but cannot generate RTT fluctuations. We used the Linktropy mini hardware emulator (Linktropy Mini2 WAN Emulator, 2010) in the $\leq$100 Mbps experiments, which can support RTT fluctuation emulations but have only a 100 Mbps bandwidth.

TCP-FIT uses $RTT_{\min}$ and $RTT_{\max}$ to determine N, which are susceptible to delay variations caused by background traffic; therefore, it is worth checking the performance of TCP-FIT with background traffic. Therefore, we generated five continuous UDP pulse flows on the bidirectional links in each experiment to check the performance of TCP-FIT with background traffic. The interval between contiguous pulses followed an exponential distribution, i.e., $f(t) = e^{\lambda t}$, where t is in seconds, and λ is set to 10. The duration time of a UDP pulse was 10 ms, and the intensity of a pulse was 2% of the bottleneck bandwidth. To remove the influence of irrelevant factors, the SACK and Timestamp options were disabled in our experiments, and the other parameters used the default setting of Linux v2.6.31.

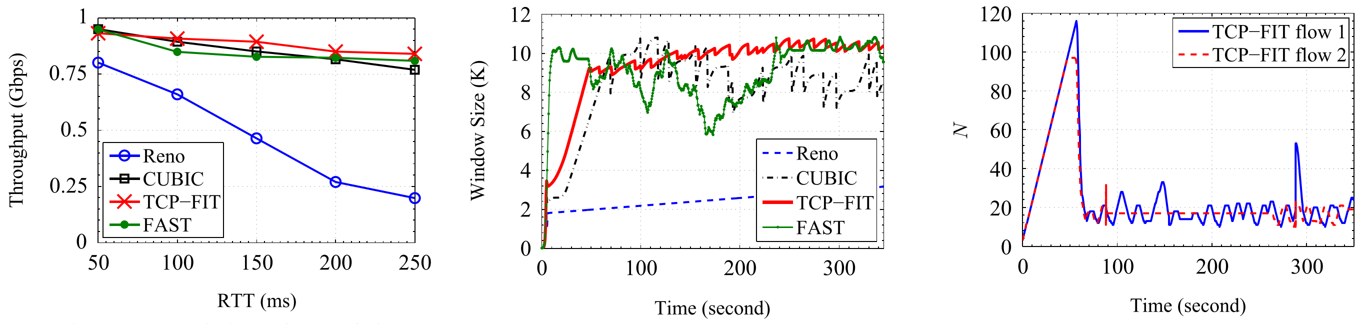
6.1. Large BDP networks

We first compared the performance for large BDP links. In the experiments, two TCP flows shared a 1 Gbps link with a 50–250 ms propagation RTT. The aggregated throughput of the two flows are plotted in Fig. 3(a). The throughput of TCP-FIT in the large BDP scenario was near that of TCP CUBIC and FAST TCP and much higher than TCP Reno. As shown in the figure, the throughput of TCP-FIT was obviously higher than TCP Reno and very close to TCP CUBIC and FAST TCP. TCP Reno is not designed for high-speed networks; therefore, its performance was inferior to that of TCP-FIT. TCP CUBIC and FAST TCP are optimized for high-speed networks; therefore, the throughput performance of these algorithms was very near network capacity. TCP-FIT did not achieve an observable performance gain compared with these two algorithms. Fig. 3(b) plots the window size variation of different TCP variants when the propagation RTT was 250 ms, which gives the stability comparison of TCP-FIT with other algorithms. As shown in the figure, the window size of TCP-FIT was more stable than TCP CUBIC and FAST TCP. TCP CUBIC is a loss-based TCP algorithm, and FAST TCP is a delay-based algorithm. These two algorithms use either packet losses or queuing delay as an input signal to control the sliding window size, but in contrast, TCP-FIT uses both packet losses and queuing delay as the symptom for congestion. This feature ensures that TCP-FIT performs better than TCP CUBIC and FAST TCP in terms of stability. As shown in Fig. 3(b), the BDP of the experimental setup was approximately 10,000 packets. TCP Reno achieved only approximately 3000 packets by the end of the experiment. The growth speed of TCP Reno was too slow to fully saturate the network BDP. TCP-FIT took approximately 40 s to achieve the network BDP, and this speed was significantly faster than TCP Reno and very close to TCP CUBIC. After the starting step, TCP-FIT steadied the window at a large size and guaranteed that the network bandwidth was fully saturated. As shown in the figure, the stability of TCP-FIT was better than that of other TCPs.

The N variations of the two TCP-FIT flows are plotted in Fig. 3(c). TCP-FIT continuously increased N at the beginning stage. The increase in N at the beginning stage allowed TCP-FIT to saturate the network bandwidth as rapidly as possible. At the end of the beginning stage, the value of N even reached more than 110. Nevertheless, the increase in N was not out of control because TCP-FIT has an exponential N back-off mechanism. When the window size reached the network BDP capacity and the network congestion was detected by increasing the queuing delay, N rapidly converged to a reasonable value and then became stable.

6.2. Random packet losses

To evaluate the performance of TCP-FIT with wireless packet

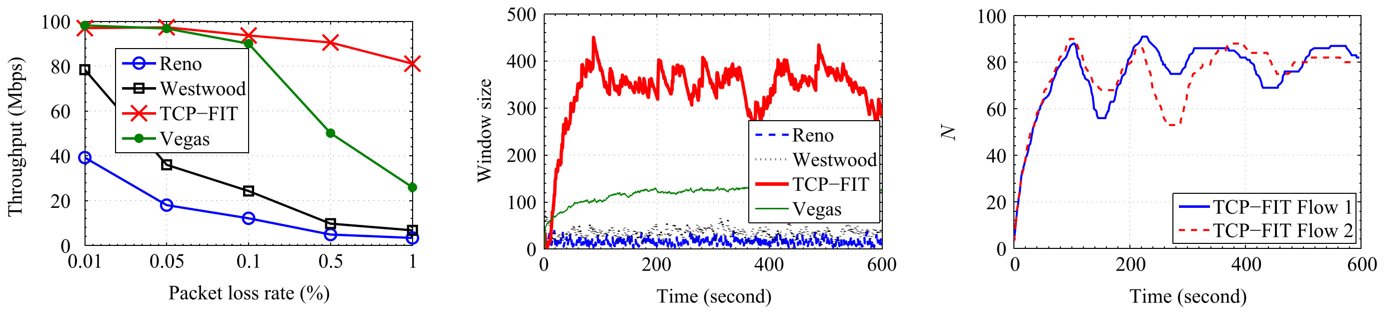


(a) The aggregated throughput of the two flows with different TCP algorithms.

(b) The window size variation of different TCP variants when the propagation RTT was 250ms.

(c) The N variations of the two TCP-FIT flows.

Fig. 3. Performance of the different TCP variants over large BDP links. In the experiments, two TCP flows shared a 1 Gbps link with 50–250 ms propagation RTT.



(a) The aggregated throughput of the two flows using different algorithms with different loss rate.

(b) The window size variation of different algorithms when the packet loss rate is 1%.

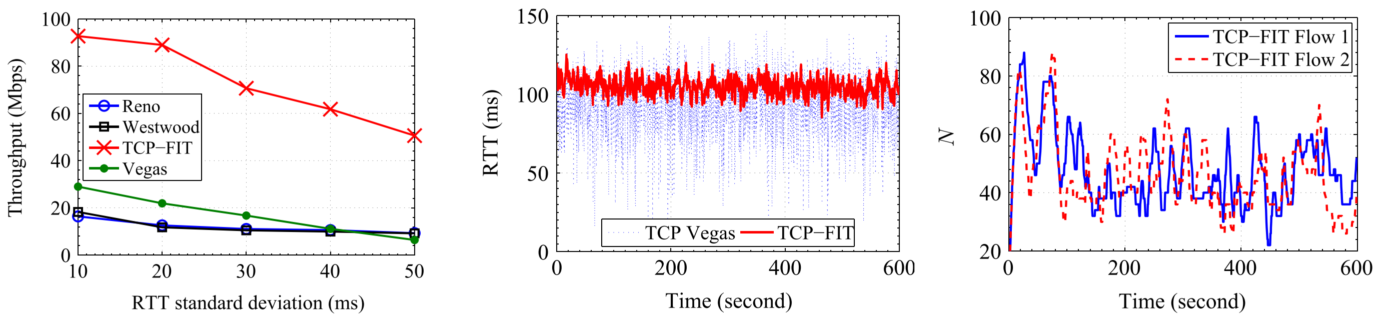
(c) The N variation of the two TCP-FIT flows with 1% packet loss rate.

Fig. 4. Performance of the different TCP variants over wireless lossy links. In the experiments, two TCP flows sharing a link with a 100 Mbps bandwidth, 100 ms RTT and 0.01–1% packet loss rate.

losses, we set an experimental scenario in which two TCP flows shared a link with a 100 Mbps bandwidth, 100 ms RTT and 0.01–1% packet loss rate.We set the bandwidth as 100 Mbps because it is the peak speed requirement for 4G service. According to Fu and Liew (2003) and Mascolo et al. (2001), the performance of conventional wireless TCP variants degenerates sharply when the packet loss rate is in a range 0.01–1%; therefore, we set the packet loss rate in our experiments to 0.01% to 1%. In addition, 100 ms is a typical RTT of high-speed wireless networks, such as HSDPA; therefore, we set the propagation delay in our experiments to 100 ms. Fig. 4(a) plots the aggregated throughputs of the two flows using different algorithms with different loss rates. As shown in the figure, all algorithms suffered throughput degradation with the appearance of packet losses; even so, the throughput of TCP-FIT was notably higher than that of the others. The performance of TCP Vegas severely degenerated when the packet loss rate was higher than 0.1%. This is because in high packet loss environments, TCP congestion avoidance is frequently interrupted by TimeOut and fast recovery, and TCP Vegas increases its window by one packet during an RTT that is too slow to saturate the network capacity with frequent interrupts. Fig. 4(b) plots the window size variation of different algorithms when the packet loss rate is 1%. We observe that the window size of TCP-FIT steeply increased after a window reduction, whereas the window of TCP Vegas increased very slightly. The N variation of the two TCP-FIT flows with a 1% packet loss rate is shown in Fig. 4(c). N converged to a stable value after several long-term oscillations. The convergence slowed by random packet losses.

6.3. RTT fluctuations

Random RTT fluctuations are another characteristic of wireless networks. The experimental setup that was used to evaluate TCP-FIT with RTT fluctuations included two flows sharing a link with a 100 Mbps bandwidth, 0.1% packet loss rate, and 100 ms base RTT. We generated random RTT fluctuations that follow a normal distribution with a 0 ms mean and 10–50 ms standard deviations. The random fluctuations were added onto the base RTT of the link. (According to the results in Fig. 4 of Section 6.2, when there were no RTT fluctuations, both TCP-FIT and TCP Vegas could fully use the network bandwidth under this setup.) The aggregated throughput of the two flows with different algorithms with RTT fluctuations is plotted in Fig. 5(a). As shown in the figure, the throughput of TCP Vegas is severely degraded by RTT fluctuations. TCP-FIT also had some performance degradation, but it was much less severe. The reason why TCP-FIT performs more robustly is that TCP-FIT does not directly use queuing delay to control its congestion window. Queuing delay is used only to adjust the parameter N, and TCP-FIT indirectly uses N to set the increase and decrease parameters of the congestion control window. The adjustment periods of N are significantly longer than the window size; therefore, TCP-FIT can obtain sufficient queuing delay samples during an adjustment period such that an average filter can be used to denoise RTT fluctuations. In contrast, TCP Vegas/FAST uses RTT samples during a round trip time to approximate the current RTT, which is highly sensitive to fluctuation noise. Fig. 5(b) shows the estimated RTT of TCP-FIT and TCP Vegas/FAST with 50 ms standard deviation fluctuation. We can see that the impact of fluctuation noise in TCP-FIT is obviously lower than that of TCP Vegas/FAST.

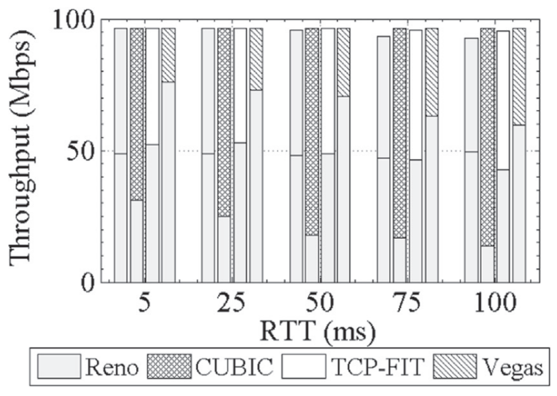


(a) The aggregated throughput of the two flows with different algorithms with RTT fluctuations.

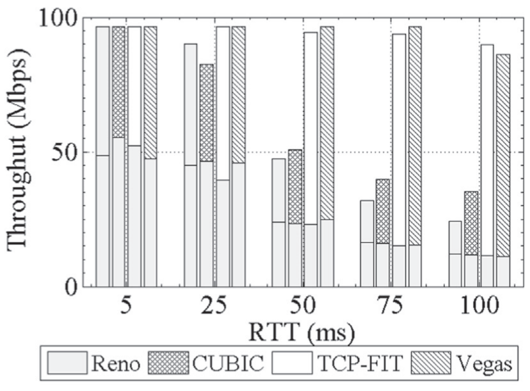
(b) The estimated queuing delay of TCP-FIT and TCP Vegas/FAST with a 50 ms standard deviation fluctuation.

(c) The N variations of the two TCP-FIT flows with a 50 ms RTT standard deviation.

Fig. 5. Performance of the different TCP variants with random RTT fluctuations. In the experiments, two flows shared a link with a 100 Mbps bandwidth, 0.1% packet loss rate, and 100 ms base RTT. We generated random RTT fluctuations that follow a normal distribution with a 0 ms mean and 10–50 ms standard deviations.



(a) The fairness results with a 0% packet loss rate.



(b) The fairness results with a 0.1% packet loss rate.

Fig. 6. RTT fairness performance of TCP-FIT were compared with TCP Reno, TCP CUBIC and TCP Vegas. In the experiments, a TCP flow that used one of the tested algorithms shared a 100 Mbps link with a TCP Reno flow. The RTT of the link varied from 5 ms to 100 ms.

Although the throughput of TCP-FIT is very robust with RTT fluctuations, RTT fluctuations were detrimental to the convergence of the parameter *N*. Fig. 5(c) shows the *N* variations of the two TCP-FIT flows with a 50 ms RTT standard deviation. The *N* of TCP-FIT flows cannot converge to a steady state in the figure.

6.4. TCP Reno friendliness

Fig. 6 shows the comparison results of RTT fairness performance between TCP-FIT and TCP Reno, TCP CUBIC and TCP Vegas. In the experiments, a TCP flow that used one of the tested algorithms shared a 100 Mbps link with a TCP Reno flow. The RTT of the link varied from 5 ms to 100 ms. The bars with different marks in Fig. 6 represent the bandwidth occupied by different algorithm flows. Fig. 6(a) shows the results without random packet losses. In the figures, the bars contain two parts; the lower parts of the bars represent the throughput of TCP Reno, and the upper parts of the bars represent the throughput of the target algorithms. As shown in the figure, the network bandwidth occupied by TCP-FIT is the same as that of TCP Reno, which is the theoretical "fair share" with the Reno flows. However, the network bandwidth occupied by TCP Vegas was less than that of TCP Reno, whereas TCP CUBIC occupied more bandwidth than Reno, indicating that these algorithms are not TCP Reno friendly. Fig. 6(b) shows the results with a 0.1% packet loss rate. In this setup, TCP-FIT could pick up the capacity unused by the TCP Reno flow. Moreover, the TCP Reno flows competing with TCP-FIT flows under the same RTT had a similar throughput, which means that the improved TCP-FIT throughput did not come at the expense of the CUBIC flows. The results in Fig. 6 indicate that TCP-FIT can fully use network bandwidth and maintain friendliness with TCP Reno flows over both lossy and loss-free networks.

In recent years, TCP CUBIC and Compound TCP have been widely deployed with Linux and Windows workstations; therefore, the fairness performance of TCP-FIT with TCP CUBIC and Compound TCP must be investigated. In theory, TCP-FIT can achieve fairness with Compound TCP because both TCP-FIT and Compound TCP are TCP Reno fair. TCP-FIT cannot maintain fairness with TCP CUBIC because, as reported in Wang et al. (2013b), the algorithm of TCP CUBIC is much more aggressive than that of TCP-FIT. We tested the fairness of TCP-FIT with TCP CUBIC and Compound TCP via an experiment. In the experiment, a TCP-FIT flow shared a 100 Mbps bottleneck link with a TCP CUBIC or Compound TCP. The propagation delay of the bottleneck varies from 5 ms to 100 ms. The experimental results are shown in Fig. 7. The bars in

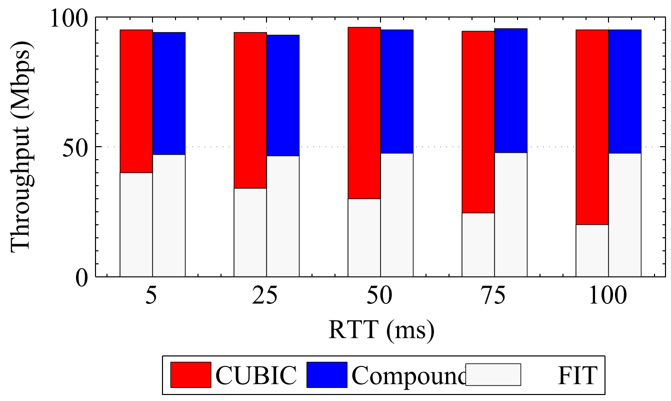


Fig. 7. Fairness of TCP-FIT compared with TCP CUBIC and Compound TCP. In the experiments, a TCP flow that used one of the tested algorithms shared a 100 Mbps link with a TCP Reno flow.

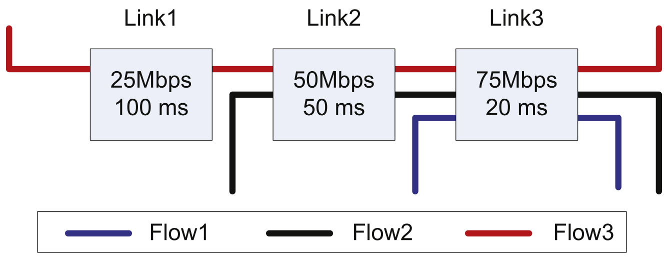


Fig. 8. The topology of the experiments, which contains three TCP flows marked by blue, black, and red lines and three network links marked by boxes. The bandwidth and propagation delay of Link1–Link3 are written on the link boxes. In this scenario, TCP Flow1 passed Link3, TCP Flow2 passed Link2 and Link3, and TCP Flow3 passed all of Link1–Link3. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

Fig. 7 contain two parts; the lower parts of the bars represent the throughput of TCP-FIT, and the upper parts of the bars represent the throughput of target TCP CUBIC or Compound TCP. As shown in the figure, the throughputs of the TCP-FIT flows are very similar to the throughput of Compound TCP when the TCP-FIT flow and the Compound TCP flow shared the same bottleneck link. When a TCP-FIT flow shared a bottleneck with a TCP CUBIC flow, the throughput of TCP CUBIC was obviously higher than that of TCP-FIT, which agrees with the conclusion of Wang et al. (2013b). To overcome the unfair problem of TCP-FIT with TCP CUBIC, we proposed a TCP CUBIC-based TCP-FIT algorithm in Wang et al. (2013b), named CUBIC-FIT.

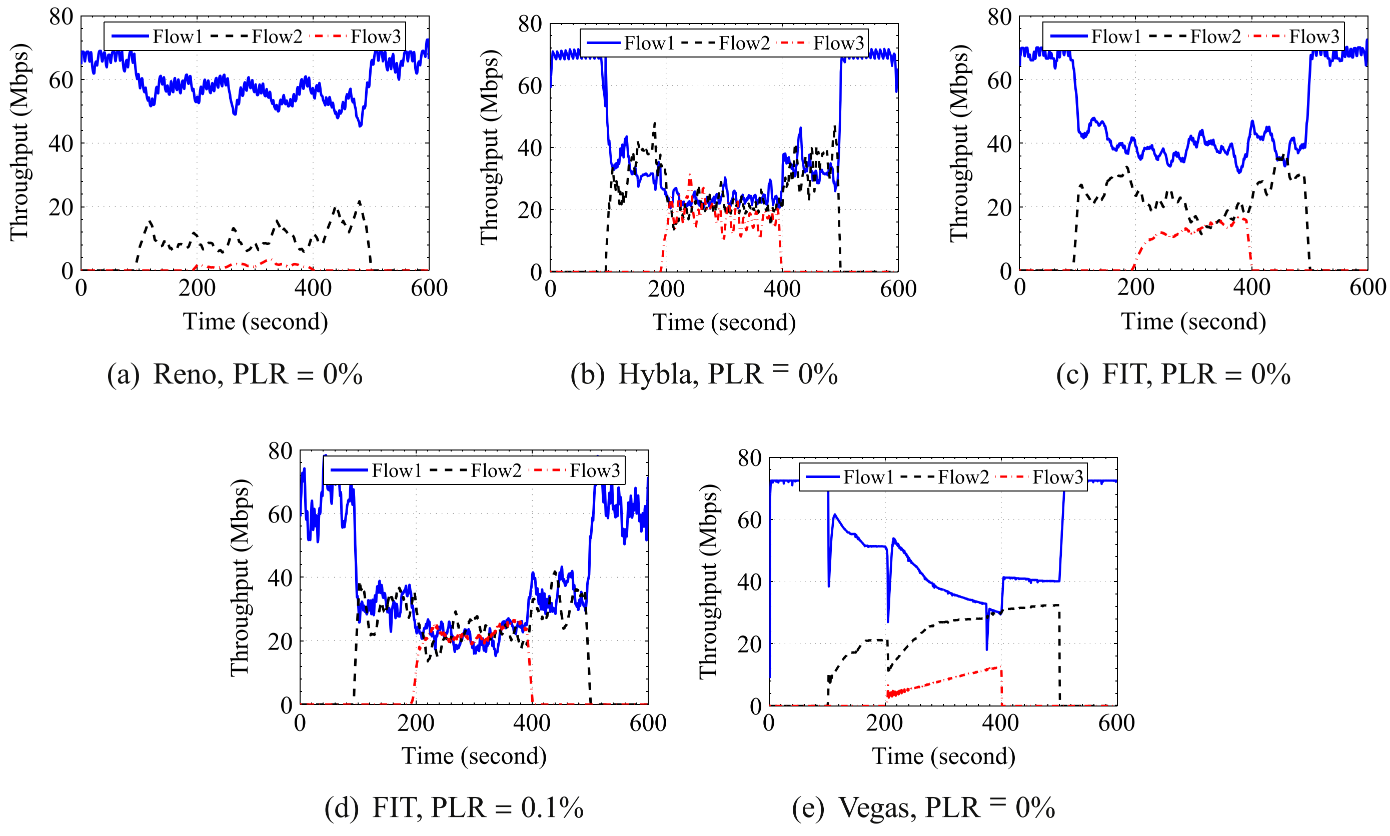


Fig. 9. Throughput variation of different algorithms in the multiple bottleneck scenario given in Fig. 8.

6.5. *RTT fairness*

RTT fairness of TCP-FIT was tested over a multiple bottlenecks scenario. Fig. 8 gives a network topology that contains three TCP flows marked by blue, black, and red lines and three network links marked by boxes. The bandwidth and propagation delay of Link1-Link3 are written on the link boxes. In this scenario, TCP Flow1 passed Link3, TCP Flow2 passed Link2 and Link3, and TCP Flow3 passed all of Link1–Link3. Under this network setup, the end-to-end propagation delays of Flow1–Flow3 are 150 ms, 125 ms and 75 ms, respectively. If the TCP algorithm adopted by this network is an RTT fair algorithm, Flow1–Flow3 should fairly share the bandwidth of bottleneck Link3. Otherwise, the throughput of TCP Flow3 would be much higher than that of the other flows because it suffered the shortest network propagation delay. Fig. 9(a) plots the throughput variation of TCP Reno when random packet loss rate for all links was 0%. There is obviously RTT unfairness among the three TCP Reno flows. The flow1 occupied most of network bandwidth because its end-to-end RTT was only 20 ms. The flow2 and flow3, for which the end-to-end RTTs were 70 ms and 170 ms respectively, suffered severe bandwidth starvation. The throughput of flow3, which had the longest RTT, was close to zero. In comparison, Fig. 9(b) plots the results for TCP Hybla, demonstrating great fairness among the three flows. This is consistent with the theory analysis for the TCP Hybla algorithm in Caini and Firrincieli (2004). Fig. 9(c) plots the results of TCP-FIT with a 0% random packet loss rate. The fairness shown in Fig. 9(c) was remarkably improved compared with TCP Reno, but was still worse than TCP Hybla. The throughput of the flow1 for TCP-FIT is slightly higher than the other two flows. The flow2 and flow3 shared the network in approximate fairness. Fig. 9(d) plots the throughput variation of TCP-FIT when link3, the last hop, had a 0.1% packet loss rate. TCP-FIT fairly shared the network bandwidth under this condition. As analyzed in Section 5, TCP-FIT is RTT fair only when $E[N] > 1$. TCP-FIT over the lossy links allows $E[N] > 1$; thus, graceful RTT fairness was performed in Fig. 9(d). Fig. 9(c) and (d) demonstrates that, although TCP-FIT cannot completely solve the RTT fairness problem, the RTT fairness was improved significantly over the TCP Reno algorithm. As a TCP algorithm cannot be both TCP Reno friendly and RTT fair, the TCP-FIT algorithm presents a good compromise between them.

The RTT fairness of TCP Vegas is plotted in Fig. 9(e). The fairness among TCP Vegas flows was not as good as the theoretical analysis. This phenomenon is caused by the TCP Vegas AIAD (additive increase adaptive decrease) window adjustment mechanism. Using AIAD, the early-comer flows cannot back off in a timely manner for later-comer flows, causing the TCP Vegas algorithm to underachieve compared with its theoretical limits.

6.6. *Responsiveness*

We investigated the responsiveness of TCP-FIT over two scenarios. In the first scenario, an existing TCP-FIT flow and a new TCP-FIT flow shared a 1 Gbps link. The RTT of the link was set from 20 to 200 ms. Table 8 lists the number of times for the two flows to converge to the fair-share state for different RTTs. In the second scenario, there were 100 TCP-FIT flows on a 1 Gbps link, 99 of which were completed at the same time. The time required for the

Table 8
Response time of the first scenario.

RTT	20 ms	50 ms	100 ms	200 ms
TCP Reno	9.2 s	1.0 m	3.8 m	24.3 m
TCP CUBIC	3.1 s	5.7 s	9.0 s	13.9 s
FAST TCP	0.2 s	0.4 s	1.2 s	3.9 s
TCP-FIT $\beta=1$	1.5 s	4.9 s	10.1 s	19.0 s
TCP-FIT $\beta=5$	0.3 s	0.9 s	2.1 s	4.8 s

Table 9
Response time of the second scenario.

RTT	20 ms	50 ms	100 ms	200 ms
TCP Reno	21.0 s	4.1 m	19.7 m	68.5 m
TCP CUBIC	12.0 s	22.0 s	35.9 s	57.0 s
FAST TCP	0.1 s	1.3 s	6.4 s	11.5 s
TCP-FIT $\beta=1$	8.3 s	21.4 s	29.2 s	1.1 m
TCP-FIT $\beta=5$	1.6 s	3.9 s	7.1 s	14.1 s

Table 10
Throughput improvements of TCP-FIT on Planetlab.

	TCP-FIT	Reno	CUBIC	Vegas	Compound
Throughput	20.2 Mbps	14 Mbps	16.2 Mbps	9.8 Mbps	14.7 Mbps
Speedup	×	42%	23%	104%	37%

single remaining TCP-FIT session to fully utilize the newly available bandwidth is listed in Table 9. Other TCP algorithms were also tested in the two scenarios for comparison. As shown in the tables, the response times of TCP-FIT are acceptable for most long-term TCP applications. For applications requesting short-lived sessions that involve only several RTTs, TCP-FIT is compatible with the technologies that can improve the performance of short-term TCP flows, such as Wang et al. (2013) and Flach et al. (2013). We can implement these solutions on top of TCP-FIT to improve its responsiveness for short-lived sessions.

7. Experiments over real networks

To investigate the performance of TCP-FIT over wide area networks, we used PlanetLab (Chun et al., 2003) as a testbed to compare TCP-FIT with TCP Reno, CUBIC, and TCP Vegas. In our experiments, 245 PlanetLab nodes located in 192 cities of 43 countries were used. These nodes covered 233 ISPs, representative of the current conditions of the Internet. In our experiments, the PlanetLab nodes simultaneously and repeatedly downloaded seven video clips from an HTTP server located in San Diego, California, USA for 6 h. The average size of these video clips was 300 MB. Table 10 summarizes the average throughputs of different TCP variants. As shown in the table, the speedups of TCP-FIT to TCP Reno, CUBIC and Vegas are 42%, 23% and 104%, respectively. The results indicate that TCP-FIT is effective for improving the TCP performance on the actual Internet.

8. Conclusions

In this study, we describe a novel TCP congestion control algorithm for application over heterogeneous networks that contain both large BDP and wireless links. Theoretical and emulation results as well as performance measured using live real-world networks show significant performance improvements in the throughput, fairness responsiveness and robustness over state-of-the-art TCP variants.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (Grant nos. 61572059, 61202426), the State Key Program of National Natural Science of China (Grant no. 71531001), the Basic Science Research Program of Shenzhen (Grant no. JCYJ20140828113329138) and the Open Research Fund Program of State Key Laboratory of Software Development Environment (Grant no. SKLSDE-2015ZX-19).

Appendix A. Proof of Theorem 1

We prove the following lemma that bounds the measure prices before proving Theorem 1.

Lemma 2. *Suppose that the A1 and A2 hold. Given a network (L, S, R), there is a scalar $x_{\max}$ that is an upper bound to any x_l.*

Proof 5. Choose $\epsilon = \min B_l/S$, and let $X_{\max}$ be the corresponding scalar in A2. Suppose that there exists an equilibrium price x and a link l, such that $x_l > x_{\max}$. The A2 implies that aggregated equilibrium throughput at link l satisfies

$$\sum_s R_{l,s}T_s(x) < S{\cdot}\epsilon = \min_l B_l$$

We therefore have a link with $x_l > 0$, but it is not fully utilized, i.e.,

$$\mathbf{X}(\mathbf{RT} - \mathbf{B}) \neq 0,$$

which contradicts with definition of equilibrium.□

Proof of Theorem 1. Let $x_{\max}$ be the scalar upper bound in Lemma 2. For $x_l \in [0, x_{\max}]$, define $F(x_l) = \mathbf{RT}(x_l) - \mathbf{B} = F_l(x_l, x_{-l})$, where $x_{-l} = (x_1, x_2, x_{l-1}, x_{l+1}, \ldots, x_L)^T$, and $H_l(x_l, x_{-l}) = -F_l^2(x_l, x_{-l})$.

For any fixed x_{-l}, $H_l(x_l, x_{-l})$ is a quasi-concave function on x_l. In fact, we only need to confirm that the upper contour set $A_l = \{x_l | H_l(x_l, x_{-l}) \geq \delta\}$ is convex for all $\delta \in \Re$:

If $\delta > 0$, $A_l = \phi$.

If $\delta \leq 0$, $A_l = \{x_l | -\sqrt{\delta} \leq F_l(x_l, x_{-l}) \leq \sqrt{\delta}\}$.

Because $T_s(x_l)$ is a non-increasing function, $F_l(x_l, x_{-l})$ is a non-increasing function in x_l. The set A_l is convex; thus, $H_l(x_l, x_{-l})$ is a quasi-concave function in x_l.

Therefore, from the game theorem of Nash (Osborne and Rubinstein, 1994), there exists $x_l^* \in [0, x_{\max}]$ such that, for all l

$$x_l^* = \arg \max_{x_l \in [0, x_{\max}]} H_l(x_l, x_{-l}^*).$$

We then prove that $F_l(x_l^*) = 0$, or $F_l(x_l^*) < 0$ with $x_l^* = 0$.

If $F_l(0, x_{-l}^*) > 0$, because $F_l(x_l, x_{-l}^*)$ is non-increasing in $[0, x_{\max}]$ and because $F_l(x_{\max}, x_{-l}^*) < 0$, there exists $x_l^* \in [0, x_{\max}]$ that satisfies $F_l(x_l^*, x_{-l}^*) = 0$. This x_l^* maximizes $H_l(x_l, x_{-l}^*)$.

If $F_l(0, x_{-l}^*) \leq 0$, because $F_l(x_l, x_{-l}^*)$ is a non-increasing function, we obtain that $F_l(x_l, x_{-l}^*) \leq 0$ for any $x_l \in [0, x_{\max}]$, and $H_l(x_l, x_{-l}^*)$ is also a non-increasing function. We have

$$x_l^* = \arg \max_{x_l \in [0, x_{\max}]} H_l(x_l, x_{-l}^*) = 0$$

Thus, we have proved that, for $l = 1, \ldots, L$, there exists a x_l^* such that

$$x_l^* F_l(x_l^*, x_{-l}^*) = 0, \quad F_l(x_l^*, x_{-l}^*) \leq 0, x* \geq 0,$$

which is equilibrium in (29) of the main document.□

Appendix B. Proof of Lemma 1

Proof 7. According to Theorem 2, the link defined in (27) of the main document works at the state

$$\sum_s \frac{D_s}{D_s + \tilde{q}(t)} u_s(t) = \tilde{B}. \tag{B.1}$$

Let $U(t) = \sum_i u_i(t)$ be aggregation of normalized windows. Noting that $D/(D+q)$ is a strictly increasing function of D, we have, from

(B.1)

$$\frac{\min_s D_s}{\min_s D_s + \tilde{q}(t)} \cdot U(t) \le \sum_s \frac{D_s}{D_s + \tilde{q}(t)} u_s(t) \le \frac{\max_s D_s}{\max_s D_s + \tilde{q}(t)} \cdot U(t).$$

Hence,

$$1 + \frac{\tilde{q}(t)}{\min_s D_s} \ge \frac{U(t)}{\tilde{B}} \ge 1 + \frac{\tilde{q}(t)}{\max_s D_s}. \tag{B.2}$$

From (33) of the main document and (B.2), we have

$$U(t+1) = \tilde{B} + \hat{\alpha}. \tag{B.3}$$

Plugging (B.3) into (B.2),

$$\frac{\tilde{q}(t)}{\min_s D_s} \ge \frac{\hat{\alpha}}{\tilde{B}} \ge \frac{\tilde{q}(t)}{\max_s D_s}.$$

Hence, given $\epsilon' > 0$,

$$\frac{\tilde{q}(t)}{\min_s D_s} \ge \frac{\hat{\alpha}}{\tilde{B}} - \epsilon \quad \text{and} \quad \frac{\tilde{q}(t)}{\max_s D_s} \le \frac{\hat{\alpha}}{\tilde{B}} + \epsilon$$

for all sufficiently large t. This series of statements proves the lemma.□

References

Ahmed, A., Bakar, K.A., Channa, M.I., Haseeb, K., Khan, A.W., 2015. A survey on trust based detection and isolation of malicious nodes in ad-hoc and sensor networks. Front. Comput. Sci. 9 (2), 280–296.

Alizadeh, M., Greenberg, A., Maltz, D.A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., Sridharan, M., 2011. Data center TCP (DCTCP). ACM SIGCOMM Comput. Commun. Rev. 41 (4), 63–74.

Bowker, S.L., Richardson, K., Marra, C.A., 1994. TCP Vegas: new techniques for congestion detection and avoidance. ACM SIGCOMM Comput. Commun. Rev. 24 (4), 24–35.

Cai, H., Ha, S., Rhee, I., Xu, L., et al., 2007. Stochastic ordering for internet congestion control and its applications. In: Proceedings of IEEE INFOCOM 2007, IEEE, Anchorage, Alaska, USA, pp. 910–918.

Caini, C., Firrincieli, R., 2004. TCP Hybla: a TCP enhancement for heterogeneous networks. Int. J. Satell. Commun. Netw. 22 (5), 547–566.

Chakravorty, R., Katti, S., Crowcroft, J., Pratt, I., 2003. Flow aggregation for enhanced TCP over wide-area wireless. In: Proceedings of the IEEE INFOCOM'03, vol. 3, pp. 1754–1764.

Chen, M., 2006. A general framework for flow control in wireless networks (Ph.D. thesis). University of California.

Chen, M., Zakhor, A., 2006. Flow control over wireless network and application layer implementation. In: Proceedings of IEEE INFOCOM'06, pp. 103–113.

Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M., 2003. Planetlab: an overlay testbed for broad-coverage services. ACM SIGCOMM Comput. Commun. Rev. 33 (3), 3–12.

Crowcroft, J., Oechslin, P., 1998. Differentiated end-to-end Internet services using a weighted proportional fair sharing TCP. ACM SIGCOMM Comput. Commun. Rev. 28 (3), 53–69.

Cui, T., Andrew, L., 2010. FAST TCP for ns-2. URL ⟨http://www.cubinlab.ee.mu.oz.au/ns2fasttcp⟩.

Flach, T., Dukkipati, N., Terzis, A., Raghavan, B., Cardwell, N., Cheng, Y., Jain, A., Hao, S., Katz-Bassett, E., Govindan, R., 2013. Reducing web latency: the virtue of gentle aggression. ACM SIGCOMM Comput. Commun. Rev. 43 (4), 159–170.

Floyd, S., 2003. HighSpeed TCP for Large Congestion Windows, RFC 3649.

Floyd, S., Allman, M., 2008. Comments on the Usefulness of Simple Best-effort Traffic, RFC 5290.

Fu, C., Liew, S., 2003. TCP Veno: TCP enhancement for transmission over wireless access networks. IEEE J. Sel. Areas Commun. 21 (2), 216–228.

Ghaffari, A., 2015. Congestion control mechanisms in wireless sensor networks: a survey. J. Netw. Comput. Appl. 52, 101–115.

Grieco, L., Mascolo, S., 2005. Mathematical analysis of Westwood+ TCP congestion control. In: Proceedings of IEE Control Theory and Applications, vol. 152, pp. 35–42.

Ha, S., Rhee, I., Xu, L., 2008. CUBIC: a new TCP-friendly high-speed TCP variant. ACM SIGOPS Oper. Syst. Rev. 42 (5), 64–74.

Jacobson, V., 1995. Congestion avoidance and control. ACM SIGCOMM CCR 25 (1), 157–187.

Jingyuan, W., Jiang, Y., Chao, L., Ouyang, Y., Xiong, Z., 2014. Improving the incast performance of datacenter TCP by using rate-based congestion control. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. 97 (7), 1654–1658.

Karrer, R.P., Matyasovszki, I., Botta, A., PescapéA., 2007. Magnets-experiences from deploying a joint research-operational next-generation wireless access network testbed. In: 2007 3rd International Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities, TridentCom 2007, IEEE, Guangzhou, China, pp. 1–10.

Kelly, T., 2003. Scalable TCP: improving performance in highspeed wide area networks. ACM SIGCOMM Comput. Commun. Rev. 33 (2), 83–91.

Kushwaha, V., Gupta, R., 2014. Congestion control for high-speed wired network: a systematic literature review. J. Netw. Comput. Appl. 45, 62–78.

Lee, C., Jang, K., Moon, S., 2012. Reviving delay-based tcp for data centers. In: Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, ACM, Helsinki, Finland, pp. 111–112.

Linktropy Mini2 WAN Emulator, 2010. URL ⟨http://www.apposite-tech.com/products/mini2.html⟩.

Luo, L., Guo, D., Li, W., Zhang, T., Xie, J., Zhou, X., 2015. Compound graph based hybrid data center topologies. Front. Comput. Sci. 9 (6), 860–874.

Mascolo, S., Casetti, C., Gerla, M., Sanadidi, M., Wang, R., 2001. TCP Westwood: bandwidth estimation for enhanced transport over wireless links. In: Proceedings of ACM MobiCom'01, pp. 287–297.

Osborne, M.J., Rubinstein, A., 1994. A Course in Game Theory. MIT Press, Cambridge, Massachusetts, US.

Padhye, J., Firoiu, V., Towsley, D., Kurose, J., 1998. Modeling TCP throughput: a simple model and its empirical validation. In: Proceedings of the ACM SIGCOMM'98, pp. 303–314.

Rizzo, L., 2010. Dummynet. URL ⟨http://info.iet.unipi.it/luigi/ip_dummynet/⟩.

Samios, C., Vernon, M., 2003. Modeling the throughput of TCP Vegas. In: Proceedings of ACM SIGMETRICS'03, pp. 71–81.

Shorten, R., Leith, D., 2004. H-TCP: TCP for high-speed and long-distance networks. In: Proceedings of PFLDnet'04, pp. 95–101.

Tan, K., Song, J., Zhang, Q., Sridharan, M., 2006. A compound TCP approach for high-speed and long distance networks. In: Proceedings of IEEE INFOCOM'06, pp. 1–12.

Tang, A., Wang, J., Low, S.H., Chiang, M., 2007. Equilibrium of heterogeneous congestion control: existence and uniqueness. IEEE/ACM Trans. Netw. 15, 824–837.

Tian, Y., Xu, K., Ansari, N., 2005. TCP in wireless environments: problems and solutions. IEEE Commun. Mag. 43 (3), S27–S32.

Wang, J., Wen, J., Zhang, J., Han, Y., 2011. TCP-FIT: an improved TCP congestion control algorithm and its performance. In: The 30th IEEE International Conference on Computer Communications (INFOCOM 2011), Shanghai, PR China.

Wang, X.S., Balasubramanian, A., Krishnamurthy, A., Wetherall, D., 2013. Demystifying page load performance with wprof. In: Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), pp. 473–485.

Wang, J., Jiang, Y., Ouyang, Y., Li, C., Xiong, Z., Cai, J., 2013a. Tcp congestion control for wireless datacenters. IEICE Electron. Express 10 (12), 20130349.

Wang, J., Wen, J., Han, Y., Zhang, J., Li, C., Xiong, Z., 2013b. CUBIC-FIT: a high performance and TCP CUBIC friendly congestion control algorithm. IEEE Commun. Lett. 17 (8), 1664–1667.

Wang, J., Wen, J., Han, Y., Zhang, J., Li, C., Xiong, Z., 2014. Achieving high throughput and TCP Reno fairness in delay-based tcp over large networks. Front. Comput. Sci. 8 (3), 426–439.

Wang, J., Wen, J., Li, C., Xiong, Z., Han, Y., 2015. DC-Vegas: a delay-based TCP congestion control algorithm for datacenter applications. J. Netw. Comput. Appl. 53, 103–114.

Wei, D., Jin, C., Low, S., Hegde, S., 2006. FAST TCP: motivation, architecture, algorithms, performance. IEEE/ACM Trans. Netw. 14 (6), 1246–1259.

Wu, H., Feng, Z., Guo, C., Zhang, Y., 2013. Ictcp: incast congestion control for tcp in data-center networks. IEEE/ACM Trans. Netw. 21 (2), 345–358.

Xu, L., Harfoush, K., Rhee, I., 2004. Binary Increase Congestion Control (BIC) for fast, long distance networks. In: Proceedings of IEEE INFOCOM'04, pp. 2514–2524.

Xu, W., Zhou, Z., Pham, D., Ji, C., Yang, M., Liu, Q., 2011. Hybrid congestion control for high-speed networks. J. Netw. Comput. Appl. 34 (4), 1416–1428, Advanced Topics in Cloud Computing.

Yu, B., Han, Y., Yuan, H., Zhou, X., Xu, Z., 2015. A cost-effective scheme supporting adaptive service migration in cloud data center. Front. Comput. Sci. 9 (6), 875–886.

Zhou, B., Fu, C., Chiu, D., Lau, C., Ngoh, L., 2006. A simple throughput model for TCP Veno. In: Proceedings of IEEE ICC'06, vol. 12, pp. 5395–5400.