Modeling and Solving TCP Incast Problem in Data Center Networks

Jiao Zhang, Fengyuan Ren, *Member, IEEE*, Li Tang, and Chuang Lin, *Senior Member, IEEE*

Abstract—TCP Incast problem attracts much attention due to the catastrophic goodput drop. In this paper, a goodput model of the problem is built to understand why goodput collapse occurs and a solution to the problem based on the theoretical analysis is proposed. We found that the TCP Incast goodput deterioration is mainly caused by two types of timeouts, one happens at the tail of data blocks and dominates the goodput when the number of senders is small, while the other one at the head of data blocks and governs the goodput when the number of senders is large. The proposed model describes the relationship between these two types of timeouts and the Incast communication pattern, block size, bottleneck buffer size, and so on. The simulation results indicate that the model well characterizes the features of the TCP Incast problem. Enlightened by the analysis, a PRiority-based solution to the TCP INcast problem (PRIN) is proposed, which avoids timeouts at the head of blocks by reducing TCP send window and prevents timeouts at the tail of blocks by leveraging priority technology. The experimental results show that PRIN solves the TCP Incast problem.

Index Terms—Data center network, TCP incast, modeling, goodput, experiment

1 Introduction

TCP Incast has risen to be a critical problem recently in data center networks due to its catastrophic goodput collapse [2], [3], [4]. *Incast*, a communication pattern, was first termed by Nagle et al. in file storage systems [5]. In the Incast communication pattern, multiple senders concurrently transmit data blocks to a single receiver, and any sender cannot send another data block until all the senders finish transmitting the current ones. As the number of senders increases, the goodput of the receiver will become lower than the capacity of the bottleneck link in one or even two orders of magnitudes. The Incast communication pattern exists in many popular applications, such as cluster-based storage systems [5], [6], [7], MapReduce-based applications, including web research, digital media processing and so on.

Much attempt has been made to avoid the TCP performance deterioration in the Incast communication pattern [2], [3], [4], [8], [9]. However, few work has been done to understand the radical reasons of the TCP Incast problem theoretically. In this paper, first a goodput model of the TCP Incast problem is built to understand the problem in depth, then enlightened by the goodput model, a simple TCP modification mechanism, named PRIN, is proposed to address the problem.

The main challenges of modeling the TCP Incast goodput are twofold. (1) Most of traditional modeling work on TCP assumes that the application layer always delivers enough data to the transport layer [10], [11], [12], [13], [14]. Thus, the goodput of TCP will not be affected due to insufficient data from the application layer. However, the workflow in our model exhibits the *Incast* communication pattern. All the senders deliver data from the application layer to the transport layer synchronously. The laggard sender will cause that the transport layers of the other ones have no data to transmit. (2) The proposed TCP Incast goodput model describes the overall goodput of the bottleneck link that contains multiple flows instead of focusing on the throughput of only one flow as traditional TCP goodput models do. It is difficult to model the interaction of multiple flows in the Incast communication pattern.

In the proposed TCP Incast goodput model, we summarize that the goodput collapse in the incast communication pattern is mainly caused by two kinds of TimeOuts (TO).

- Block tail TimeOut (BTTO). It is caused by the special Incast communication pattern. Since each sender cannot get the next block data from the application layer until all the senders finish transmitting the current ones, if one of the last three packets (assume three duplicate ACKs are needed to trigger fast retransmission (FR)) in current block of a sender is dropped, then the sender will not receive enough ACKs to trigger FR, a timeout naturally occurs.
- Block head TimeOut (BHTO). BHTO is apt to happen when the number of senders becomes larger. During transmitting a block, some senders finish earlier, and they have to wait for the others to finish without taking any bandwidth. Therefore, the behindhand flows will finish their blocks using more capacity on average, which results that they have larger send window size on average when finishing their current blocks. At the beginning of the next blocks, all the senders inject their whole windows to the small Ethernet buffer, which usually causes lots of dropped packets. If a flow unfortunately losses its whole window, which can easily happen since the

- *J. Zhang is with the School of Information and Communication Engineering, BUPT and State Key Laboratory of Networking and Switching Technology, BUPT, China. E-mail: jiaozhang@bupt.edu.cn.*
- *F. Ren, L. Tang, and C. Lin are with the Department of Computer Science and Technology, Tsinghua University, Beijing, China and the Tsinghua National Laboratory for Information Science and Technology, Beijing, China. E-mail: {renfy, tangli, clin}@csnet1.cs.tsinghua.edu.cn.*

Manuscript received 12 Oct. 2013; accepted 16 Dec. 2013. Date of publication 10 Mar. 2014; date of current version 9 Jan. 2015.
Recommended for acceptance by V. B. Misic.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2014.2310210

1045-9219 © 2014 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

window of each flow becomes smaller as the number of senders increases, then it will enter a TO period.

Enlightened by the theoretical analysis results of the TCP Incast goodput model, we developed a simple mechanism PRIN which modifies TCP a little and avoids the two kinds of timeouts, BHTO and BTTO. The proposed mechanism reduces the send window of each connection at the start of blocks to avoid BHTOs, and prevents BTTOs by configuring higher priority for the last three packets at the end of blocks to prevent them from dropping. The existed priority mechanisms mainly include IEEE 802.1p [15], differentiated services code point [16], type Of services [17], and so on. IEEE 802.1p has long been implemented in most switches [18] and it cooperates with IEEE 802.1q (VLAN). VLAN is widely used in today's enterprise networks [19] and is extended to be virtual eXtensible local area network to adapt to virtualized data center networks recently [20]. Thus, we employ IEEE 802.1p to configure the priority in our implementation of PRIN.

The performance of PRIN is evaluated on a real testbed that is consist of Dell servers, a HP ProCurve 2910al Ethernet Switch and a Cisco Catalyst 2960G Ethernet Gigabit Switch. The experimental results demonstrate that the proposed mechanism, RPIN, avoids almost all the TOs and thus averts the goodput collapse in Incast communication pattern. From the results, we can also infer that the performance deterioration of Incast applications is indeed mainly caused by BHTOs and BTTOs.

The remainder of the paper is organized as follows. Next section introduces related work. In Section 3, the main assumptions and denotations used in our goodput model are listed. Subsequently, the goodput of TCP Incast with and without awnd limitation is modeled and validated in Sections 4 and 5, respectively. In Section 6, the mechanism, PRIN, is proposed to avoid BHTOs and BTTOs, and the implementation of PRIN is described. In Section 7, the performance of the proposed PRIN is evaluated on a real testbed. Finally, the paper is concluded in Section 8.

2 Related Work

The existing approaches to solve the TCP Incast problem can be classified into three categories.

First, avoiding or reducing TOs by modifying TCP. In [2], several trials have been made to avoid TOs, such as reducing the duplicate ACK threshold of entering fast retransmission from 3 to 1, disabling slow start phase, and trying different TCP versions. However, most of these methods are ineffective. Since the serious bandwidth wastage in TCP Incast is caused by the large RTOmin, which typically equals 200 milliseconds in TCP, Vasudevan et al. suggested decreasing RTOmin to microsecond-granularity to reduce the capacity wastage caused by TOs. This method reduces the bandwidth wastage caused by TO periods. However, it does not decrease the number of TO periods. Since a TCP flow will enter slow start phase after a TO period, the frequent slow start periods will possibly reduce the goodput of flows. Besides, small RTOmin is likely to cause spurious retransmission.

The above methods attempt to modify TCP at the sender side. Wu et al. proposed ICTCP [8] which controls flow rate by adaptively adjusting the awnd at the receiver side. The bottleneck link is assumed to directly connect to the receiver. The receiver estimates the available bandwidth and round trip time (RTT) to compute the reasonable awnd and thus each flow fairly injects proper traffic to the network. However, exact estimation of real-time available bandwidth and RTT is challenging. Foremost, ICTCP fails to work well if the bottleneck is not the link that connects to the receiver as stated in [8].

Second, replacing TCP with other new transport protocols. Some recently proposed new transport protocols for data centers can also mitigate the TCP Incast performance deterioration. For example, DCTCP [9] employs explicit congestion notification technology to avoid packet losses and thus reduces the number of TOs. In D^3 [21], the senders of delay-sensitive flows compute their expected send rates and transmit them to switches. Each switch assigns proper rate for each flow based on the collected rate requirements to avoid traffic congestion and thus solve the TCP Incast problem. PDQ [22] provides delay-aware transmission control by emulating preemptive scheduling mechanisms. pFabric [23] is a novel transport protocol for data center networks wherein switches provide priority-based scheduling. Flows start at the line rate and throttle back only under high and persistent packet losses. These new proposed transport protocols can solve the TCP Incast problem. However, before they are widely deployed, it is better to design a light-weight TCP modification mechanism that can be easily deployed and poses no impact on other kinds of applications, to solve the TCP Incast problem.

Third, employing mechanisms at other layers. Several solutions have been proposed at the data link layer. Phanishayee et al. proposed using Ethernet Flow Control to solve the TCP Incast problem [2]. However, it cannot work well if multiple switches exist between the senders and the receiver due to head of blocking. Pan et al. suggest preventing packet losses by modifying quantized congestion notification (QCN) [24], which is an Ethernet layer congestion control mechanism designed for data center ethernet. Simulation results show that QCN proposed by IEEE 802.1 qau group fails to solve TCP Incast problem. Thus, QCN is modified by increasing the sampling frequency at the congestion point and making the link rate increase adaptively to the number of flows at the reaction point. Zhang et al. proposed fair QCN (FQCN) which modifies the congestion feedback in QCN to improve the fairness of different flows along the same single bottleneck. However, FQCN requires that the switch monitors the packet arrival rate of each flow, which incurs high overhead due to the large number of flows in data centers. Besides, modifying QCN incurs overhead for all the other applications except for that with TCP Incast problem in data center networks since QCN works in the link layer. At the application layer, Facebook [25] engineers proposed limiting the number of outstanding requests to alleviate Incast congestion.

3 Assumptions and Notations

3.1 Assumptions

3.1.1 TCP Incast Scenario

The bottleneck buffer employs the Drop Tail queue management scheme. Packets will not be dropped unless the

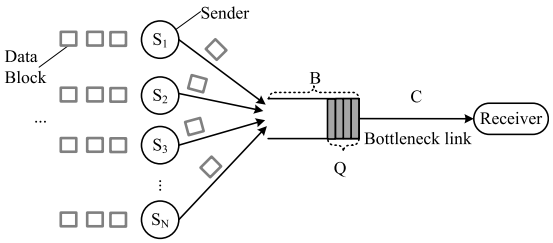

Fig. 1. A scenario of TCP Incast, where multiple senders concurrently transmit data blocks to a single receiver.

bottleneck buffer overflows. Besides, if the number of senders is larger than the bottleneck buffer size in unit of packets, then even if each sender transmits one packet, the bottleneck buffer will be overwhelmed. Therefore, we assume that the number of senders is smaller than the buffer size.

3.1.2 TCP

Assume that the TCP version is NewReno, which is popular in practice. The receiver sends one ACK for each received packet and ACKs are not lost. The threshold of duplicate ACKs to trigger FR phase is 3. Since the unabiding slow start process imposes a negligible impact on TCP througput, it is igonred in our modeling.

3.2 Notations

Before defining the notations, we first introduce a concept called *round*. The first round starts from the beginning of a congestion avoidance (CA) period and lasts one RTT. The after round starts from the end of the last round and also lasts one RTT. A CA period finishes with the next round after some packets being dropped. If the dropped packets are detected by the sender through three duplicate ACKs, then a FR period will be entered, else if through a fired retransmission timer, then a TO period occurs.

A scenario of TCP Incast is shown in Fig. 1. N senders transmit data blocks to a single receiver. The bottleneck bandwidth is C packets per second. The bottleneck buffer size is B packets. Each packet has the same payload S_p Bytes. Considering a CA period, let W_i be the window size of a flow in round i whose duration is R_i. Q_i denotes the queue length of the bottleneck buffer at the end of round i. The other key notations are summarized in Table 1 for the sake of terseness.

TABLE 1
Key Notations in Our Model

Not.	Description
W_m	Window size when some of the N flows begin to drop packets
W_n	Expected maximum window size
W_l	Advertised window size of the receiver
D	Propagation delay between each sender and the receiver
T_N^C	Expected duration of a CA period with total N flows
Y_N^C	Expected number of packets successfully transmitted in a CA period
N_m	The number of flows which lost packets when window size is W_m
Y^B	The block size in unit of packets
Y_N^F	Expected number of successfully sent packets in a CA + FR period
T_N^F	Expected duration of a CA + FR period
N^*	Critical point between BHTO dominating goodput and BTTO doing
G	Goodput of the receiver without advertised window limitation
G_l	Goodput with window limitation

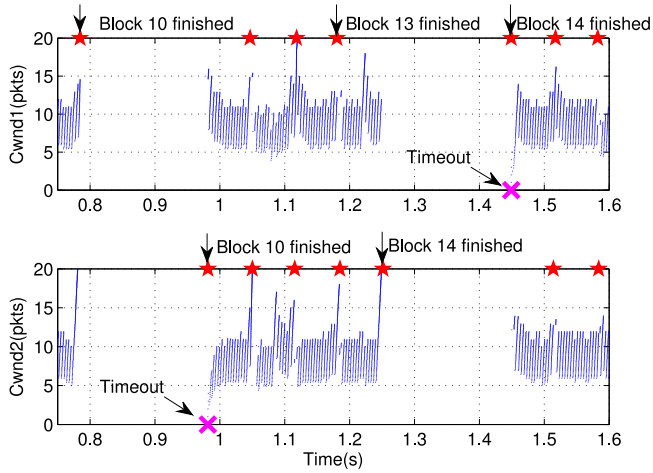

Fig. 2. The scenario where BTTO happens. $N = 8$ senders concurrently transmit packets to the same receiver. The packet size $S_p = 1$ KB, bottleneck bandwidth $C = 1$ Gbps = 12.5 pkts, buffer $B = 64$ packets, synchronized block $S_b = 1,024$ KB. The advertised window of the receiver is set to $1,000$ packets. We can see that as long as one flow enters a TO period at the end of a block, the other flow will also undergo a TO period.

4 Goodput Model without awnd Limitation

The goodput of TCP NewReno [26] in the Incast environment without awnd limitation will be modeled in this section. As aforementioned in Section 1, two types of TOs lead to TCP goodput drop. We will first show them in Figs. 2 and 3 which are plotted based on the results of simulations conducted on the ns-2 platform.

Fig. 2 shows the scenario where BTTO happens. Eight senders transmit synchronized data blocks to the same receiver. The figure plots the window evolution of two senders among them. A pentagram plotted at $(t, 20)$ represents

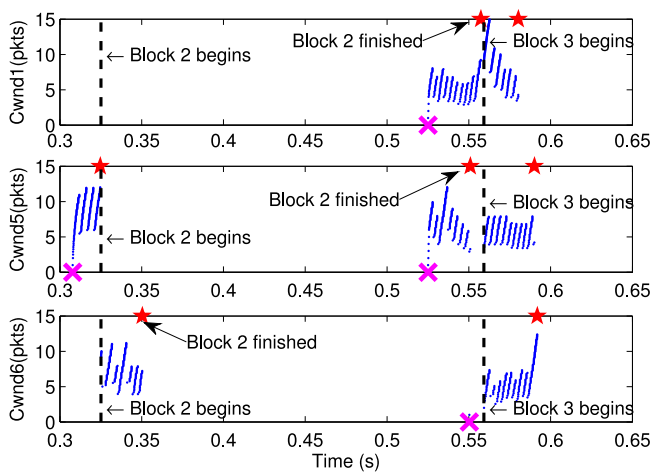

Fig. 3. The scenario where BHTO happens. The main parameters of this scenario are: $N = 32, S_p = 1KB, B = 64, S_b = 256\,KB, C = 12.5\,pkts$. The window evolutions of sender $1, 5, 6$ are plotted to illustrate BHTO.

that a block finishes at time t. The big X represents that a retransmission timer is fired. The *awnd* of the receiver is set to $1{,}000$ packets, which is large enough that it has no impact on the send window evolution. We can see that at about time $t = 0.79$ s, sender 1 finishes Block 10 and then the window size does not vary. While sender 2 suffers a TO period before finishing Block 10 during time about $(0.79 \sim 0.98)$ s. By observing the congestion window evolution of flow 2, we found that the penultimate packet of Block 10 of sender 2 was dropped. Although the last packet of Block 10 was successfully transmitted, only one duplicate ACK was received by sender 2. What's more, according to the Incast communication pattern, the packets of Block 11 will not be delivered to the transport layer from the application layer until all senders finish Block 10. Therefore, sender 2 has to wait until the retransmission timer fires at about 0.98 s. A timeout event happens. Then sender 2 retransmits the dropped packet, and then Block 10 is finished. With respect to sender 1, although it finishes Block 10 much earlier than sender 2, it cannot get the data of Block 11 immediately since sender 2 does not finish Block 10 yet. Hence, it also waits until sender 2 finishes Block 10. The bandwidth is wasted during $0.79 \sim 0.98$ s, which deteriorates the goodput. While at about 1.25 s, sender 1 delays finishing Block 14 due to the same kind of timeout and therefore also degrades goodput.

Fig. 3 illustrates the situation where BHTO happens. $N = 32$ flows concurrently send packets to the same receiver. The advertised window size of the receiver is also set to $1{,}000$ packets. We plot three of the 32 flows to illustrate the behavior of BHTO. The pentagram represents that a block is finished. At time 0.325 s, all the senders begin to transmit Block 2. We can see that sender 6 finishes Block 2 at about 0.35s. Unfortunately, sender 1 and 5 do not receive any ACKs and thus their windows do not change. Through tracking the simulation data, we can find that both sender 1 and 5 lose all packets sent in their first windows at the beginning of Block 2 and thus no new and duplicate ACKs are fed back to the sender. Therefore, their retransmission timers fire at about time 0.52 s. And at 0.56 s, both of them finish Block 2. It can be inferred that all the other flows also finish Block 2 before 0.56 s since the transmission of Block 3 begins at this moment. With respect to Block 3, all the three senders luckily finish their third Block soon without undergoing TO periods. However, they do not continue to transmit Block 4 at once, which implies that some other flows delay finishing their third Blocks.

Through investigating numerous simulation data, we found that BTTO dominates TCP goodput when N is small while BHTO does when N is large. Let $N^*(1 \le N^* \le N)$ denote the critical value between these two situations. We will first model the goodput when N is small where BTTO is the main factor of degrading TCP performance. Subsequently, N^* will be computed and the goodput when N is large where BHTO largely deteriorates TCP goodput will be modeled.

4.1 Goodput As $N < N^*$

4.1.1 Dynamics of Queue Length

During the ith round of a CA period, N senders transmit data to one receiver. Then NW_i packets will be injected into the bottleneck buffer, and CR_i packets will be served by the bottleneck link. Thus, we can obtain Q_i, the number of packets in the queue at the end of the ith round, as follows:

$$Q_i = \min\{(Q_{i-1} + N \times W_i - C \times R_i)^+, B\}, \tag{1}$$

$(\cdot)^+$ equals $(\cdot)$ if $(\cdot) > 0$, else equals 0.

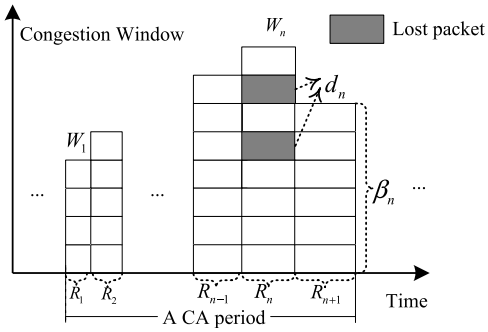

Fig. 4. Congestion window evolution during a CA period.

4.1.2 Relationship between RTT and Queue Length

Assume that the queue employs First-Come-First-Served model and the propagation delay between each source to the destination is a constant value D. As a rough approximation, R_i is the sum of the propagation delay and the queuing delay as follows:

$$R_i = \mathbb{E}\left(D + \frac{Q_{i-1} + \phi}{C}\right), \tag{2}$$

where ϕ is a stochastic variable which models the possible longer queuing delay of the transmitted packets than $\frac{Q_{i-1}}{C}$ in round i. Clearly, the first packet transmitted in round i will undergo $\frac{Q_{i-1}}{C}$ queuing delay since the queue length at the end of round $(i-1)$ is Q_{i-1}. However, the afterwards packets will suffer longer queuing delay if the rate of the arrival traffic is larger than the departure rate.

4.1.3 Number of Packets Successfully Transmitted in a CA Period

Fig. 4 illustrates the congestion window evolution in a CA period. The window size increases by 1 in each round $i(i > 1)$ until some packets are dropped at round n. Rounds $(1 \sim n+1)$ form a CA period. Let W_n be the maximum congestion window size in a CA period. d_n packets will be lost when the window size becomes W_n. In the last round, β_n packets will be transmitted. Therefore, the number of successfully transmitted packets Y_n in a CA period is

$$Y_n \quad = S_n + \beta_n - d_n, \tag{3}$$

where $S_n = \sum_{j=0}^{\frac{W_n}{2}} (\frac{W_n}{2} + j) = \frac{3}{8}(W_n)^2 + \frac{3}{4}W_n$.

Now compute the maximum window size W_n. According to Eqs. (1) and (2), we can infer

$$Q_i = \min\{(NW_i - CD - \phi)^+, B\}. \tag{4}$$

In a CA phase, the difference between Q_i and Q_{i-1} is about N according to Eq. (4). The first packet in round i suffers $\frac{Q_{i-1}}{C}$ delay, and the last packet in round i suffers $\frac{Q_i}{C} = \frac{Q_{i-1}+N}{C}$ delay. Since the arrival rate and departure rate

are both constant, we can infer that $\mathbb{E}(\phi) = \frac{N}{2}$. If $Q_i > B$, i.e., $W_i > \frac{CD+B+\frac{N}{2}}{N}$, some packets will be dropped. Let

$$W_m = \left\lfloor \frac{CD+B}{N} + \frac{1}{2} \right\rfloor + 1. \tag{5}$$

In the mth round, approximately $N_m = NW_m - \lfloor CD + \frac{N}{2} + B \rfloor$ packets will be dropped. Obviously, $1 \le N_m \le N$. Since the windows evolutions of all the flows are synchronized when N is small, their packets will be fairly dropped. So we can infer that about N_m flows will lose one packet each. While the window of the other $(N - N_m)$ flows will increase to $W_m + 1$. Assume that each of the $(N - N_m)$ flows will lose one packet when their window sizes are $W_m + 1$, we can get the maximum window size in a CA period

$$W_n = \begin{cases} W_m, & \text{with probability } \frac{N_m}{N}, \\ W_m + 1, & \text{with probability } (1 - \frac{N_m}{N}). \end{cases} \tag{6}$$

Hence, the expected number of packets successfully transmitted by one of N flows in a CA period, Y_N^C, is

$$Y_N^C = \left\lfloor \frac{N_m}{N} Y_m + \left(1 - \frac{N_m}{N}\right) Y_{m+1} \right\rfloor. \tag{7}$$

Assume β_i uniformly distributes between 1 and W_i, then its expectation $\mathbb{E}(\beta_i) = \frac{W_i}{2}$. Based on the analysis above, we have $d_m = d_{m+1} = 1$. Hence, from Eqs. (3) and (7), we can obtain

$$\begin{aligned} Y_N^C &= \mathbb{E}\left(\left\lfloor \frac{N_m}{N}(S_m + \beta_m - d_m) \right. \right. \\ &\quad \left. \left. + \left(1 - \frac{N_m}{N}\right)(S_{m+1} + \beta_{m+1} - d_{m+1}) \right\rfloor \right) \\ &= \left\lfloor \frac{3}{8}(W_m)^2 + 2W_m + \frac{5}{8} - \left(\frac{3}{4}W_m + \frac{13}{8}\right)\frac{N_m}{N} \right\rfloor. \end{aligned} \tag{8}$$

4.1.4 Duration of a CA Period

Assume $Q_{i-1} > 0 (1 \le i \le n)$, namely, there are backlog in the buffer during CA phases. Combining Eqs. (2) and (4), we have $R_i = \frac{NW_{i-1}}{C}$. Since $W_1 = \frac{W_n}{2}$, according to the window regulation law in slow start phases, the window size W_0 before the first round should be $\frac{W_n}{4}$. Thus, the duration T_n of a CA period with the maximum window size W_n is as follows:

$$T_n = \sum_{i=1}^{n+1} R_i = \frac{N}{C}\left(\frac{3}{8}(W_n)^2 + W_n\right). \tag{9}$$

Similar to the analysis of computing the number of the successfully transmitted packets Y_N^C in a CA period, the expectation of the duration T_N^C of a CA period is

$$\begin{aligned} T_N^C &= \frac{N_m}{N} T_m + \left(1 - \frac{N_m}{N}\right) T_{m+1} \\ &= \frac{N}{C}\left(\frac{3}{8}(W_m)^2 + \frac{7}{4}W_m + \frac{11}{8} - \left(\frac{3}{4}W_m + \frac{11}{8}\right)\frac{N_m}{N}\right). \end{aligned} \tag{10}$$

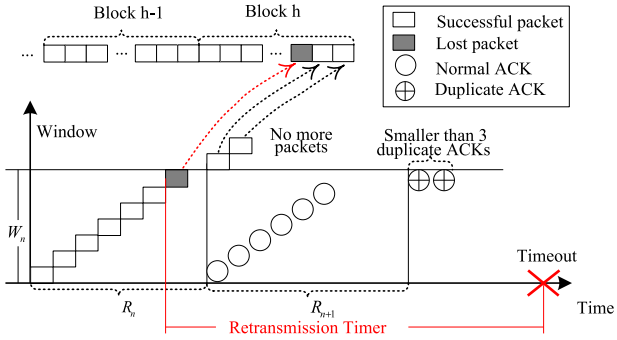

Fig. 5. The scenario where a BTTO happens.

4.1.5 Probability of Block Tail Timeout

Fig. 5 illustrates a timeout event which happens at the tail of blocks. As long as any one of the last three packets in a block is lost, a timeout event will appear due to inadequate ACKs to trigger FR. As shown in Fig. 5, the third packet from the end of Block h is lost, then even if the last two packets of Block h are successfully transmitted, the sender can receive only two duplicate ACKs, which insufficiently triggers a FR procedure. Then, when the retransmission timer fires, a TO event occurs. We refer to this type of TO as BTTO. Next, the probability of this event occurrence is deduced.

The number of packets successfully transmitted by a flow during a CA period is Y_N^C. The number of packets that a block contains, denoted by Y^B, is $Y^B = \lceil \frac{S_b}{S_p} \rceil$, where S_b, S_p are the sizes of a block and a packet, respectively. If a TO period appears after k CA periods, then at least one lost packet in the kth CA period is one of the last three packets in a block, namely

$$kY_N^C - \alpha = hY^B - \beta, \ k \text{ and } h \text{ are integers}, \tag{11}$$

where α, a stochastic variable, is the number of packets successfully transmitted after a lost packet in the kth CA period. If a packet is dropped in round n, then $\alpha = W_n - 1$. Based on the model of the maximum window size W_n defined in Eq. (6), we have

$$\alpha = \begin{cases} W_m - 1, & \text{with probability } \frac{N_m}{N}, \\ W_{m+1} - 1 = W_m, & \text{with probability } \left(1 - \frac{N_m}{N}\right). \end{cases} \tag{12}$$

If $(kY_N^C - \alpha)$ is just one of the last three packets in the hth Block, then a TO period will appear. $hY^B - \beta (\beta \in \{0, 1, 2\})$ models one of the last three packet in the hth Block.

Denote $y(x) = \min\{k | kY_N^C - hY^B = x\}$. For a specific α value and a flow f, the number of CA periods between two successive TO periods is

$$k_\alpha^f = \min\{y(\alpha), y(\alpha - 1), y(\alpha - 2)\}. \tag{13}$$

Now consider N flows. If at least one of the N flows enters a BTTO period, the other flows will also wait for a period which almost equals the BTTO period even if they have successfully transmitted Block h (Fig. 2). This is because they cannot get the next block data from the application layer. Therefore, the probability of the BTTO period of a flow is the maximum probability of BTTO of the N flows.

Let k_{min} be the number of CA periods between two successive TO periods when there are total N flows. Define

$$\alpha_1 = \begin{cases} W_m - 1, & \text{if } k^f_{W_m-1} < k^f_{W_m}, \\ W_m, & \text{else}, \end{cases} \tag{14}$$

and $\{\alpha_2\} = \{W_m - 1, W_m\} - \{\alpha_1\}$, then we have

$$k_{min} = \begin{cases} k^f_{\alpha_1}, & \text{with probability } 1 - (Pr[\alpha = \alpha_2])^N, \\ k^f_{\alpha_2}, & \text{with probability } (Pr[\alpha = \alpha_2])^N. \end{cases} \tag{15}$$

Thus, the expectation of k_{min} is

$$\mathbb{E}(k_{min}) = k^f_{\alpha_1}(1 - Pr[\alpha = \alpha_2])^N + k^f_{\alpha_2} Pr[\alpha = \alpha_2]^N. \tag{16}$$

Hence, the probability of entering a TO period from a CA period is $P^O = \frac{1}{\mathbb{E}(k_{min})}$.

Now compute the duration of a TO period T^O. A TO period possibly contains several timeouts and ends with a successfully retransmitted packet. In our model, the window evolution of all the flows can be assumed to be synchronized when N is small, and the packets will only be dropped when the bottleneck buffer overflows, so the retransmitted packet after the first timeout in a TO period will be successfully transmitted if $N < B$ since the windows of all the flows start from 1 after a timeout. Let T_0 denote the duration of the first timeout duration, which usually equals `RTOmin` since the RTT of data center networks is quite small. The duration of a TO period equals the first timeout duration, namely, $T^O = T_0$.

4.1.6 Goodput

When $N < N^*$, at the end of each CA period, a TO period happens with the probability P^O. Thus, TCP goodput G_1 as $N < N^*$ is calculated as follows:

$$\begin{aligned} G_1 &= N \frac{Y^C_N}{T^C_N + P^O T^O} S_p \\ &= \frac{N S_p \lfloor \frac{3}{8} W_m^2 + 2W_m + \frac{5}{8} - (\frac{3}{4} W_m + \frac{13}{8}) \frac{N_m}{N} \rfloor}{\frac{N}{C}(\frac{3}{8} W_m^2 + \frac{7}{4} W_m + \frac{11}{8} - (\frac{3}{4} W_m + \frac{11}{8}) \frac{N_m}{N}) + P^O T_0}. \end{aligned} \tag{17}$$

Remarks. If there are only CA periods, then according to the expression of Y^C_N and T^C_N, TCP goodput of all the N senders approximately equals C no matter what values N, B, S_b take. Therefore, the probability of TO periods, P^O, is the main factor of degrading goodput. P^O is decided by whether the lost packet is one of the last three packets in a block, which has negative correlation to the Least Common Multiple of Y^B and Y^C_N, i.e., LCM(Y^B, Y^C_N). Thereby, larger block size Y^B decreases P^O and thus improves goodput G_1.

4.2 Goodput As $N > N^*$

4.2.1 Calculation of N^*

By observing simulation results, we found that when N becomes relatively large, most TO periods happen at the beginning of blocks. While few TO periods happen in the subsequent rounds. This is because some flows will have larger window sizes at the end of current block if the other flows finish their blocks earlier. Besides, each flow injects all the packets into networks according to its window size at a quite short interval at the start of the next block. If these packets cannot be accommodated by the buffer, some of them will be dropped. A unlucky flow, which unfortunately loses its whole window, will enter a TO period. While in the subsequent rounds, the congestion windows are regulated by CA procedures. Only one packet is transmitted after an ACK is received. Hence, there are less traffic burst. All the flows lose packet more fairly than that at the beginning of a data block transmission. And the flows will timely respond to packet droppings and thus few TO events appear due to full window losses. We refer to the TO periods that occur at the start of blocks as Block head TimeOut.

In reality, the numbers of the two types of TO periods, i.e., BTTO and BHTO, per data block vary with different number of senders N. Next the critical point N^* is computed. If N^* senders transmit data to the same client, then on average one flow will suffer a BHTO period in each block. When $N < N^*$, BTTOs are dominative, while when $N \geq N^*$, BHTOs are significant.

To determine whether a flow enters a TO period at the beginning of Block b, we first compute the number of lost packets D_b at the first round of its transmission. Let A_b be the expected summation of the first windows of all the flows at the start of Block b. If all the windows of the N flows vary synchronously, then the windows of all the flows uniformly distribute between $\frac{W_n}{2}$ and W_n. While the system can accommodate about NW_n packets. Hence, in theory few packets will be lost at the start of Block b. However, simulation results tell that when N becomes large, the asynchronism of the windows evolutions cannot be ignored. As stated in Section 4.1.3, N_m flows lose packets when their congestion window sizes equal to W_m while the window of the other $(N - N_m)$ flows continue to increase. Hence, approximate $(N - N_m)$ flows will finish transmitting Block $(b-1)$ earlier than the other N_m flows. Then the N_m flows will compete for the bandwidth of the bottleneck link to transmit their remaining $(b-1)$th Blocks. Therefore, at the start of Block b, the windows of the $(N - N_m)$ flows take values between $\frac{W_m}{2}$ and W_m, while the windows of the other N_m flows uniformly distribute between $\frac{W_m^{N_m}}{2}$ and $W_m^{N_m}$. Let

$$W_m^{N_m} = \begin{cases} \lfloor \frac{CD+B}{N_m} + \frac{1}{2} \rfloor + 1, & N_m > 0, \\ W_m, & N_m = 0. \end{cases} \tag{18}$$

Hence, we can obtain that

$$A_b = (N - N_m) \times \frac{3}{4} W_m + N_m \times \frac{3}{4} W_m^{N_m}. \tag{19}$$

The number of dropped packets D_b at the beginning of Block b is the difference between the arrival and the summation of departure plus the backlog in the buffer, namely

$$D_b = A_b - (\tau C + B), 0 < \eta < 1, \tag{20}$$

where τ is the maximum time spent by the senders injecting the packets in their first windows. Since $W_m^{N_m} \geq W_m$, and the capacity of the link between each sender and the intermediate switch is C packets, the maximum time taken by the senders is $\tau = \frac{W_m^{N_m}}{C}$.

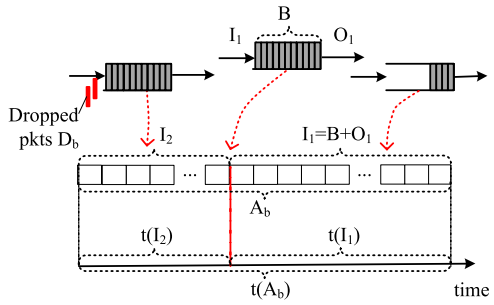

Fig. 6. The behavior of the packets in the first windows of all the flows at the beginning of Block b.

Next let us compute the number of flows suffering a TO at the start of Block b. Assume that the arrival rate to the bottleneck buffer, denoted by r, is constant. As shown in Fig. 6, if all the flows totally spend time $t(A_b)$[1] injecting their first windows to the network, then during time $t(I_1)(t(I_1) < t(A_b))$, no packets will be dropped since $I_1 = B + O_1$, where O_1 is the number of packets served by the bottleneck link during $t(I_1)$. However, some packets could be lost if the arrival rate is larger than the capacity of the link during $t(I_2)$. The probability P_t that a flow starts during $t(I_2)$ is

$$P_t = \frac{t(I_2)}{t(A_b)} = \frac{rt(I_2)}{rt(A_b)} = \frac{I_2}{A_b}.$$

Therefore, the expected number of flows N_t which starts during $t(I_2)$ is

$$N_t = N \times P_t = N \times \frac{I_2}{A_b}. \tag{21}$$

All the lost packets D_b are dropped during $t(I_2)$. Thus, the packet loss probability P_p is $P_p = \frac{D_b}{I_2}$. Then the probability P_w that all the packets of a window W are dropped is

$$P_w = (P_p)^W = \left(\frac{D_b}{I_2}\right)^W. \tag{22}$$

Since the link capacity is constant, the number of departure packets during $t(I_1)$ is $O_1 = \tau C \times \frac{I_1}{A_b}$. Therefore, we can obtain that $I_1 = B + \tau C \times \frac{I_1}{A_b}$. Clearly $A_b = I_1 + I_2$, Therefore,

$$I_2 = A_b - \frac{BA_b}{A_b - \tau C}. \tag{23}$$

Combining Eqs. (21)-(23), we can obtain that the expected number of flows entering TO period after the first round is

$$N^O = N_t \times (P_p)^{\overline{W}} = N\left(1 - \frac{B}{A_b - \tau C}\right) \times \left(\frac{D_b}{A_b - \frac{BA_b}{A_b - \tau C}}\right)^{\overline{W}},$$

here $\overline{W} = (1 - \frac{N_m}{N}) \times \frac{3}{4} W_m + \frac{N_m}{N} \times \frac{3}{4} W_m^{N_m}$ is the expected first window size of each flow.

The minimum N, which enables $N^O = 1$, is N^*.

1. $t(x)$ represents the time taken by transmitting x packets, i.e., $t(x) = \frac{x}{r}$.

4.2.2 Goodput

When $N = N^*$, on average one flow will enter a TO period at the beginning of each block. And when $N > N^*$, we can infer that on average $(N^* - 1)$ lucky flows can transmit a block without undergoing any TOs, and the other $(N - N^* + 1)$ flows will enter a TO period. Assume that the $(N^* - 1)$ lucky flows can finish their blocks during this TO period. Then, after the TO period, the other $(N - N^* + 1)$ flows compete for the bandwidth to transmit packets. All of their windows start from 1, and they transmit a packet only after receiving an ACK. They can relatively fairly use the bandwidth of the bottleneck link without full window losses. But their maximum window sizes are very small when N is large, the FR periods are so frequent that the corresponding time cannot be ignored.

Let TF denote a CA period plus the subsequent FR period. Similar to the analysis as $N < N^*$, we need to compute the number of packets successfully transmitted in a TF period by one of the $(N - N^* + 1)$ unlucky flows, $Y^F_{N-N^*+1}$, and the duration of a TF period, $T^F_{N-N^*+1}$. NewReno will enter into FR after receiving three duplicate ACKs. If the current window is W, d packets are dropped, then the congestion window is $(\frac{W}{2} + W - d)$ since each duplicate ACK increases the window by 1 [11]. If $d < \frac{W}{2}$, then $(\frac{W}{2} - d)$ packets will be transmitted. According to the analysis of computing Y^C_N in Section 4.1.3, when the window size reaches W_n, a flow will drop one packet, i.e., $d = 1$. Hence, in FR, a flow will send $(\frac{W_n}{2} - 1)$ packets. In our model, we only consider $N \leq B$, which implies that $\frac{W_n}{2} > 1$. Assume the packets in the first cycle of a FR period, which lasts about D, are successfully sent, then we can get

$$Y^F_{(N-N^*+1)} = Y^C_{(N-N^*+1)} + \frac{W_n}{2} - d \tag{24}$$

$$T^F_{(N-N^*+1)} = T^C_{(N-N^*+1)} + D. \tag{25}$$

The time that one of the $(N - N^* + 1)$ unlucky flows spends transmitting a block determines when the next block can be transmitted. Since the $(N^* - 1)$ flows can finish their blocks in a TO period and the other flows will not undergo more TOs after the first round, the time that a unlucky flow needs to finish one block is

$$T^B = T_0 + \frac{Y^B}{Y^F_{(N-N^*+1)}} T^F_{(N-N^*+1)}. \tag{26}$$

Therefore, we can get the goodput G_2 as $N \geq N^*$

$$G_2 = N \times \frac{Y^B}{T^B} \times S_p = \frac{NS_p Y^B Y^F_{(N-N^*+1)}}{T_0 Y^F_{(N-N^*+1)} + Y^B T^F_{(N-N^*+1)}}. \tag{27}$$

Combining Eqs. (17) and (27), we can obtain that the TCP goodput of N senders concurrently transmitting data blocks to a receiver is

$$G = \begin{cases} \frac{NS_p \left\lfloor \frac{3}{8}W_m^2 + 2W_m + \frac{5}{8} - (\frac{3}{4}W_m + \frac{13}{8})\frac{N_m}{N} \right\rfloor}{\frac{N}{C}\left(\frac{3}{8}W_m^2 + \frac{7}{4}W_m + \frac{11}{8} - (\frac{3}{4}W_m + \frac{11}{8})\frac{N_m}{N}\right) + P^O T_0}, & N < N^*, \\ \frac{NS_p Y^B Y^F_{(N-N^*+1)}}{T_0 Y^F_{(N-N^*+1)} + Y^B T^F_{(N-N^*+1)}}, & N \geq N^*. \end{cases}$$

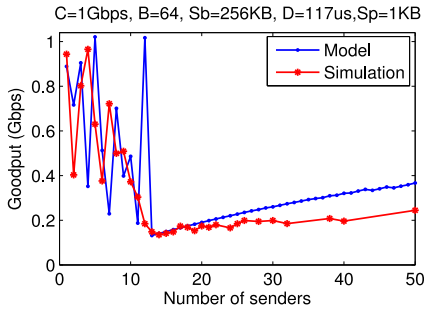

Fig. 7. Normalized goodput with 64 KB buffer.

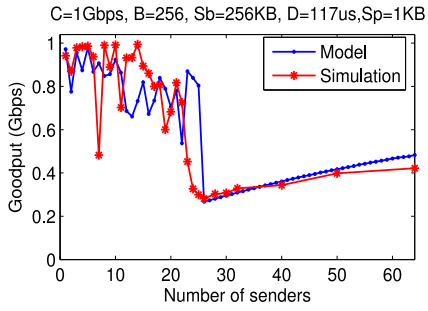

Fig. 9. Normalized goodput with 256 KB buffer.

4.3 Validation

In this section, we validate our model through simulations on the ns-2 platform in scenarios with different buffer size and different data block size, and discuss the impact of some parameters upon TCP Incast goodput. The module for TCP Incast is developed by Phanishayee et.al. in [2]. The `RTOmin` is set to 0.2s.

4.3.1 *Different Buffer Size B*

Figs. 7, 8, and 9 show the normalized goodput of our proposed model and simulation results with different buffer size B. The title of the graph indicates the bottleneck link capacity C, the bottleneck buffer size B, the synchronized data block S_b, the propagation delay D, and the packet size S_p.

The model well characterizes the general goodput tendency of TCP Incast, which indicates that the two types of TOs, BTTO and BHTO, indeed are the essential causes of the TCP Incast problem. Besides, we found that the critical point N^* is just the goodput collapse point. Specifically, when $N < N^*$, i.e., before goodput collapse, some model results are not in conformity with the simulation data. The reason is that the frequency of BTTO is quite sensitive to the location of the lost packet since the lost packet must be one of the last three packets in a block. Therefore, the imprecise number of packets successfully transmitted in a CA period Y_N^C and locations of lost packets will both have negative impact on the accuracy of the model. When $N > N^*$, the model results are almost the same as the simulation data with different buffer size.

From the three simulation curves in Figs. 7, 8, and 9, we can summarize three features. (1) Larger buffer size B improves the whole goodput with different N. This fact can be explained by our proposed model. Larger buffer size B augments the maximum window size $W_m = \lfloor \frac{CD+B}{N} + \frac{1}{2} \rfloor + 1$.

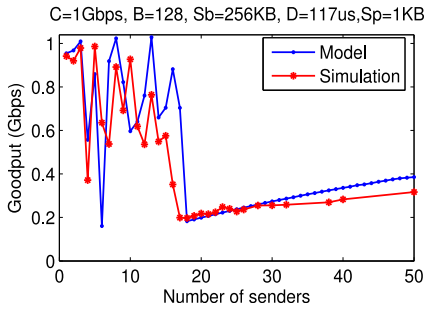

Fig. 8. Normalized goodput with 128 KB buffer.

Then the expected number of packets successfully transmitted in a CA period, Y_N^C, increases. When $N < N^*$, the probability of a TO period has a negative correlation with $\text{LCM}(Y_N^C, Y^B)$, hence the goodput has an ascendant trend as B increases. When $N > N^*$, large maximum window size decreases the time taken by one of the $(N - N^* + 1)$ unlucky flows transmitting a block, namely, $T^B = T_0 + \frac{Y^B}{Y^F_{N-N^*+1}} T^F_{N-N^*+1}$ becomes small. Thus, G_2 increases. (2) Large B makes the critical point N^* shift right. Before goodput collapse, that is, when the number of senders N is smaller than the critical point N^*, the goodput largely depends on BTTO, while as $N > N^*$, goodput is mainly determined by the frequency of BHTO, which will severely decrease TCP goodput. Larger buffer can make N^* shift right since it can cache more packets. Therefore, larger B delays the onset of goodput collapse. (3) After the critical point N^*, goodput becomes larger as N increases. The goodput of a flow is quite low when N equals N^*. However, the goodput slowly increases as N becomes larger. This can be explained using our model. Transmitting a block spends time $T^B = T_0 + \frac{Y^B}{Y^F_{N-N^*+1}} T^F_{N-N^*+1}$. In our analytical model, a TO period lasts only one timeout, namely, its duration equals T_0. Hence, when N becomes large, although the unlucky flows spend more time transmitting their blocks, the time taken by the TO period keeps unchangeable. Thus, larger N increases T^B a little. But the increment is quite small compared with T_0. Hence, goodput slowly increases with larger number of senders N. In fact, when N becomes very large, packets will likely be lost in the TO periods due to more severe bandwidth contention, that is, a TO period will possibly take longer time than T_0. In our model, we do not take this longer TO into consideration, so the analytical results slightly deviate from the simulation data when N becomes quite large. The gap between the model results and the simulation data is more obvious when the buffer size is small, this is maybe because the probability that longer timeout periods happen is larger when the buffer size is small.

4.3.2 *Different Synchronized Data Block S_b*

Figs. 10, 11, and 12 plot the proposed model and simulation results with different block size S_b. We can see that the goodput becomes larger when the block size increases. But large block size has little impact on the onset of goodput collapse. According to our model, block size S_b is irrelative to the maximum window size. Therefore, the goodput of a CA period does not vary. When

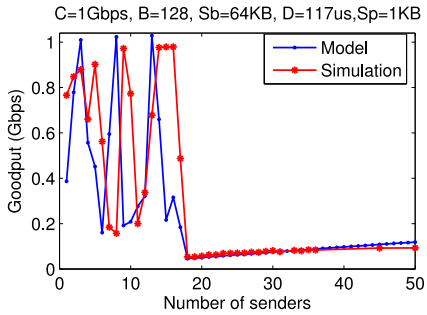

Fig. 10. Normalized goodput with 64 KB block.

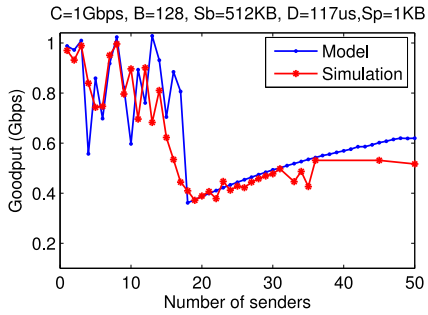

Fig. 12. Normalized goodput with 512KB block.

$N < N^*$, the probability of a TO period has a negative correlation with LCM(Y_N^C, Y^B). Hence, when S_b becomes large, the probability that a TO period happens will have a decline tendency and consequently the goodput will increase. When $N > N^*$, since on average one TO happens in each block. Thereby, when the block size becomes large, the ratio of the time wasted by a TO period to the time spent by unlucky flows transmitting packets becomes smaller. As a result, the goodput increases.

4.3.3 Parameter Analysis

N^* is a quite important point since it is the critical point of the goodput collapse. We conducted a series of simulations with different N, B, C, and obtained the value of N^* using our analytical model. The results are presented in Figs. 13 and 14. We can see that the critical point is mainly related to B, while the bandwidth of the bottleneck link has little impact on it. This is because the window of a flow becomes larger as C increases with a specific N, the number of served packets also increases during the first round of a block. These two impacts just counteract and thus larger C does not enlarge the probability of BHTO. Therefore, larger C does not delay the onset of goodput collapse. With respect to the buffer size, larger buffer size can temporarily accommodate more packets to prevent them from being dropped. Hence, larger B can delay the onset of goodput collapse. However, larger buffer size will increase the end-to-end delay and is not economical. Thus, the commodity switches in data centers usually employ shallow buffer.

5 Goodput with awnd Limitation

The model above is based on the premise that the window limitation of the receiver is so large that its impact can be neglected. In this section, the window limitation W_l will be taken into account.

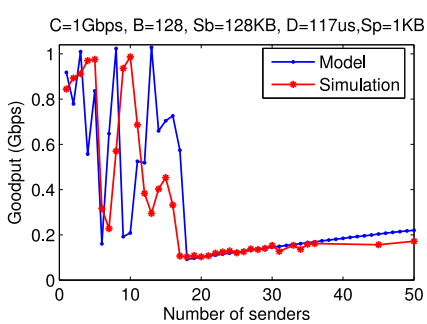

Fig. 11. Normalized goodput with 128KB block.

5.1 Model

5.1.1 $W_l < W_m$

All the windows stop increasing after reaching W_l if $W_l < W_m$. Based on our analysis, no packets will be dropped. Therefore, all the flows keep transmitting data at the rate of $\frac{W_l}{R_l}$. Since they are totally synchronous, all the flows will finish transmitting a block almost at the same time. No senders need to wait for other sluggish senders. Thus, the goodput when window limitation W_l is smaller than W_m is

$$G_l = NS_p \frac{W_l}{R_l}. \tag{28}$$

In Eq. (4), the dynamics of the queue system is modeled as $Q_i = \min\{(NW_i - CD - \phi)^+, B\}$. The window limitation $W_l < W_m$ indicates that $Q_i < B$. Thus $Q_i = (NW_i - CD - \phi)^+$. Since ϕ describes the difference between Q_{i-1} and Q_i, while $W_{i-1} = W_i = W_l$, hence $Q_{i-1} = Q_i$ and further $\phi = 0$. Thus, $Q_i = (NW_i - CD)^+$. From Eq. (2), we can get that $R_l = \frac{NW_l}{C}$ if $NW_l - CD > 0$, else $R_l = D$. Finally, we can obtain the goodput when $W_l < W_m$ as follows:

$$G_l = \begin{cases} \frac{NS_pW_l}{D}, & W_l \le \frac{CD}{N}, \\ CS_p, & \frac{CD}{N} < W_l < W_m. \end{cases} \tag{29}$$

5.1.2 $W_l = W_m$

When the windows of N flows increase to W_m, the windows of N_m flows will drop to $\frac{W_m}{2}$ due to one lost packet based on the analysis in Section 4.1.3, and the other $(N - N_m)$ flows will keep W_m until the windows of the N_m flows increase to W_m again. Then another N_m flows will drop to $\frac{W_m}{2}$. Consequently, in a CA period, the expected number of successfully transmitted packets is

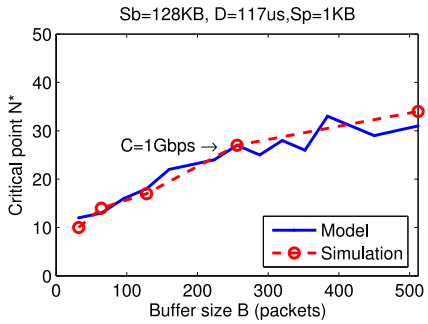

Fig. 13. Critical point N^* with different B and $C = 1$ Gbps.

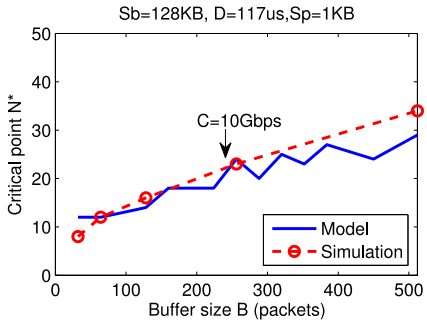

Fig. 14. Critical point N^* with different B and $C = 10$ Gbps.

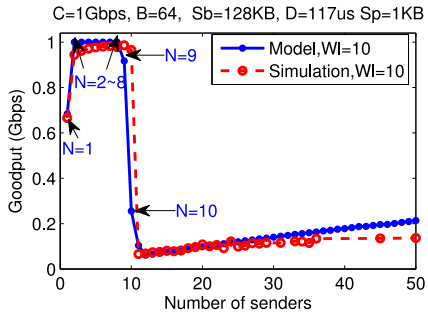

Fig. 15. Normalized goodput with window limitation $W_l = 10$.

$$\hat{Y}^C = \frac{N_m}{N} Y_m + \left(1 - \frac{N_m}{N}\right) W_m T_m, \tag{30}$$

where Y_m and T_m is defined in Eqs. (3) and (9), respectively.

The expected duration of a CA period is $\hat{T}^C = T_m$. Thus, the goodput with window limitation $(W_l = W_m)$ is

$$G_l = \frac{NS_p\hat{Y}^C}{\hat{T}^C}, W_l = W_m. \tag{31}$$

5.1.3 $W_l \geq W_{m+1}$

When W_l is larger than W_m, the goodput of TCP Incast will not be affected by the advertised window size, thus we have

$$G_l = G, \quad W_l \geq W_{m+1}. \tag{32}$$

5.2 Validation

Fig. 15 shows the impact of the advertised window of the receiver on the goodput. We select a typical $W_l = 10$, According to Eqs. (29)-(32), we get that

1) As $N = 1, W_l < \frac{CD}{1}$, hence, $G_l = \frac{NS_pW_l}{D}$.
2) As $N \in [2, 8]$, $\frac{CD}{N} < W_l < W_m$, hence $G_l = CS_p$.
3) As $N = 9$, $W_l = W_m^9$, hence $G_l = \frac{NS_p\hat{Y}^C}{\hat{T}^C}$.
4) As $N > 9$, $W_l > W_m$, hence $G_l = G$.

The results shown in Fig. 15 validate that our model is accurate, and the advertised window W_l can directly affect the goodput if there are N_i flows as well as $W_l \leq W_m^{N_i}$.

Remarks. From Eq. (29), we can infer that if the advertised window size is larger than $\frac{CD}{N}$ and smaller than W_m, then the maximum aggregate throughput can be achieved. The simulation results corroborate the conclusion. Besides, ICTCP [8] solves TCP Incast problem by adjusting *awnd* to a proper value.

6 Solving TCP Incast Problem Using Priority

From the throughput model of the TCP Incast problem, we can infer that BHTO and BTTO should be eliminated to avoid the TCP Incast goodput collapse. Next, we will first present our priority-based solution to the TCP Incast problem, PRIN. Then the implementation of PRIN is described. At last,the performance of PRIN is validated on a real testbed.

6.1 Details of PRIN

6.1.1 Avoiding BHTO

BHTO occurs at the start of each data block due to too large accumulated initial congestion window size. Synchronously sending a large amount of data at the same time will overwhelm the small switch buffer and thus cause many packet losses. To avoid BHTOs, the congestion window size of each flow is reduced at the beginning of every data block.

Then how large the initial window size should be? From Eqs. (5), (18), and (19), we can infer that the aggregated congestion window size at the start of each data block is

$$\begin{aligned} A_b &= (N - N_m) \times \frac{3}{4} W_m + N_m \times \frac{3}{4} W_m^{N_m} \\ &\approx (N - N_m) \times \frac{3}{4} W_m + N_m \times \frac{3}{4} \left(W_m \times \frac{N}{N_m}\right) \\ &< 2N \times \frac{3}{4} W_m. \end{aligned} \tag{33}$$

Normally if N long-lived TCP flows, without limitations at the application layer, compete for a bottleneck, the summation of their window sizes is approximate $N \times \frac{3}{4} W_m$. Thus, the initial window of each flow should be multiplied by $\frac{1}{2}$. If the value is smaller than the slow start window size, which equals 2 if delayed ACK is enabled, otherwise 1, the slow start window size is used as the initial window.

6.1.2 Preventing BTTO

BTTO occurs if at least one of the last three packets is lost at the tail of data blocks. In this case, senders cannot receive enough duplicate ACKs to trigger the fast retransmission period and thus have to retransmit the lost packets after the retransmission timer fires. To address this problem, we can either retransmit the lost packets in advance or prevent the last three packets from dropping. In the former method, it is hard to determine a proper duration before retransmitting the packets. A small value possibly incurs many unnecessary retransmissions, while a large value causes much bandwidth wastage. Thus, in PRIN, we use a higher priority to ensure the successful transmission of the last three packets. Three packets are set to higher priority since generally three duplicated ACKs are needed to trigger the TCP FR/FR procedure [27].

6.2 Implementation

Our algorithm is implemented in Linux kernel with version 2.6.18. The `flags` in the socket interface between the applications and TCP is used as an on-off switch to enable or disable our algorithm. To ensure that TCP can work normally when our modification is disabled, TCP header is not modified and no new TCP options are added. The implementation of

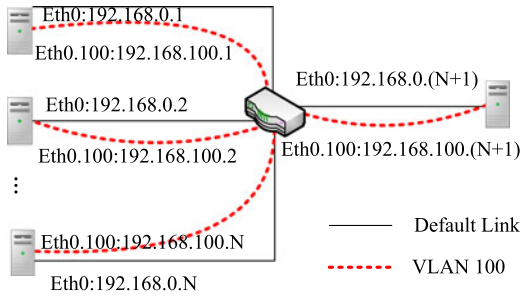

Fig. 16. VLAN structure.

PRIN has a slight change to TCP kernel. The patch of our algorithm only contains decades of lines of codes.

The implementation contains three parts: notifying data block boundaries, reducing the initial congestion window at the start of each block and configuring priority to the last three packets at the end of every block.

6.2.1 Notifying Block Boundary

Applications can use the parameter `flags` of the socket API, i.e., `flags` in function *ssize_t send (int s, const void* buf, size_t len, int flags)*, to notify TCP layer the block boundary. The type of `flags` is signed integer, which has 31 bits to be used. Until now, the maximum used `flags` is $0x800$ (MSG_MORE since Linux kernel 2.4.4) which tells TCP that the application has more data to send. Hence, we use the 16th bit in `flags` to notify TCP that the data in `buf` is the beginning of the current data block. Besides, the 17th bit is used to notify TCP layer to fragment packets in advance. Also, we use this flag as the on-off switch mentioned above to indicate whether our algorithm is enabled.

6.2.2 Reducing Congestion Window at the Start of Blocks

If the 16th bit of the `flags` is set to 1, it can be inferred that a new block starts, then the congestion window will be multiplied by $\frac{1}{2}$. The lower bound of the initial window size is set to 2 since the delayed ACK mechanism [28] is enabled in default, which delays the first ACK for $40\,\text{ms}$ if no other ACKs are to be sent.

6.2.3 Configuring Priority

IEEE 802.1p is employed in our implementation. Thus, we first configure VLAN in our testbed to enable VLAN tag, then the last three packets of a data block are set higher priority.

VLAN configuration. With VLAN tag, higher priority can be set for the last three packets. For Linux hosts, the command *vconfig add [interface-name] [vlan-id]* can add a virtual network interface card, which needs corresponding new IP address and network mask. Packets sent from the virtual interface card will have VLAN tag automatically. To configure a VLAN at a switch, all the ports of it are configured to be trunk mode. Trunk mode can support multiple VLANs in one local area network. Fig. 16 shows the VLAN configuration in our experiment. A VLAN named 100 is configured by adding a virtual network interface $eth0.100$ to each PC. $N+1$ black lines form default LAN with $192.168.0.x$ network segment, while $N+1$ red dashed lines form VLAN 100 with $192.168.100.x$ network segment. Applications can choose proper destination IP address to use default LAN or VLAN 100.

TCP fragmentation. TCP provides stream-like data transmission. The data in the socket buffer is cut into packets whose size is not larger than maximum transmission unit (MTU) before transmission. This fragmentation mechanism leads to various packet sizes. Thus it's difficult to precisely estimate the start position of the last three packets. To address this problem, our algorithm fragments data just after data is delivered to the TCP layer from the application layer. We fragment the data at the tail of each strip unit to three packets by using TCP's fragmentation function and each packet size is smaller than MTU size which ensures that TCP won't fragment these packets again.

Setting priority. After the tail of one data block is fragmented, we set higher priority for the last three packets in VLAN tag. In Linux kernel, `struct sk_buff` is the data structure of data buffer, which has a variable `priority`. We set `sk_buff.priority` to a higher priority 4 for the last three packets. Besides, to avoid this value being overwritten by the TCP option SO_PRIORITY in the IP layer, we modify the codes in the IP layer to choose the higher priority for the last three packets. To adapt the normal TCP connection, this modification is also controlled by the on-off switch of our mechanism.

7 Experimental Evaluation of PRIN

7.1 Experimental Configuration

Before conducting experiments, to make sure that the transmission rate of TCP can reach 1 Gbps, two important parameters, `net.ipv4.tcp_wmem` and `net.ipv4.tcp_rmem`, in the kernel should be enlarged. They are both a vector of three integers: `[min, default, max]`. `tcp_wmem` are used by TCP to regulate the send buffer size and `tcp_rmem` stands for receive buffer size. We modify the default sizes of both the send and the receive buffers to 128 KB. The maximum sizes of them are set to 256 KB. Then we use *Iperf* [29] to test the maximum throughput of a TCP connection between two servers that connect to a HP ProCurve 2910al switch without background traffic. The result can reach 950 Mbps.

We deploy a testbed with Dell servers, a HP ProCurve 2910al Ethernet Switch and a Cisco Catalyst 2960G Ethernet Gigabit Switch. Each PC is a DELL OptiPlex 360 desktop with Intel 2.93 GHz dual-core CPU, 6 GB DRAM, and one Intel corporation 82567LM-3 Gigabit Network Interface Card. The operating system is CentOS 5.5.

First, the performance of our algorithm PRIN is compared with TCP NewReno in a single-hop topology using HP ProCurve 2910al Ethernet Switch. Then, we add a Cisco 2960G Ethernet switch and build a multiple hop topology to evaluate the performance of our algorithm when the congestion does not happen at the last hop.

The Incast application in our experiments transmits 100 data blocks in each scenario. All the results are the average value.

7.2 Results

7.2.1 Single-Hop Topology

In this topology, PRIN is compared with TCP Newreno without or with UDP background traffic.

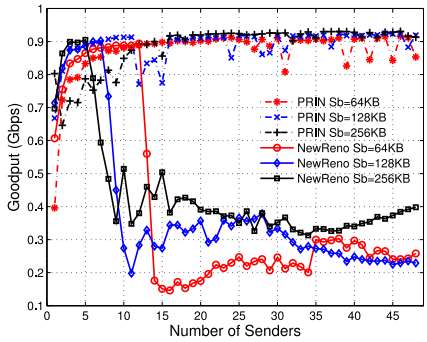

Fig. 17. The normalized goodput of TCP with different data block sizes.

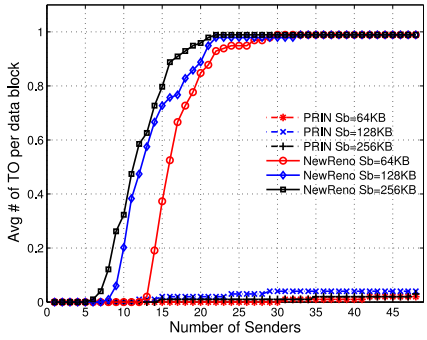

Fig. 18. Average number of TOs per data block with different block sizes.

Without background traffic. Fig. 17 shows the normalized goodput of PRIN and Newreno without background traffic. We can see that with all the three different data block sizes, TCP NewReno achieves quite small goodput when the number of senders is larger than 3. Especially, when the data block size is 64 KB, the goodput of TCP NewReno decreases from 90 percent to approximate 20 percent of the bottleneck bandwidth as the number of servers grows, while PRIN achieves high goodput which is about 90 percent of the link bandwidth. Note that the goodput is low when there is only one sender. This is because one sender does not lead to the TCP Incast problem since the sender can transmit the next data block right after it finishes the current one without waiting for the other senders. Its congestion window will not exceed the link capacity more than one packets. However, PRIN multiplies the send window size by $\frac{1}{2}$ at the beginning of each data block. Therefore, the goodput is lower than TCP Newreno. Since the number of senders in Incast communication pattern is generally large, at least more than 1, this special case can be ignored.

Fig. 18 plots the average number of timeouts during one data block which is computed in the following way. First the maximum number of timeouts of all the senders during one data block is recorded, then we compute the average value of all the 100 data blocks. Since TCP Incast goodput is determined by the slowest sender, the number of TOs shown in Fig. 18 can explain the goodput performance in Fig. 17. In TCP NewReno, the senders suffer quite many TOs per data block. The number reaches 1 as the number of senders increases, which indicates that there is about one TO period in every data block. While in PRIN, the average number of TOs is close to zero. Thus, TCP NewReno suffers goodput collapse while PRIN does not.

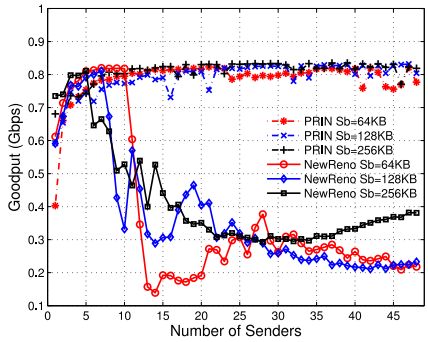

Fig. 19. Normalized goodput of TCP with UDP background traffic.

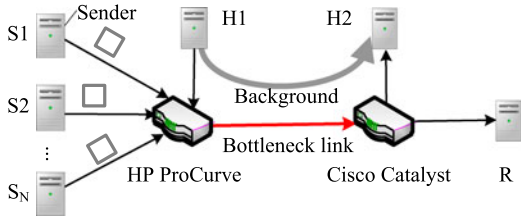

Fig. 20. Multi-hop topology.

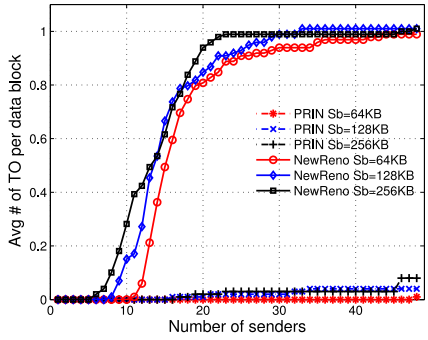

Fig. 21. Average number of TOs with UDP background traffic.

With background traffic. We add UDP background traffic using Iperf with the rate of 100 Mbps. The normalized goodput and the average number of TOs per data block are shown in Figs. 19 and 21, respectively. Fig. 19 shows that PRIN can still achieve about 80 percent utilization of the link bandwidth, which is smaller than the highest value 90 percent in Fig. 17 since UDP traffic takes 100 Mbps bandwidth. While the goodput of TCP NewReno decreases from $70 \sim 80$ percent to $20 \sim 40$ percent as the number of servers increases.

Similar to Fig. 18, the average number of TOs increases as the number of senders grows. The difference is that TCP NewReno has a little larger number of TOs on average than that without background traffic. The additional TOs are caused by the bandwidth contention of the UDP traffic. The average number of TOs per data block in PRIN is still quite small, which indicates that PRIN can work well with background traffic.

7.2.2 *Multi-Hop topology*

ICTCP only focuses on the scenario where the congestion happens at the switch port that connects to the receiver [8]. We next evaluate whether our solution PRIN can work well when the bottleneck is not the last hop as shown in Fig. 20.

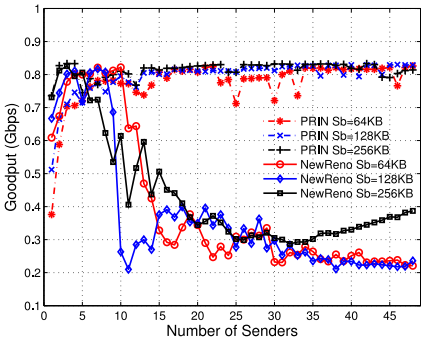

Fig. 22. Normalized goodput in multi-hop topology.

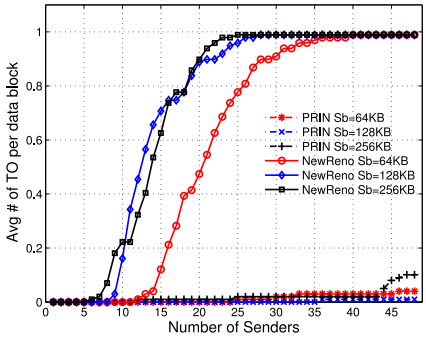

Fig. 23. Average number of TOs per data block in multi-hop topology.

This kind of scenarios possibly happen in data centers, such as the reducer fetches data from the mappers that do not locate in the same rack as the reducer and one intermediate link is congested.

Figs. 22 and 23 show the throughput and the number of timeouts happened in a block with 100 Mbps UDP background traffic on the intermediate link, respectively. Similar to the performance with background traffic in the single-hop topology, PRIN still exhibits stable and high goodput, which indicates that PRIN works well no matter where the congestion link is located. The difference is that the maximum throughput is about 800 Mbps since 100 Mbps bandwidth is taken by the background traffic.

8 Conclusions

In this paper, an analytical model is built to understand the essential causes of the TCP Incast problem. The existing investigations on the problem try to find an effective solution to address it. However, they either are hard to be deployed, such as substituting TCP by new transport protocols, or only can temporarily mitigate goodput drop, such as reducing `RTOmin`.

To solve the TCP Incast problem substantially, the fundamental reasons should be first explored. we found that two types of TOs, BTTO and BHTO, significantly degrade the TCP goodput. The critical point between them is the onset of TCP goodput collapse. BTTO caused by one of the last three packets in a block being dropped happens when the number of concurrent senders is small, while BHTO caused by the first whole window loss happens when the number of concurrent senders becomes large. The proposed model is validated by comparing with simulation data. We found that our model well characterizes the goodput of TCP Incast.

Based on the insights provided by the proposed model, we design a simple mechanism PRIN by leveraging the IEEE 802.1p technology to eliminate BHTOs and BTTOs. PRIN modifies TCP a little and poses no impact on the other applications. The experimental results on a real testbed demonstrate the effectiveness of our solution, which also corroborates the goodput model.

Acknowledgments

The authors gratefully acknowledge the anonymous reviewers for their constructive comments. This work was supported in part by the National Basic Research Program of China (973 Program) under Grant No. 2014CB347800 and 2012CB315803, and the National Natural Science Foundation of China (NSFC) under Grant No. 61225011. Part of this paper was presented at the IEEE INFOCOM, 2011 [1].

References

[1] J. Zhang, F. Ren, and C. Lin, "Modeling and understanding TCP incast in data center networks," in *Proc. IEEE Conf. Comput. Commun.*, 2011, pp. 1377–1385.

[2] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, "Measurement and analysis of TCP throughput collapse in cluster-based storage systems," in *Proc. 6th USENIX Conf. File Storage Technol.*, 2008, pp. 1–14.

[3] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and effective fine-grained TCP retransmissions for datacenter communication," in *Proc. ACM SIGCOMM Conf. Data Commun.*, Aug. 2009, pp. 303–314.

[4] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding TCP incast throughput collapse in datacenter networks," in *Proc. 1st ACM Workshop Res. Enterprise Netw.*, 2009, pp. 73–82.

[5] D. Nagle, D. Serenyi, and A. Matthews, "The panasas activescale storage cluster: Delivering scalable high bandwidth storage," in *Proc. ACM/IEEE Conf. Supercomput.*, 2004, pp. 53–62.

[6] M. Abd-El-Malek, W. Courtright, C. Cranor, G. R. Ganger, J. Hendricks, A. J. Klosterman, M. Mesnier, M. Prasad, B. Salmon, R. R. Sambasivan, S. Sinnamohideen, J. D. Strunk, E. Thereska, M. Wachs, and J. J. Wylie, "Ursa minor: Versatile cluster-based storage," in *Proc. 4th Conf. USENIX Conf. File Storage Technol.*, 2005, p. 5.

[7] S. Ghemawat, H. Gobioff, and S. Leung, "The google file system," in *Proc. 19th ACM Symp. Oper. Syst. Principles*, 2003, pp. 29–43.

[8] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast congestion control for TCP in data center networks," in *Proc. ACM CoNEXT*, 2010, pp. 1–12.

[9] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan,, "DCTCP: Efficient packet transport for the commoditized data center," in *Proc. ACM SIGCOMM Conf.*, Aug. 2010, pp. 1–12.

[10] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, Sep. 1998, pp. 303–314.

[11] N. Parvez, A. Mahanti, and C. Williamson, "An analytic throughput model for TCP NewReno," *IEEE/ACM Trans. Netw.*, vol. 18, no. 2, pp. 448–461, Apr. 2010.

[12] E. Altman, K. Avrachenkov, and C. Barakat, "A stochastic model of TCP/IP with stationary random losses," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, Aug. 2000, pp. 231–242.

[13] M. Goyal, R. Guerin, and R. Rajan, "Predicting TCP throughput from non-invasive network sampling," in *Proc. IEEE Conf. Comput. Commun.*, Mar. 2002, pp. 180–189.

[14] A. Kumar, "Comparative performance analysis of versions of TCP in a local network with a lossy link," *IEEE/ACM Trans. Netw.*, vol. 6, no. 4, pp. 485–498, Aug. 1998.

[15] IEEE standards for local and metropolitan area networks: Virtual bridged local area networks [Online]. Available: http://standards.ieee.org/getieee802/download/802.1Q-1998.pdf, 1998.

[16] RFC 2474-definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers [Online]. Available: http://www.faqs.org/rfcs/rfc2474.html, 1998.
[17] RFC 791-internet protocol specification [Online]. Available: http://www.faqs.org/rfcs/rfc791.html, 1981.
[18] LAN QoS: Access switches get intelligent for high-stakes applications. (Mar. 2012) [Online]. Available: http://searchnetworking.techtarget.com/tip/LAN-QoS-Access-switches-get-intelligent-for-high-stakes-applications
[19] M. Yu, J. Rexford, X. Sun, S. Rao, and N. Feamster, "A Survey of virtual LAN usage in campus networks," *IEEE Commun. Mag.*, vol. 49, no. 7, pp. 98–103, Jul. 2011.
[20] VXLAN: A framework for overlaying virtualized layer 2 networks over layer 3 networks. (Aug. 2011) [Online]. Available: http://tools.ietf.org/pdf/draft-mahalingam-dutt-dcops-vxlan-00.pdf
[21] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowstron, "Better never than late: Meeting deadlines in datacenter networks," in *Proc. ACM SIGCOMM Conf.*, 2011, pp. 50–61.
[22] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2012, pp. 127–138.
[23] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pFabric: Minimal near-optimal datacenter transport," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 435–446.
[24] R. Pan, B. Prabhakar, and A. Laxmikantha, "QCN: Quantized congestion notification," IEEE 802.1 Qau Presentation, 2007.
[25] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani, "Scaling memcache at facebook," in *Proc. 10th USENIX Conf. Netw. Syst. Des. Implementation*, 2013, pp. 385–398.
[26] S. Floyd, T. Henderson, and A. Gurtov, "The newreno modification to TCP's fast recovery algorithm," in *RFC 3782*, Apr. 2004.
[27] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control," in *RFC 2581*, Apr. 1999.
[28] R. Braden, "Requirements for internet hosts—Communication layers," in *RFC 1122*, Oct. 1989.
[29] Iperf [Online]. Available: http://iperf.sourceforge.net/

Jiao Zhang is currently an assistant professor in the School of Information and Communication Engineering at Beijing University of Posts and Telecommunications (BUPT), Beijing, China. In July 2014, she received her PhD degree from the Department of Computer Science and Technology, Tsinghua University, China. Her supervisor is Prof. Fengyuan Ren. From August 2012 to August 2013, she was a visiting student in the networking group of ICSI, UC Berkeley. In July 2008, she obtained her Bachelor's Degree from the School of Computer Science and Technology from BUPT. Her research interests include traffic management in data center networks, routing in wireless sensor networks and future Internet architecture. Until now, she has (co)-authored more than 10 international journal and conference papers. She is a member of the IEEE.

Fengyuan Ren received the BA and MSc degrees in automatic control and the PhD degree in computer science from Northwestern Polytechnic University, China, in 1993, 1996, and 1999, respectively. He is currently a professor in the Department of Computer Science and Technology at Tsinghua University, Beijing, China. From 2000 to 2001, he was at the Electronic Engineering Department of Tsinghua University as a postdoctoral researcher. In Jan. 2002, he moved to the Computer Science and Technology Department of Tsinghua University. His research interests include network traffic management and control, control in/over computer networks, wireless networks, and wireless sensor networks. He coauthored more than 80 international journal and conference papers. He has served as a technical program committee member and local arrangement chair for various IEEE and ACM international conferences. He is a member of the IEEE.

Li Tang graduated from Tsinghua University as an undergraduate in 2009 and is currently working toward the master's degree in the Department of Computer Science and Technology, Tsinghua University under the supervision of Prof. Fengyuan Ren. His research interests include traffic control in various networks, including wireless networks, internet and data center networks.

Chuang Lin received the PhD degree in computer science from Tsinghua University, China in 1994. He is a professor in the Department of Computer Science and Technology at Tsinghua University, Beijing, China. He is an honorary visiting professor, University of Bradford, United Kingdom. His current research interests include computer networks, performance evaluation, network security analysis, and Petri net theory and its applications. He has published more than 300 papers in research journals and IEEE conference proceedings in these areas and has published four books. He served as the technical program vice chair, the 10th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS 2004); the general chair, ACM SIGCOMM Asia Workshop 2005 and the 2010 IEEE International Workshop on Quality of Service (IWQoS 2010). He is an associate editor of *IEEE Transactions on Vehicular Technology* and an area editor of Computer Networks and the *Journal of Parallel and Distributed Computing*. He is a senior member of the IEEE and the Chinese Delegate in TC6 of IFIP.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.