Computer Communications 105 (2017) 44–52

Contents lists available at ScienceDirect

Computer Communications

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

Adaptive low-priority congestion control for high bandwidth-delay product and wireless networks☆

Xianliang Jiang[a,b], Guang Jin[a,*]

[a] Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo, China
[b] College of Computer Science and Technology, Zhejiang University, Hangzhou, China

ARTICLE INFO

Article history:
Received 18 May 2016
Revised 7 October 2016
Accepted 18 November 2016
Available online 25 November 2016

Keywords:
High bandwidth-delay product
Wireless network
Congestion control
Low priority
Congestion level

ABSTRACT

The low-priority service is an exciting and attractive choice for networking applications (e.g. automatic update, backup, peer-to-peer file share) which create traffic that is considered less urgent than that of others and become a renewed interest at the Internet Engineering Task Force (IETF). A low-priority protocol, which provides the low-priority service, can exploit the residual bandwidth of the bottleneck link and achieve the high throughput, low-latency data delivery in traditional networks. However, due to the conservative and inappropriate congestion control mechanisms, the existing low-priority protocols (e.g. LEDBAT) cannot effectively utilize the residual bandwidth of the bottleneck link in high bandwidth-delay product (HBDP) and wireless networks. In this paper, we propose an adaptive Congestion level-based Low-Priority congestion Control (CLPC) protocol to improve the efficiency of low-priority protocols and maintain the low-priority features. Specifically, the CLPC sender adopts an one-way path delay to estimate the congestion level and adjust the aggressiveness of congestion control mechanisms. Different from other low-priority protocols, the CLPC protocol is more aggressive when the bottleneck link of HBDP and wireless networks has residual bandwidth. This makes a faster convergence of link utilization in HBDP networks. Combining the random loss detection, CLPC can achieve the high throughput in wireless networks. The extensive simulations in NS-2 show that CLPC can improve the transmission performance significantly as compared to other low-priority protocols in HBDP and wireless networks. Furthermore, we implement the CLPC protocol in the Linux kernel (3.13) and setup a testbed to measure it. The results also indicate the feasibility and effectiveness of CLPC.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

The Transmission Control Protocol (TCP), which adopts the Additive Increase Multiplicative Decrease (AIMD) mechanism [1], is an usual choice of the bulk data transfer over high-speed links. This may be driven by the requirements of the end-to-end reliable data delivery. Because of the built-in congestion control mechanisms, picking TCP as an end-to-end protocol means employing a given way of sharing network resources (buffers, bandwidth, ⋯) among competing flows. Under ideal conditions, long-lived TCP flows sharing a bottleneck link at a best-effort manner tend to obtain the fair share of bottleneck bandwidth. However, the fair share is not necessarily always the best objective for different applications. For example, TCP-based unattended applications, e.g. automatic update, may be less urgent than others and can tolerate longer flow-completion time than interactive web applications. In addition, a TCP-based bulk data transfer for unattended applications may saturate the bottleneck buffer and cause a large queueing delay that affect the performance of coexisting interactive applications.

To tackle aforementioned issues, the low-priority protocol (LPP), which achieves the low-priority service (LPS) by inferring and reacting to the occurrence of the network congestion on a network path earlier than the standard TCP, was proposed. Note that the bandwidth reservation can also ensure the performance of interactive applications [2]. In [3], Ros et al. regards the low-priority as a service which results in a smaller bandwidth and delay impact on the standard TCP than standard TCP itself when both low-priority and standard TCP flows share a bottleneck link. It also means that LPS allows latency-sensitive flows to occupy more available bandwidth when they need it. Actually, delay-based protocols, e.g. Ve-

☆ Manuscript received May xx, 2016. This work was supported in part by the National Natural Science Foundation of China (61601252); Natural Science Foundation of Zhejiang Province (LY12F02013) of China; Ningbo Municipal Technology Innovation Team (2011B81002) of China.

* Corresponding author.
E-mail addresses: jiangxianliang@nbu.edu.cn (X. Jiang), jinguang@nbu.edu.cn (G. Jin).

http://dx.doi.org/10.1016/j.comcom.2016.11.007
0140-3664/© 2016 Elsevier B.V. All rights reserved.

gas [4] and FAST [5], can detect and handle the incipient congestion earlier than loss-based protocols, e.g. Reno [1] and CUBIC [6]. Hence, delay-based protocols have the LPS characteristics when they coexist with loss-based protocols. In implementation, the LPS characteristics can be achieved through a mechanism that resembles TCP but exhibits a more cautious behavior, e.g. increasing the congestion window by less than one packet per round-trip time (RTT) in congestion avoidance. However, due to the conservative increasing and aggressive decreasing of the congestion window, LPP can not effectively utilize the residual bandwidth of the bottleneck link(s) in high bandwidth-delay product (high-BDP, named HBDP) and wireless networks. Therein, the BDP, which can be calculated by *bandwidth* × *delay*, means that the "network pipe" has a large capacity (e.g. in satellite network, wide area network). The HBDP indicates that TCP has a large congestion window. Furthermore, the wireless means that the complex transmission channel would cause the high packet loss rate. In short, the study of LPP is still a challenge in HBDP and wireless networks.

To understand why LPP performs poorly in HBDP and wireless networks, one should has insights into the operation of LPP. Note that LPP inherits main features of standard TCP while exhibits the low-priority characteristics. LPP regards the timeout or triple duplicate acknowledgements (ACKs) [3] events as an indication of packets loss and reduces the size of the sender's congestion window. This mechanism usually works well in traditional (not HBDP) networks. Unfortunately, due to the existence of the random packet loss caused by the high link error probability, fading, and interference *etc*., existing LPPs may reduce the congestion window unnecessarily and result in a poor performance in wireless networks. Furthermore, existing LPP congestion avoidance algorithms adopt the similar AIMD mechanism like standard TCP. This makes LPPs cannot utilize the network bandwidth efficiently, especially in HBDP networks. The LPP sender increases the congestion window (*cwnd*) by one or less than the maximum segment size (MSS) every RTT when the bottleneck link has residual bandwidths, which is inferred from the one-way path delay. Otherwise, the LPP sender reduces *cwnd* by half when triple duplicate ACKs are received or reduces *cwnd* to one (2 or 4 probably [7]) when the retransmission timeout occurs. In HBDP networks, LPP requires a large window to fully utilize the network capacity. According to above analyses, one can see that LPPs need a long time to fully utilize the bottleneck bandwidth after that *cwnd* is reduced by half or to one.

In this paper, we propose an adaptive **C**ongestion level-based **L**ow-**P**riority congestion **C**ontrol (**CLPC**) protocol to resolve above issues in HBDP and wireless networks. The idea is inspired by LEDBAT [8], CLTCP [9] and TCP-FIT [10]. Therein, LEDBAT has been introduced into BitTorrent and standardized by the IETF recently. CLTCP is our previously proposed algorithm which uses the bit stream of explicit congestion notification (ECN) to measure the extent of network congestion and adaptively adjust the TCP congestion control mechanism. TCP-FIT uses N virtual Reno sessions, which can be adjusted according to the end-to-end delay, to simulate a single Reno session. Different from other state-of-the-art LPP schemes, CLPC can occupy the residual bottleneck bandwidth rapidly in HBDP and wireless networks and keep the LPS gracefully. We conduct extensive experiments in the packet-level simulator and results show that CLPC outperforms other LPP algorithms in HBDP and wireless networks. We also implement the CLPC protocol in end-hosts with Linux kernel version 3.13 and construct a testbed to verify the feasibility of CLPC.

The rest of the paper is organized as follows. In Section 2, the related work is reviewed. We describe the key idea and components of CLPC in Section 3. In Section 4, we evaluate the effectiveness of CLPC using packet-level simulations. And at last, the conclusion is concluded in Section 6.

2. Related work

A variety of congestion control algorithms, which are used in different TCPs for different purposes, have been proposed to achieve the high link utilization in HBDP and wireless networks. They have their own merits and shortcomings respectively. Generally, these protocols can be classified into two categories: *router-assisted* and *end-to-end*. Therein, the *router-assisted* congestion control algorithms, like XCP [11], RCP [12], BMCC [13], perform well in HBDP and wireless networks but confront the challenges of incremental deployment and backward compatibility with legacy TCPs. Accordingly, the *end-to-end* congestion control algorithms are more attractive since they do not require any special support from routers in the core networks. Actually, the *end-to-end* congestion control [14] relies on the packet loss or delay to detect the network congestion. And related works can be further divided into loss-based congestion control (LCC), delay-based congestion control (DCC), and the synergy of both LCC and DCC.

The LCC algorithms, which are widely adopted in the current Internet, perform the congestion control reactively by considering the packet loss. To improve the performance of LCC algorithms in HBDP networks, most TCP variants, e.g. HSTCP [15], CUBIC [6], Agile-SD [16], modify the additive increase and multiplicative decrease factors of the TCP's congestion control to achieve the high link utilization and throughput rapidly. However, the aggressiveness in adjusting *cwnd* intensifies the oscillation of the TCP throughput. Existing researches also indicated that most TCP variants cannot obtain the fair share of bottleneck bandwidth when competing with the standard TCP. How to achieve the high throughput and maintain the TCP-friendly is a dilemma. Wang et al. proposed a virtual parallel TCP [10] performing gracefully in both HBDP and wireless networks. To achieve near-optimal throughputs while preserving TCP-friendly and fairness, Mittal et al. [17] proposed a recursively cautious congestion control algorithm coupling the standard TCP with LPP.

The DCC algorithms, which are more efficient in the stable networks, assume that the increasing of packets' RTT indicates the coming of the network congestion and attempt to proactively adjust *cwnd* based on the variation of packets' RTT. In [23], the first DCC scheme was proposed in an interconnected and heterogeneous computer network. The author believes that the optimal *cwnd* is related to the gradient of the *delay-window* curve. Thereafter, Vegas [4], which detects the network congestion by observing the changes of the sending rate, was proposed to improve the performance of TCP Reno. To adapt to HBDP networks, FAST [5], which adopts the minimum RTT to detect the network congestion, was proposed to grab the network bandwidth rapidly. To solve the increasing queue backlog when the number of flows increases, Tan et al. proposed an enhanced FAST, which can achieve the (α, n)-proportional fairness [18,19], based on the virtual link price. In [20], Ge et al. analyzed the impacts of two-way FAST flows. Jung et al. proposed ACP [21], which combines the estimation of the bottleneck queue size and a measure of fair sharing, to achieve the high utilization, fair sharing of the bottleneck bandwidth, and fast convergence. To keep the low packet latency while delivering the bulk data, Mittal et al. [22] proposed to use RTT gradients to adjust the sending rate. Actually, the inaccurate RTT measurement would make DCC algorithms unstable. In addition, DCC algorithms also suffer from significantly low throughput if the competing flows are LCC ones, e.g. Reno.

Considering the advantages of the LCC and DCC algorithms, several researchers focused on the synthetical algorithms of both categories. In [24], Tan et al. proposed the compound TCP by adding a scalable delay-based component into the standard TCP. Xu et al. [25] regarded the queuing delay as the primary congestion indicator and the packet loss as the second congestion indicator,

and then proposed a hybrid congestion control algorithm. Combine FAST and CUBIC, Wang et al. [26] proposed an adaptive framework to automatically switch among existing congestion control mechanisms according to the change of the network status.

To meet the requirements of LPS applications, the study of the congestion control mechanism of LPP attracts more and more attentions. In [14], Afanasyev et al. summarizes and analyzes the end-to-end TCPs and considers LPS as differential services for providing user-perceived quality of service (QoS). In [3], Ros and Welzlinvestigate end-host approaches, including the upper-layer methods or techniques, to reduce the impact on other competing flows and achieve LPS. Overall, there are three typical algorithms, namely TCP NICE [27], TCP LP [28], and LEDBAT [8], for LPS in the transport and application layer. Thereinto, TCP NICE follows the same mechanism as TCP Vegas but changes the congestion response from the moderate linear-decrease to the aggressive non-linear decrease (one per RTT in most cases, like standard TCP). TCP LP utilizes only the residual bottleneck bandwidth as compared to non-LPPs and uses the one-way packet delay to detect the early congestion. LEDBAT is a windowed protocol, governed by a linear controller designed to infer the occurrence of the network congestion earlier than TCP. It's also based on the one-way delay estimation. After analyzing the existing LPP schemes, we find that they are designed and implemented based on the standard TCP and face the similar drawbacks of the standard TCP in HBDP and wireless networks. Hence, we try to solve these issues using the CLPC protocol. It can achieve better convergence, fairness, robustness than other LPPs.

3. The CLPC protocol

This section presents the core idea of the CLPC protocol and discusses how to estimate the network congestion state and distinguish the random packet loss of wireless links. After that we design a mechanism to update the congestion window of CLPC when the network state is in congestion avoiding stages.

3.1. CLPC overview

To better understand the motivations behind CLPC, let's review the standard TCP and LEDBAT [8] firstly. The former uses the packet loss as an indication to back off. It means that the standard TCP would fills the buffer and causes a huge delay under a drop-tail FIFO queuing discipline. This may hurt the performance of interactive applications. To tackle this issue, LEDBAT, customized for the transmission of non-interactive traffic, was proposed. It can quickly yield to other TCPs and keep low delay when no other traffic arise. However, due to the conservative increasing, LEDBAT can not effectively utilize the residual bandwidth of the bottleneck link(s) in HBDP and wireless networks.

To solve above issue, we propose CLPC to improve the bottleneck link utilization and maintain the characteristics of LPS. The main design goals of CLPC are as following.

- It can saturate the bottleneck link with the residual bandwidth rapidly, but quickly yield to TCPs and its variants when the bottleneck link is close to saturation.
- It can keep the low delay when no other traffic appears.
- It can operate well with the first-in-first-out queuing disciplines.

Specifically, CLPC adopts the similar method like LEDBAT to achieve the low-priority. Yet, different from LEDBAT, CLPC uses an one-way path delay (instead of RTT [29]) to estimate the extent of the network congestion (called "congestion level") and implement an adaptive congestion control in congestion avoiding stages. It can achieve the high throughput when the bottleneck link has residual capacities and maintain the low-priority characterize when the bottleneck link is high-load in HBDP and wireless networks. Roughly, the CLPC protocol consists of three core components, namely congestion state estimation (CSE), random loss discrimination (RLD), and congestion control function (CCF), as shown in Fig. 1. Their functions are listed as following.

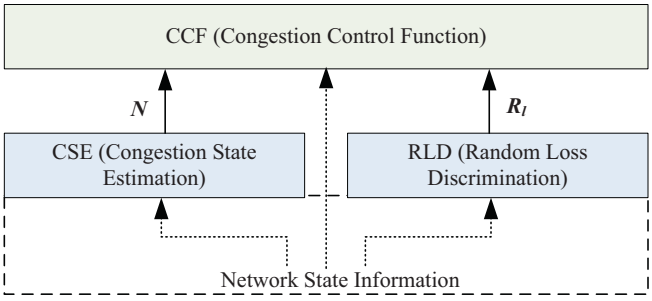


Fig. 1. The main components and their relationship of the CLPC protocol.

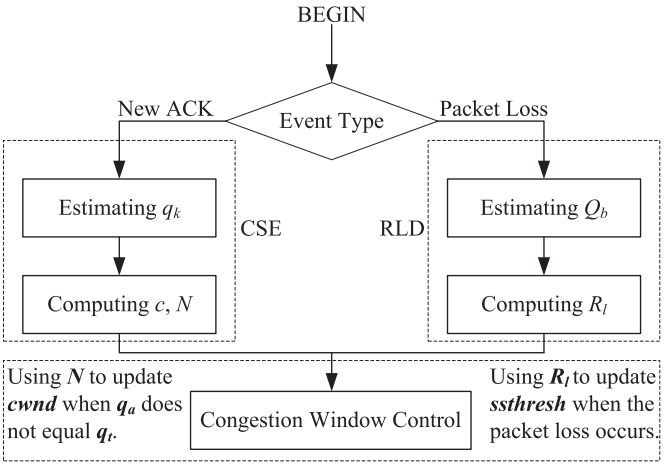


Fig. 2. The modified congestion control mechanism in the CLPC protocol.

- The CSE estimates the extent of the network congestion and then adjusts the aggressive factor of the CLPC protocol based on the estimated value.
- The RLD uses the estimation of the queue backlogging to distinguish between the congestion-induced loss and the random loss.
- The CCF regulates the congestion window of CLPC according to the results of CSE and RLD.

Fig. 2 is the flow chart of CLPC. It can be seen that two different procedures are adopted to update the congestion window of CLPC according to the event type. If a new ACK event occurs, CLPC uses N to update $cwnd$ when q_a not equals q_t. If a packet loss event occurs, CLPC uses R_l to update the $ssthresh$ value of CLPC. Note that q_k is the queueing delay deduced from the packet's one-way path delay. α denotes the extent of the network congestion and its value is in [0, 1]. N is an aggressive factor of CLPC and can make CLPC to perform well in HBDP and wireless networks. Q_b is the backlog of the bottleneck queue. Note that the packet loss event in Fig. 2 is triggered when triple duplicate ACKs is received.

3.2. Congestion state estimating

To achieve the adaptive LPS in the presence of the standard TCP, the CSE estimates the extent of network congestion and then

translate it into a normalized value, α, using Eq. (1).

$$\alpha = \begin{cases} q_a/q_t, & \text{if } q_a \le q_t, \quad (a) \\ 1, & \text{if } q_a > q_t, \quad (b) \end{cases} \tag{1}$$

where, q_a is the average queuing delay and q_t is the target queuing delay (like *TARGET* in LEDBAT). Suppose that q_k, obtained using the similar way as [8], and γ denotes the current queuing delay of the k_{th} packet and the smoothing parameter respectively. q_a can be calculated by Eq. (2) using an exponentially weighted moving average.

$$q_a \leftarrow (1-\gamma)\cdot q_a + \gamma \cdot q_k \tag{2}$$

After the aforementioned estimation, the CSE can use α to calculate the aggressive factor N, which is initialized to 1, periodically (e.g., every 10 ms) using Eq. (3).

$$N = N + 1 - \alpha \cdot N \tag{3}$$

Note that q_k equals the current one-way path delay minus the base one-way path delay. And CSE adopts the timestamps to measure the one-way path delay.

3.3. Random loss distinguishing

How to distinguish the types of packets loss, the random loss caused by unstable wireless link and the congestion-induced loss caused by buffer overflow, and provide an efficient method to deal with them is still a challenge. To identify the random loss, RLD adopts the similar method like [30] in this paper. In the case of random packets loss, the intermediate buffer has no backlogging in most cases. Otherwise, some packets have been queued in the intermediate buffer when the network congestion occurs. The details are shown in Eqs. (4)–(7). We consider that the RTT of the LPP connection suffers from the interference of the backward path congestion. So the RLD use the one-way path delay instead of the RTT to measure the backlog at the bottleneck queue. If the packet loss is detected and the backlog of this connection increases, RLD regards the loss as congestion-induced. Otherwise, it's a random loss.

Next, we present the detailed steps to discriminate the random loss. In Vegas [4], we have known that the sender uses the congestion window (*cwnd*) and RTT to estimate the sending rate (S_r) by

$$S_r = cwnd/rtt \tag{4}$$

where *rtt* is the smoothed RTT. Let's assume $rtt_{\min}$ denotes the base round trip time. If $rtt > rtt_{\min}$, the packets of this connection accumulate at the bottleneck queue. And the backlog at the bottleneck queue is denoted by Q_b. We have

$$rtt = rtt_{min} + Q_b/S_r \tag{5}$$

Solving Eq. (5), we can get

$$Q_b = S_r \cdot (rtt - rtt_{min}) = S_r \cdot q_k \tag{6}$$

Note that RLD uses the one-way path delay to measure q_k. Similar to [30], we use Q_b as an indication of whether the connection is in a congestion state or not. The expression is

$$R_l = \begin{cases} 1, & \text{if } Q_b \le \beta, \quad (a) \\ 0, & \text{if } other, \quad (b) \end{cases} \tag{7}$$

Specifically, if $Q_b \le \beta$ and a packet loss is detected, RLD considers the loss is random rather than congestion-induced. Otherwise, RLD considers the loss as congestion-induced. In CLPC, $\beta = 3$ like that in [30].

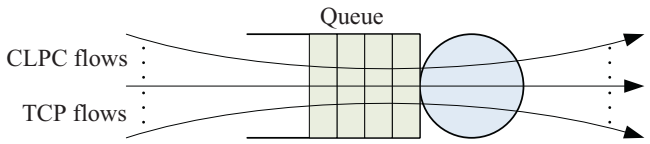


Fig. 3. The schematic diagram for analyzing the low-priority feature and efficiency of CLPC when coexist with legacy TCPs.

3.4. Congestion control function

After the results of CSE and RLD are obtained, CCF adopts a piecewise function to increase the congestion window of CLPC when a new ACK is received (lines 1–10 of Algorithm 1). When a packet loss occurs, CCF use a trade-off method (an aggressive decrease for the congestion loss and a cautious decrease for the random loss), which can achieve the high throughput in wireless networks, to update the *ssthresh* variable of CLPC (lines 12–17 of Algorithm 1).

Algorithm 1 Congestion Window Update.

```
1: //1: For Each ACK
2: C_d ← OWD;
3: B_d ← MIN(B_d, C_d);
4: Q_d ← C_d − B_d;
5: O_target ← Q_target − Q_d;
6: if Q_d == 0 then
7:    cwnd+ = N·G·O_target / cwnd;
8: else
9:    cwnd+ = G·O_target / cwnd;
10: end if
11:
12: //2: For Packet Loss
13: if R_l == 1 then
14:    ssthresh ← cwnd · η_1;
15: else
16:    ssthresh ← cwnd · η_2;
17: end if
```

The detailed procedure of CCF is shown in Algorithm 1. Therein, C_d is the current one-way path delay, *OWD*, which is obtained from the timestamp option in the TCP header. B_d is the global minimum one-way path delay. Q_d is the queuing delay calculated by the line 4 of Algorithm 1. O_{target}, a reference of Q_d, is the target queuing delay like the *TARGET* in [8]. N, which can be calculated by Eq. (3), is the result of CSE. $G = \frac{1}{Q_{target}}$ is a gain factor like [8]. As can be seen from lines 6–10 of Algorithm 1, the *cwnd* increases very aggressively when the bottleneck link has residual bandwidth. Otherwise, the increasing of *cwnd* is conservative (less than one per RTT). When a packet loss is detected, CCF updates the *ssthresh* variable according to the R_l of RLD. $\eta_1 = 0.8$ and $\eta_2 = 0.5$ are two controlled factors.

3.5. Discussion

To investigate the performance of the CLPC protocol, we use a simple scenario (as Fig. 3) to analyze its efficiency and the low-priority feature. In the scenario, several CLPC flows and standard TCP flows share the same bottleneck link and queue. Suppose that U flows, generated by the standard TCP and/or CLPC, indexed by $u \in \{1, \cdots, U\}$ share a bottleneck link simultaneously. The bandwidth of the bottleneck link is *BW*. When U flows pass through the same bottleneck link and queue, the relationships among the bottleneck queue length q, the congestion packet loss rate p and the aggregate

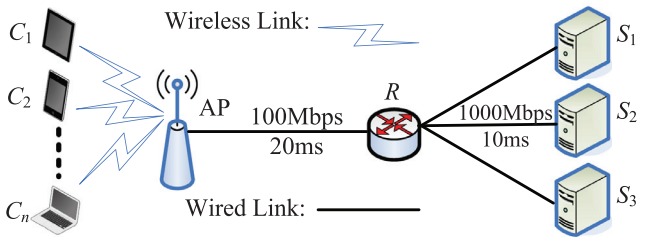


Fig. 4. The experimental topology, which includes lossy links denoted by wireless links, for evaluating CLPC.

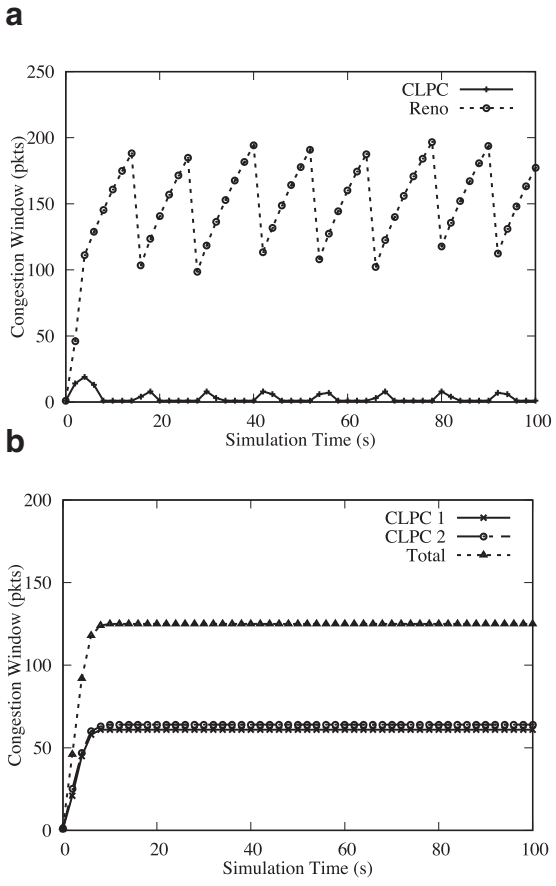


Fig. 5. Temporal evolution of the congestion window. (a) One Reno flow and one CLPC flow; (b) two CLPC flow.

throughput of U flows, $T_U = \sum_{u \in U} T_u$, are given by

$$T_U = \sum_{u \in U} T_u = \begin{cases} \leq BW, & \text{if } p = 0,\ q = 0, \quad (a) \\ = BW, & \text{if } p = f(q),\ q > 0, \quad (b) \end{cases} \tag{8}$$

where q is the number of data packets in the bottleneck queue. T_u is the throughput of flow u. And $f(.)$ is a monotone non-decreasing function when $q > 0$. In this paper, we assume that $f(0) = 0$.

Now, we assume that the bottleneck link works in (a) of Eq. (8) when CLPC flows arise. The aggregated throughput, T_U, will be less than or equal BW. Since $q_a = 0$, α is zero according to Eq. (1). Hence, $N \to +\infty$ according to Eq. (3). Using lines 7 and 9 of Algorithm 1, the bottleneck link bandwidth is rapidly filled. And this makes T_U equal BW and $q > 0$, which contradicts the preceding assumption. Therefore, the bottleneck link works in (b) of Eq. (8). It means that CLPC can achieve the high utilization of the bottleneck link.

Next, we analyze the low-priority feature of CLPC. As can be seen from above analyses, CLPC flows make the bottleneck link

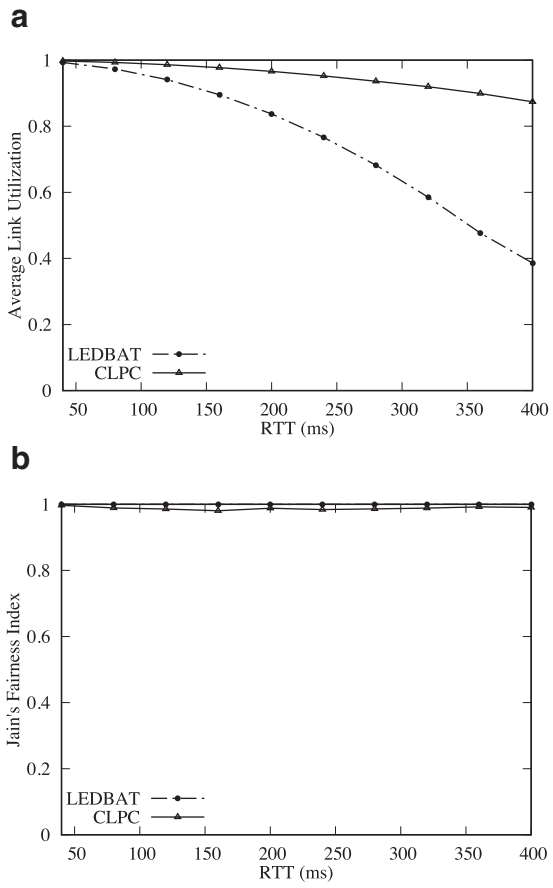


Fig. 6. Average link utilization and fairness of LPP under different RTTs. (a) Average link utilization; (b) fairness.

running in (b) of Eq. (8). This results in $q > 0$. Using the line 9 of Algorithm 1, CLPC can detect the incipient congestion earlier than that of the standard TCP and reduce its sending rate. Therefore, CLPC can meet the low-priority demand. Note that $N \to 1$ when q is always greater than 0. Note that we conduct an experiment (in Section 4) to explain the convergence and robustness of CLPC instead of the theoretical analysis [31].

4. Evaluation

To illustrate the performance of CLPC, we use a canonical packet-level simulator NS-2 [32], which has been extended with the CLPC protocol[1], to conduct several experiments.

We compare Reno [1], CUBIC [6], Veno [30], TCP-LP [28], and LEDBAT [8]. Therein, the first four schemes have been integrated into NS-2 and their parameter settings are the optimum values recommended by their authors respectively. And we add LEDBAT into NS-2 according to the authors' code of [8]. Without any further statements, $\gamma = 0.25$ is an empirical value of CLPC. Q_{target} has the same value as *TARGET* (25 ms) of LEDBAT. Furthermore, the experimental network topology is depicted in Fig. 4. C_i (means the ith client) and S_i (means the ith server) are the source hosts deployed compared protocols and destination hosts. The bandwidth and delay of the bottleneck link are 100 Mbps and 20 ms respectively. Other links are 1000 Mbps and 10 ms respectively. All queues use DropTail to manage all arrival packets. The flows' RTT is 80 ms and the bottleneck queue size equals the bandwidth-delay product, or

[1] http://www.thinkmesh.net/jiangxl/research/paper/clpc.html. The CLPC is also implemented in Linux kernel (3.13).

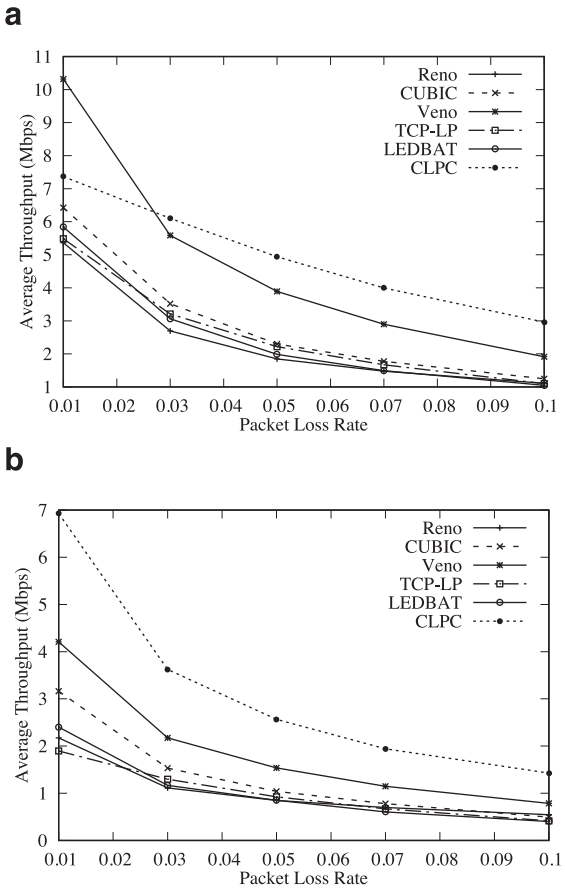


Fig. 7. Average throughput of different protocols over wireless network. (a) RTT = 80 ms; (b) RTT=200 ms.

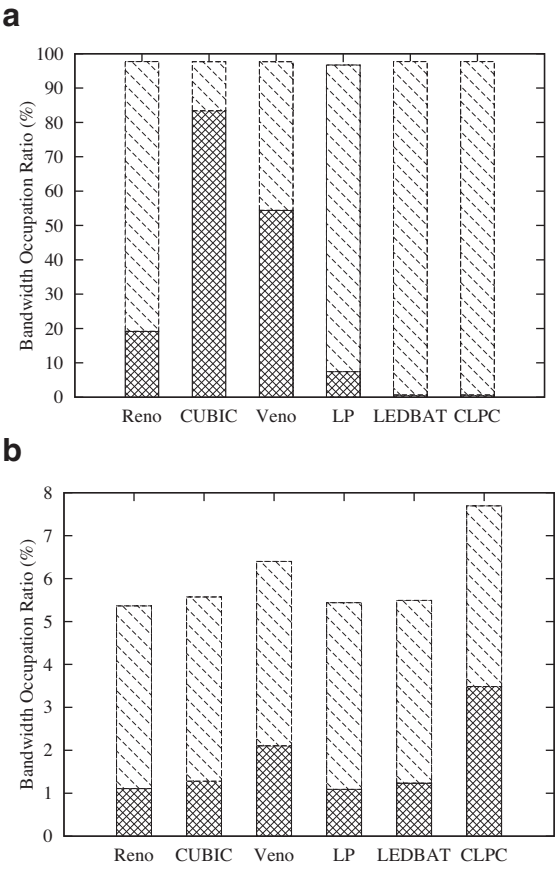


Fig. 8. Bandwidth occupation of different protocols competing with Reno. (a) packet loss rate = 0%; (b) packet loss rate = 1%.

two packets per-flow, whichever is larger. The size of data packets, which are transferred by compared protocols, is 1000 bytes, while the size of ACK packets is 40 bytes. The $RTO_{\min}$ of all compared protocols is 200 ms. Note that we use an uniform packet loss method to simulate wireless links approximately. All flows traverse the same bottleneck link and all simulations run at least 100 s unless specified otherwise.

To validate the low-priority feature of CLPC, we consider one CLPC flow competing for the bottleneck bandwidth with either one Reno or another CLPC flow firstly. Both flows start at $t = 0$ ms (the queue is empty) so that CLPC is able to accurately measure the base delay. Note, the bottleneck bandwidth is 10 Mbps in this experiment. Fig. 5(a) shows the temporal evolution of CLPC and Reno windows. It denotes that CLPC can achieve the LPS. Fig. 5(b) is the temporal evolution of CLPC and CLPC windows. It denotes that CLPC can achieve the intra-fairness. In [8], LEDBAT has the similar results.

Secondly, we compare the average link utilization and fairness of LEDBAT and CLPC in a HBDP network scenario. We vary the flows' RTT in [40 ms, 400 ms] and set five flows for different LPPs. The simulation lasts for 300 s and the fairness metric is Jain's Fairness Index (JFI) like [5].

$$JFI = \frac{(\sum_{i=1}^{n} x_i)^2}{(n \cdot \sum_{i=1}^{n} x_i^2)} \tag{9}$$

where x_i is the rate of the ith flow. n is the number of measured flows. *JFI* lies in $[\frac{1}{n}, 1]$. When $n \to \infty$, the *JFI* is in [0, 1]. Other configured parameters are default as previous. As shown in Fig. 6(a), the average link utilization of CLPC is better than that of LEDBAT. It also proves that CLPC has better convergence than LEDBAT. The fairness of different LPPs is shown in Fig. 6(b). It is observed that CLPC can achieve the intra-flow fairness like LEDBAT.

Thirdly, we evaluate different LPP and TCP variants in a wireless network scenario. And we present two cases. In the first case, we fix the flows' RTT to 80 ms and set five measured flows, which have the identical algorithm, in one experiment. And in the second case, we fix the flows' RTT to 200 ms and the number of measured flows is five too. In two cases, we vary the packet loss rate of the bottleneck link in [0.01, 0.1] and measure the average throughput of different algorithms. Other configured parameters are default as previous. Note, for the sake of consistency, we enable slow-start in LEDBAT and CLPC implementations. As shown in Fig. 7, CLPC performs better than other compared protocols. They also reflect that CLPC can improve the throughput in the wireless network scenario.

Fourthly, we measure the bandwidth occupation of LPP and TCP variants competing with the TCP Reno when the packet loss rate is 0% and 1% respectively. For the convenience, we set four Reno flows and one other LPP or TCP variant flow to compete for the bottleneck capacity. Other configured values are default as previous. As shown in Fig. 8(a), the bandwidth occupation of CLPC is 0.6%, which is the same value as that in LEDBAT, when the packet loss rate is 0%. It also indicates that CLPC can achieve the LPS feature when the bottleneck link is over-utilized. Since the bottleneck link has residual capacity when the packet loss rate is 1%, CLPC can gain more available bandwidth than others as shown in Fig 8(b). This benefits from the adaptive adjustment of congestion control mechanism of CLPC. Note that the grey grid part and white grid part represent the bandwidth occupation of LPP or TCP variants and the TCP Reno respectively.

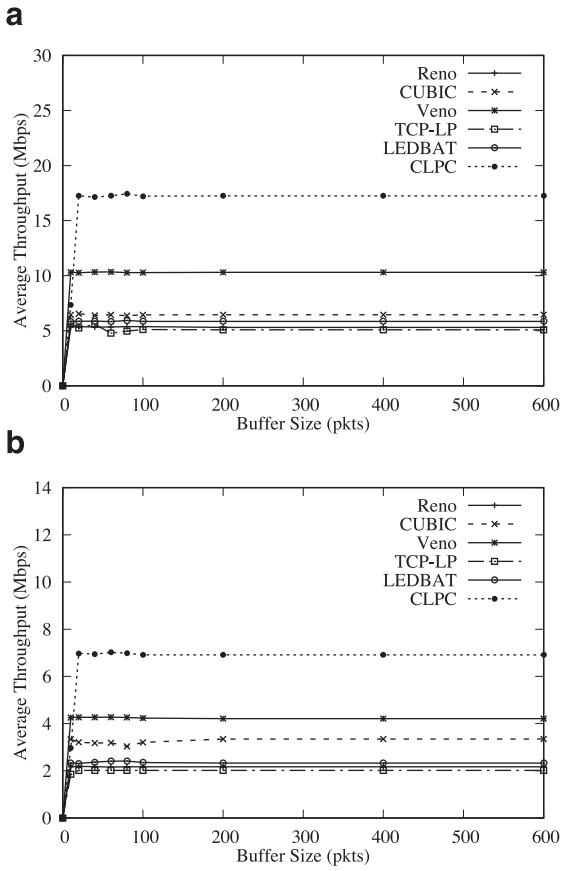


Fig. 9. Average throughput of different protocols under different buffer size. (a) RTT = 80 ms; (b) RTT = 200 ms.

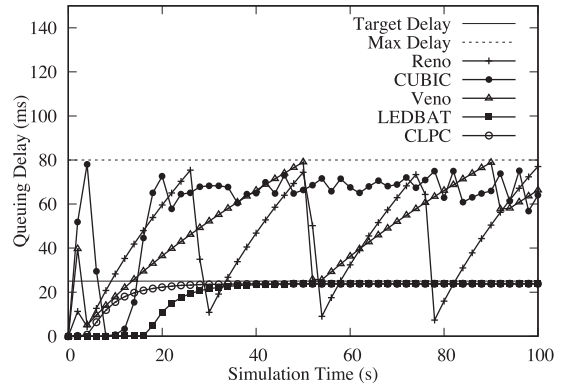


Fig. 10. The queuing delay of compared protocols. The target and maximum queuing delay are 25 ms and 80 ms respectively.

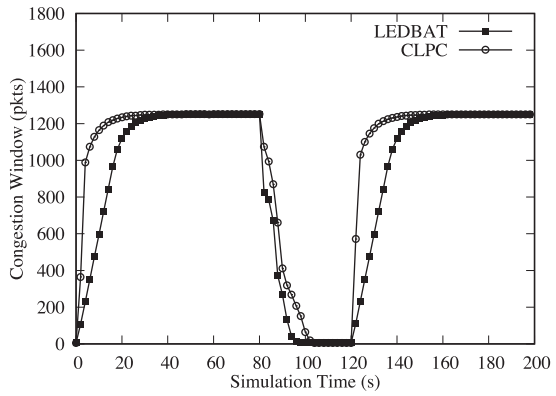


Fig. 11. The robustness and convergence of CLPC and LEDBAT in the condition of burst traffic.

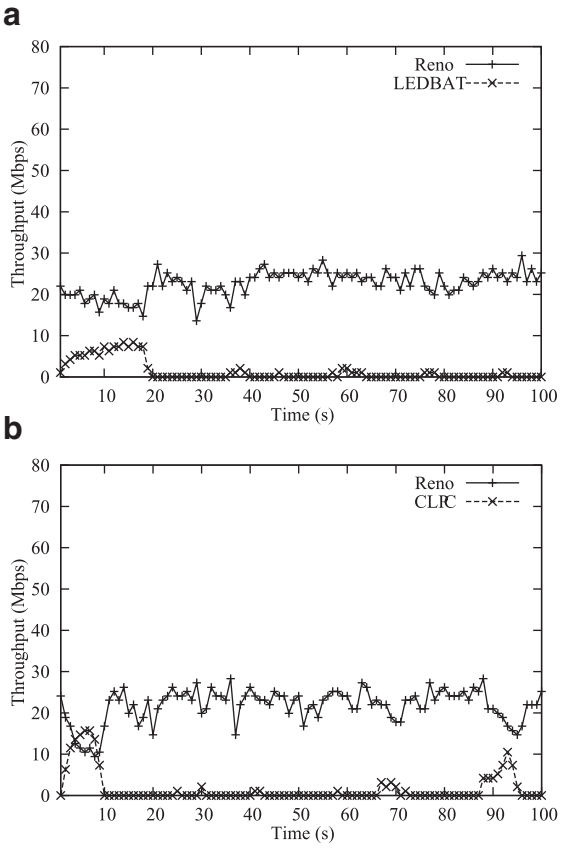


Fig. 12. The testbed measurement of CLPC and LEDBAT. (a) LEDBAT vs. Reno; (b) CLPC vs. Reno.

To measure the average throughput of different protocols under different buffer sizes, we adopt the similar parameters setting as Fig. 7 except the packet loss rate and buffer size. In this experiment, the packet loss rate is 0.01. The buffer size varies in [10, 600] packets. Other configured parameters are default as previous. Note, for the sake of consistency, we enable slow-start in LEDBAT and CLPC implementations. As shown in Fig. 9, CLPC performs better than other compared protocols. These results also reflect that CLPC can achieve better throughput with different buffer sizes in most cases (≥20 packets).

To prove the low queuing delay of CLPC, we adopt the default parameters, which are explained in the beginning of this section, to conduct an experiment. Five flows share the bottleneck bandwidth. Fig. 10 shows the queuing delay of compared protocols. We can find that CLPC and LEDBAT have the lowest queuing delay (close to 25 ms). The other protocols have large queuing delay in most cases.

In addition, to prove robustness and convergence of CLPC, we adopt the default parameters to conduct an experiment as well. Five long-term flows start at 0 s and stop at 200 s. One hundred burst flows start at 80 s and stop at 120 s. All flows share the bottleneck bandwidth. The simulation lasts for 200 s. Fig. 11 shows the evolution of congestion windows of CLPC and LEDBAT. We find that CLPC can fully utilize the bottleneck bandwidth faster than LEDBAT and keep the low-priority feature when coexists with legacy TCP flows.

5. Linux implementation

The Linux kernel supports the pluggable congestion control. Considering the convenience and compatibility, we implement the CLPC protocol in Linux kernel with version 3.13. It requires very few changes at end-hosts and none at routers. The implemented functions include RLD, CSE and CCF, which are detailed described in Section 3, with GCC 4.8.4.

To verify the practicability of the CLPC protocol, we setup a testbed to measure it. The testbed consists of two linux machines with wireless NIC (100 Mbps) and a wireless router (300 Mbps). One machine act as client and the other as a server. To measure the throughput of different flows, we use a tool called "Iperf"[2], which is widely used in the TCP measurement. In experiments, we set two flows competing the bandwidth of the wireless router. One is TCP Reno flow and the other is CLPC or LEDBAT flow. As can be seen from Fig. 12, the CLPC flow and the LEDBAT flow occupy few bandwidth of the wireless router when the Reno flow arises. This also indicates the feasibility of the CLPC protocol with low-priority features.

6. Conclusion

In this paper, we have proposed an adaptive congestion control algorithm, named CLPC, to achieve the efficient low-priority bulk data transfer in HBDP and wireless networks. The CLPC sender adopts one-way path delay to estimate the extent of network congestion and adjust the aggressiveness of the congestion control method. Through the modified sender-side, CLPC can rapidly and fully utilize the residual bandwidth of the bottleneck link, and maintain the characteristics of the low-priority coexisting with legacy TCPs when the link delay and the packet loss rate varies. Theoretical and experimental results show that the CLPC can gain significant performance improvements in the convergence, throughput and robustness. In future, we consider to extend CLPC into other networked environments. In addition, we also investigate the applicability of CLPC and integrate the cross-layer information into the design of CLPC.

References

[1] J. Postel, Transmission control protocol, IETF (September 1981). http://www.ietf.org/rfc/rfc793.txt.

[2] S. Hu, W. Bai, K. Chen, et al., Providing bandwidth guarantees, work conservation and low latency simultaneously in the cloud, in: Proc. IEEE INFOCOM, 2016. Available: http://www.cse.ust.hk/~kaichen/papers/trinity-infocom16.pdf.

[3] D. Ros, M. Welzl, Less-than-best-effort service: a survey of end-to-end approaches, IEEE Commun. Surv. Tut. 15 (2) (June 2013) 898–908.

[4] L.S. Brakmo, L.L. Peterson, TCP vegas: end to end congestion avoidance on a global internet, IEEE J. Sel. Areas Commun. 13 (8) (October 1995) 1465–1480.

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

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

[7] J. Chu, N. Dukkipati, Y. Cheng, et al., Increasing TCP's initial window, IETF (April 2013). https://tools.ietf.org/html/rfc6928.

[8] S. Shalunov, G. Hazel, J. Iyengar, et al., Low extra delay background transport (LEDBAT), IETF (October 2011). http://tools.ietf.org/html/draft-ietf-ledbat-congestion.

[9] X. Jiang, G. Jin, CLTCP: an adaptive TCP congestion control algorithm based on congestion level, IEEE Commun. Lett. 19 (8) (August 2015) 1307–1310.

[10] J. Wang, J. Wen, J. Zhang, et al., TCP-FIT: an improved TCP congestion control algorithm and its performance, in: Proc. IEEE INFOCOM, 2011, pp. 2894–2902.

[11] A. Falk, D. Katabi, Y. Pryadkin, Specification for the explicit control protocol (XCP), IETF (July 2007). http://www.isi.edu/isi-xcp/docs/draft-falk-xcp-spec-03.html.

[12] N. Dukkipati, Rate Control Protocol (RCP): Congestion Control to Make Flows Complete Quickly, Stanford University, 2006 Ph.d. thesis. http://yuba.stanford.edu/~nanditad/thesis-NanditaD.pdf.

[13] I.A. Qazi, T. Znati, L.L.H. Andrew, Congestion control with multipacket feedback, IEEE/ACM Trans. Netw. 20 (6) (December 2012) 1721–1733.

[14] A. Afanasyev, N. Tilley, P. Reiher, et al., Host-to-host congestion control for TCP, IEEE Communications Surveys & Tutorials 12 (3) (June 2010) 304–342.

[15] S. Floyd, Highspeed TCP for large congestion windows, IETF (December 2003). http://tools.ietf.org/html/rfc3649.

[16] M.A. Alrshah, M. Othman, B. Ali, et al., Agile-SD: a linux-based TCP congestion control algorithm for supporting high-speed and short-distance networks, J. Netw. Comput. Appl. 55 (Jun. 2015) 181–190.

[17] R. Mittal, J. Sherry, S. Ratnasamy, et al., Recursively cautious congestion control, in: Proc. USENIX NSDI, 2014, pp. 373–385.

[18] L. Tan, C. Yuan, M. Zukerman, A price-based internet congestion control scheme, IEEE Commun. Lett. 12 (4) (April 2008) 331–333.

[19] C. Yuan, L. Tan, L.L.H. Andrew, et al., A generalized FAST TCP scheme, Comput. Commun. 31 (14) (2008) 3242–3249.

[20] F. Ge, S. Chan, L.L.H. Andrew, et al., Performance effects of two-way FAST TCP, Comput. Netw. 55 (2011) 2976–2984.

[21] H. Jung, S. Kim, H.Y. Yeom, et al., Adaptive delay-based congestion control for high bandwidth-delay product networks, in: Proc. IEEE INFOCOM, 2011, pp. 2885–2893.

[22] R. Mittal, V.T. Lam, N. Dukkipati, et al., TIMELY: RTT-based congestion control for the datacenter, in: Proc. ACM SIGCOMM, 2015, pp. 537–550.

[23] R. Jain, A delay based approach for congestion avoidance in interconnected heterogeneous computer networks, in: Proc. ACM SIGCOMM, 1989, pp. 56–71.

[24] K. Tan, J. Song, Q. Zhang, et al., A compound TCP approach for high-speed and long distance networks, in: Proc. of IEEE INFOCOM, 2006, pp. 1–12.

[25] W. Xu, Z. Zhou, D.T. Pham, et al., Hybrid congestion control for high-speed networks, J. Netw. Comput. Appl. 34 (March 2011) 1416–1428.

[26] M. Wang, J. Wang, S. Han, Adaptive congestion control framework and a simple implementation on high bandwidth-delay product networks, Comput. Netw. 64 (March 2014) 308–321.

[27] A. Venkataramani, R. Kokku, M. Dahlin, TCP nice: a mechanism for background transfers, ACM SIGOPS Oper. Syst. Rev. 36 (SI) (2002) 329–343.

[28] A. Kuzmanovic, E. Knightly, TCP-LP: low-priority service via endpoint congestion control, IEEE/ACM Trans. Networking 14 (4) (August 2006) 739–752.

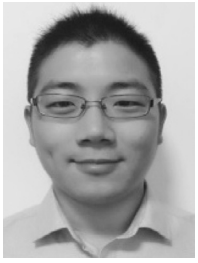
[29] L. Tan, C. Yuan, M. Zukerman, FAST TCP: fairness and queuing issues, IEEE Commun. Lett. 9 (8) (2005) 762–764.

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

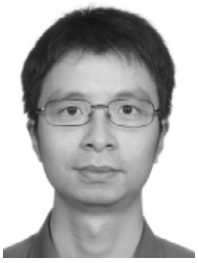
[31] L. Tan, W. Zhang, G. Peng, et al., Stability of TCP/RED systems in AQM routers, IEEE Trans. Autom. Contr. 51 (8) (August 2006) 1393–1398.

[32] L. Tan, W. Zhang, G. Peng, et al., The Network Simulator Version 2, 2012. http://www.isi.edu/nsnam/ns/.

[2] https://iperf.fr/.



Xianliang Jiang received the Ph.D. degree in Computer Science and Technology from the Zhejiang University in 2016. Before that he received the B.E. degree from the University of Science and Technology of China in 2009, and the M.S. degree from the Ningbo University in 2012. He is currently a lecturer and his research interests include protocol design, congestion control and Internet of Things.



Guang Jin received the Ph.D. degree in Computer Science and Technology from the Zhejiang University. Since 2001, he has been with the faculty of Electrical Engineering and Computer Science, Ningbo University. He is currently a full professor and his research interests focus on wireless networking, network protocol and Internet of Things.