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

Modeling and Understanding TCP Incast in Data Center Networks

Jiao Zhang, Fengyuan Ren, Chuang Lin
Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, China
Dept. of Computer Science and Technology, Tsinghua University, Beijing, 100084, China
Email: {zhangjiao08, renfy, clin}@csnet1.cs.tsinghua.edu.cn

***Abstract*—Recently, TCP incast problem attracts increasing attention since the receiver suffers drastic goodput drop when it simultaneously strips data over multiple servers. Lots of attempts have been made to address the problem through experiments and simulations. However, to the best of our knowledge, few solutions can solve it fundamentally at low cost. In this paper, a goodput model of TCP incast is built to understand why goodput collapse occurs. We conclude that TCP incast goodput deterioration is mainly caused by two types of timeouts, one happens at the tail of a data block and dominates the goodput when the number of senders is small, while the other one at the head of a data block and governs the goodput when the number of senders is large. The proposed model describes the causes of these two types of timeouts which are related to the incast communication pattern, block size, bottleneck buffer and so on. We validate the proposed model by comparing with simulation data, finding that it can well characterize the features of TCP incast. We also discuss the impact of most parameters on the goodput of TCP incast.**

***Index Terms*—Data Center Networks, TCP incast, Modeling, Goodput**

I. Introduction

TCP incast has risen to be a critical problem recently in data center networks due to its catastrophic goodput collapse [1]–[3]. *Incast*, a communication pattern, was first termed by Nagle *et al.* in [4]. In incast communication pattern, multiple senders concurrently transmit data blocks to a single receiver, and any sender can not send another data block until all the senders finish transmitting the current data block. When 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 [4]–[6], web research [7] and MapReduce [8]. To avoid the performance deterioration of TCP incast, lots of attempts have been made to find the causes of TCP incast and the methods to solve it [1]–[3].

The existing approaches to solve TCP incast problem can be classified into four categories. First, avoiding timeout in TCP [1], [3]. Experiment and simulation results show that too many TimeOut (TO) periods in TCP incast lead to goodput collapse. Therefore, several trials have been made to avoid TOs. For instance, reducing duplicate ACK threshold of entering Fast Retransmission (FR) from 3 to 1, disabling slow start phase, and trying different TCP versions. However, most of these methods are ineffective. Second, reducing minimum Retransmission Timeout (RTO_{min}) [2]. RTO_{min} typically equals to 0.2s in TCP, which will result in a big waste of bandwidth in data center networks where the link capacity is quite high. Hence, Vasudevan *et al.* suggested reducing RTO_{min} to microsecond-granularity to reduce the capacity waste during TO periods. This method not only needs to modify the Linux kernel timer into the higher resolution, which is difficult to be implemented, but also is not very safe in the networks with larger Round Trip Time (RTT). More importantly, if optical fibre, whose current maximum rate is 12.8 Tbps, becomes popular in data center networks in future [9], even if microsecond order RTO_{min} will still elicit unignorable capacity loss. Therefore, reducing RTO_{min} temporarily mitigates TCP incast, but not solves it fundamentally. Third, replacing TCP. The engineers in Facebook have adopted this crude method to avoid TCP incast [7]. They employed UDP as transport layer protocol and endowed the application layer the responsibility of flow control. Yet TCP is so popular that replacing it will cost too much. Fourth, employing other mechanisms except from modifying TCP. A. Phanishayee *et al.* proposed using Ethernet Flow Control to solve TCP incast [1]. However, it can not work well if multiple switches exist between the senders and the receiver due to head of block.

In sum, sorts of solutions to TCP incast have been proposed. Unfortunately, most of them have various limitations. To substantially solve TCP incast at low lost, firstly we need to thoroughly understand why it happens. In this paper, a goodput model of TCP incast is built to understand TCP incast in depth. Although there are many literatures on TCP modeling [10]–[14], our modeling is different in three aspects: (1) The application in our model exhibits *incast* communication pattern. Yet, existing TCP models usually assume that the application layer always passes enough data to the transport layer. (2) TCP incast model describes the overall goodput of the bottleneck link which contains multiple flows, while most of existing TCP models focus on the throughput of only one flow. (3) The RTT in previous work is generally assumed to be constant since it is difficult to be accurately computed in a network with complicated and unknown topology. However, in TCP incast environment, more precise RTT model is needed to characterize the causes of goodput collapse.

In our TCP incast model, we summarize that the goodput collapse in TCP incast is mainly caused by two kinds of TOs.

- Block Tail TimeOut (BTTO): It is caused by the special incast communication pattern. Since each sender can not

get the next block data from the application layer until all the senders finish transmitting the current block, if one of the last three (Assume three duplicate ACKs are needed to trigger FR.) packets in current block is dropped, then there will not enough ACKs to trigger FR, timeout naturally occurs.

- Block Head TimeOut (BHTO): BHTO is apt to happen when the number of senders becomes larger. During transmitting a block, some of the senders will finish earlier due to TCP unfairness in small timescale. Then they will wait for the others to finish without taking any bandwidth. Therefore the other flows will finish their blocks using more capacity in average, which results in higher window sizes when they finish the current block. At the beginning of the next block, 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 window of each flow becomes smaller as N increases, then it will enter a TO period.

Investigating the causes of these two kinds of TOs in depth is beneficial to develop an effective and simple solution to avoid goodput collapse of TCP inast.

The remainder of the paper is organized as follows. We first introduce the main assumptions and notations in Section II. Subsequently, the goodput of TCP incast is modeled in Section III in detail. In Section IV, the model results are compared with the simulation results and the impact of different parameters upon TCP goodput is analyzed. Finally, the paper is concluded in Section V.

II. Assumptions and Notations

A. Assumptions

1) TCP Incast Scenario: Assume that only packets of the synchronized data blocks are transmitted through the bottleneck link, and the bottleneck buffer employs Drop Tail queue management scheme. Also, we assume that packets will be lost only when the bottleneck buffer overflows, namely, the packets will not be dropped due to other reasons, such as link failure. Assume that all the windows evolutions of the flows are synchronized when the number of senders is small. 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, so we assume the number of senders is smaller than the buffer size.

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 for triggering FR phase is 3. Since the unabiding slow start process imposes a negligible impact on TCP througput, it is igonred in our modeling.

B. Notations

Before defining the notations, we first introduce a concept called *round*. The first round starts from a Congestion Avoidance (CA) period and lasts one RTT. The after round starts from the end of the last round and lasts one RTT. A CA period ends 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.

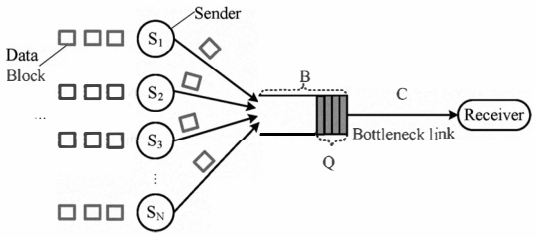

Figure 1. A scenario of TCP incast, where multiple senders concurrently transmit data blocks to a single receiver.

TABLE I
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

A scenario of TCP incast is shown in Figure 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 I for the sake of terseness.

III. Modeling Goodput of TCP Incast

The goodput of TCP NewReno [15] in the Incast environment will be modeled in this section. As aforementioned in Section I, two types of TOs lead to TCP goodput drop. We will first show them in Figures 2 and 3 which are plotted based on the results of simulations conducted on the ns-2 platform.

Figure 2 shows the scenario where BTTO happens. 8 senders transmit synchronized data block to the same receiver. The figure plots the window evolution of two senders among them. A pentagram plotted at $(t, 20)$ represents that a block finishes at time t. The big X represents a retransmission timer is fired. The advertised window size of the receiver is set to 1000 packets, which is large enough that it has no impact on the sending 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

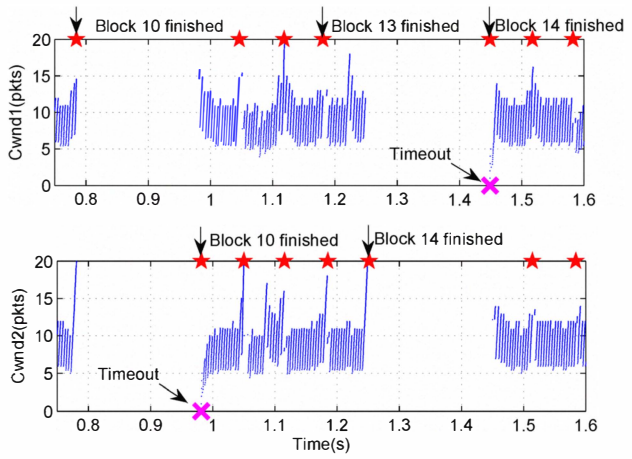

Figure 2. The scenario where BTTO happens. $N = 8$ senders concurrently transmit packets to the same receiver. The packet size $S_p = 1$KB, bottleneck bandwidth $C = 1$Gbps=12.5pkts, buffer $B = 64$packets, synchronized block $S_b = 1024$KB. The advertised window of the receiver is set to 1000 packets. 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.

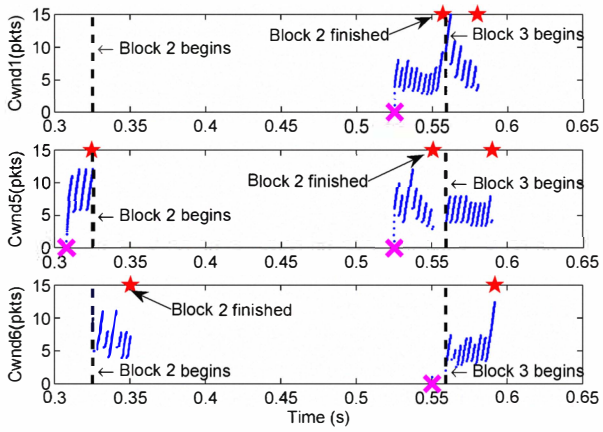

Figure 3. The scenario where BHTO happens. The main parameters of this scenario are: $N = 32, S_p = 1KB, B = 64, S_b = 256KB, C = 12.5pkts$. The window evolutions of sender 1, 5, 6 are plotted to illustrate BHTO.

suffers a TO period before finishing Block 10 during time about $(0.79 \sim 0.98)$s. By observing the congestion window evolution of TCP 2, we find 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 injected into the transport layer from the application layer until all senders finish Block 10. Therefore, sender 2 can only wait until the retransmission timer fires at about time 0.98s, namely, 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 can not 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 1.25s, sender 1 delays finishing Block 14 due to the same kind of timeout and therefore also degrades goodput.

Figure 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 1000 packets. We plot three of the 32 flows to illustrate the behavior of BHTO. The pentagram represents that a block is finished. At time 0.325s, 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 0.52s. And at 0.56s, both of them finish Block 2. It can be inferred that all the other flows also finish Block 2 before 0.56s since the transmitting of Block 3 begins at this moment. With respect to Block 3, all the three senders luckily finish their 3rd 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 the transmission of Block 3.

Through investigating numerous simulation data, we find that BTTO dominates TCP goodput when N is small while BHTO does when N is large. Let $N^*(1 \leq N^* \leq N)$ denote the critical value between these two situations. We will firstly 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 significantly deteriorates TCP goodput will be modeled .

A. Goodput As $N < N^*$

1) Dynamics of Queue Length: Considering a CA period, during the i-th round, 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 i-th round, as follows.

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

a^+ equals to a if $a > 0$, else equals to 0.

2) Relationship Between RTT and Queue Length: Assume that the queue is FIFO 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:

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

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 that of the departure traffic.

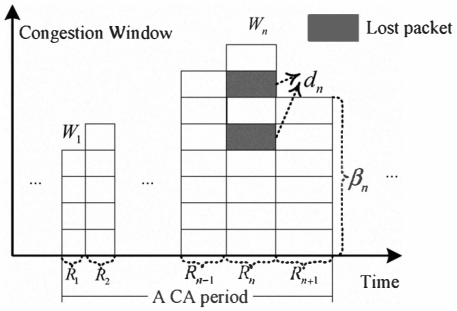

Figure 4. Congestion window evolution during a CA period

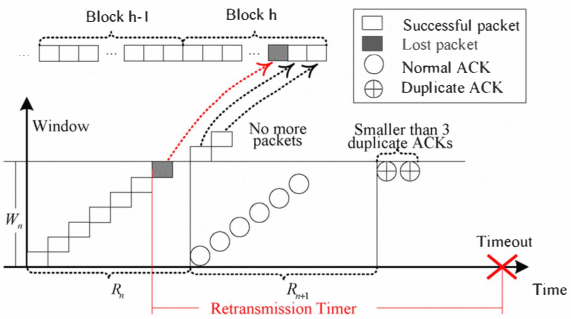

Figure 5. The scenario where a BTTO happens

3) Number of Packets Successfully Transmitted in a CA Period: Figure 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

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

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 Eq. (1) and (2), we can infer

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

In 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 m-th 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 $W_m + 1$. Therefore, 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

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

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 Eq. (3) and (7), we can obtain that

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

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 Eq. (2) and (4), we have

$$R_i = \frac{NW_{i-1}}{C} \tag{9}$$

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:

$$T_n = \sum_{i=1}^{n+1} R_i = \frac{N}{C}\Big(\frac{3}{8}(W_n)^2 + W_n\Big) \tag{10}$$

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

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

5) Probability of Block Tail Timeout (BTTO): Figure 5 illustrates when timeout events happen at the tail of blocks. As long as any one of the last three packets in a block is lost, then a timeout event will appear due to inadequate ACKs for triggering FR. As shown in Figure 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 timeout 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 k-th CA period is one of the last three packets in a block, namely

$$kY_N^C - \alpha = hY^B - \beta, \quad k \text{ and } h \text{ are integers} \tag{12}$$

where α, a stochastic variable, is the number of packets successfully transmitted after a lost packet in the k-th 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

$$\alpha = \begin{cases} W_m - 1 & \text{with probability } \frac{N_m}{N} \\ W_{m+1} - 1 = W_m & \text{with probability } (1 - \frac{N_m}{N}) \end{cases} \tag{13}$$

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

$$k_\alpha^f = \min\{y(\alpha), y(\alpha - 1), y(\alpha - 2)\} \tag{14}$$

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 to the BTTO period even if they have successfully transmitted Block h (Figure 2). This is because they can not get the next block data from the application layer. Therefore, the probability of the BTTO period of a flow is the minimum 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

$$\alpha_1 = \begin{cases} W_m - 1 & \text{if } k_{W_m - 1}^f < k_{W_m}^f \\ W_m & \text{else} \end{cases} \tag{15}$$

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

$$k_{min} = \begin{cases} k_{\alpha_1}^f & \text{with probability } 1 - (Pr[\alpha = \alpha_2])^N \\ k_{\alpha_2}^f & \text{with probability } (Pr[\alpha = \alpha_2])^N \end{cases} \tag{16}$$

Thus, the expectation of k_{min} is

$$\mathbb{E}(k_{min}) = \sum_{i=1}^{2} k_{\alpha_i}^f Pr[\alpha = \alpha_i] \tag{17}$$

Hence, the probability of entering a TO period from a CA period is

$$P^O = \frac{1}{\mathbb{E}(k_{min})} \tag{18}$$

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 to RTO_{min} since the RTT of data center networks is quite small. The duration of a TO period equals to the first timeout duration, namely, $T^O = T_0$.

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:

$$\begin{aligned} G_1 &= N \frac{Y_N^C}{T_N^C + P^O T^O} S_p \\ &= \frac{N S_p \left\lfloor \frac{3}{8} W_m^2 + 2W_m + \frac{5}{8} - (\frac{3}{4} W_m + \frac{13}{8}) \frac{N_m}{N} \right\rfloor}{\frac{N}{C} \left(\frac{3}{8} W_m^2 + \frac{7}{4} W_m + \frac{11}{8} - (\frac{3}{4} W_m + \frac{11}{8}) \frac{N_m}{N} \right) + P^O T_0} \end{aligned} \tag{19}$$

Remarks: If there are only CA periods, then according to the expression of Y_N^C and T_N^C, TCP goodput of all the N senders approximately equals to 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_N^C, *i.e.*, LCM(Y^B, Y_N^C). Thereby, larger block size Y^B decreases P^O and thus improves goodput G_1.

B. Goodput As $N > N^*$

1) Calculation of N^*: By observing simulation results, we find 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 can not be accommodated by the buffer, some of them will be dropped. A unlucky flow, which unfortunately loses all the packets in its window, will enter a TO period. While in the subsequent rounds, the congestion windows are regulated by CA procedure. Only one packet is transmitted after receiving an ACK. 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 response to packet droppings and thus few timeout events appear due to full window losses. We refer to the TO periods which occur at the start of blocks as Block Head TimeOut (BHTO).

In reality, the two types of TO periods, *i.e.*, BTTO and BHTO, likely happen when N equals to some values. Next the critical point N^* is computed. If N^* senders transmit data to the same client, then in 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

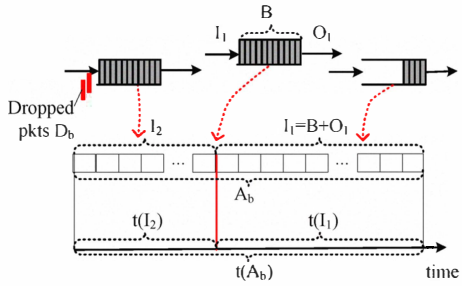

Figure 6. The behavior of the packets in the first windows of all the flows at the beginning of Block b.

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 can not be ignored. As stated in Section III-A3, 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

$$W_m^{N_m} = \begin{cases} \lfloor \frac{CD+B}{N_m} + \frac{1}{2} \rfloor + 1 & N_m > 0 \\ W_m & N_m = 0 \end{cases} \quad (20)$$

Hence, we can obtain that

$$A_b = (N - N_m) \times \frac{3}{4} W_m + N_m \times \frac{3}{4} W_m^{N_m} \quad (21)$$

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 backlog in the buffer, namely

$$D_b = A_b - (\tau C + B), 0 < \eta < 1 \quad (22)$$

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 we assume the capacity of the link between each sender and the intermediate switch is C pkts, the maximum time taken by the senders is $\tau = \frac{W_m^{N_m}}{C}$.

Next 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 Figure 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

$$N_t = N \times P_t = N \times \frac{I_2}{A_b} \quad (23)$$

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

$$P_w = (P_p)^W = (\frac{D_b}{I_2})^W \quad (24)$$

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,

$$I_2 = A_b - \frac{BA_b}{A_b - \tau C} \quad (25)$$

Combining Eq. (23)-(25), we can obtain that the expected number of flows entering TO period after the first round is

$$\begin{aligned} N^O &= N_t \times (P_p)^{\bar{W}} \\ &= N(1 - \frac{B}{A_b - \tau C}) \times (\frac{D_b}{A_b - \frac{BA_b}{A_b - \tau C}})^{\bar{W}} \end{aligned}$$

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

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

2) Goodput: When $N = N^*$, in average one flow will enter TO at the beginning of each block. And when $N > N^*$, we can infer that in 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 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 a 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 can not 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

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

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_N^C in Section III-A3, 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

$$Y^F_{(N-N^*+1)} = Y^C_{(N-N^*+1)} + \frac{W_n}{2} - d \tag{26}$$

$$T^F_{(N-N^*+1)} = T^C_{(N-N^*+1)} + D \tag{27}$$

The time that one of the $(N - N^* + 1)$ unlucky flows spends transmitting a block decides 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

$$T^B = T_0 + \frac{Y^B}{Y^F_{(N-N^*+1)}} T^F_{(N-N^*+1)} \tag{28}$$

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

$$G_2 = N \times \frac{Y^B}{T^B} \times S_p = \frac{NS_pY^BY^F_{(N-N^*+1)}}{T_0Y^F_{(N-N^*+1)} + Y^BT^F_{(N-N^*+1)}} \tag{29}$$

Combining Eq. (19) and (29), we can obtain that the TCP goodput of N senders concurrently transmitting data blocks to a receiver is

$$G = \begin{cases} \frac{NS_p\left\lfloor \frac{3}{8}W_m^2 + 2W_m + \frac{5}{8} - (\frac{3}{4}W_m + \frac{13}{8})\frac{N_m}{N} \right\rfloor}{\frac{N}{C}\left(\frac{3}{8}W_m^2 + \frac{7}{4}W_m + \frac{11}{8} - (\frac{3}{4}W_m + \frac{11}{8})\frac{N_m}{N}\right) + P^OT_0} & N < N^* \\ \frac{NS_pY^BY^F_{(N-N^*+1)}}{T_0Y^F_{(N-N^*+1)} + Y^BT^F_{(N-N^*+1)}} & N \geq N^* \end{cases}$$

C. Goodput with Window Limitation

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

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

$$G_l = NS_p\frac{W_l}{R_l} \tag{30}$$

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:

$$G_l = \begin{cases} \frac{NS_pW_l}{D} & W_l \leq \frac{CD}{N} \\ CS_p & \frac{CD}{N} < W_l < W_m \end{cases} \tag{31}$$

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 dropping one packet based on the analysis in Section III-A3, 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 packets successfully transmitted is

$$\hat{Y}^C = \frac{N_m}{N}Y_m + (1 - \frac{N_m}{N})W_mT_m \tag{32}$$

where Y_m and T_m is defined in Eq. (3) and (10), 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

$$G_l = \frac{NS_p\hat{Y}^C}{\hat{T}^C}, \quad W_l = W_m \tag{33}$$

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

$$G_l = G, \quad W_l \geq W_{m+1} \tag{34}$$

IV. Validation and Analysis

In this section, we validate our model through simulations on the ns-2 platform and discuss the impact of some parameters upon TCP incast goodput. The module for TCP incast is developed by A. Phanishayee *et.al.* in [1]. The RTO_{min} is set to be 0.2s.

A. Without Window Limitation

1) Different Buffer Size B: Figure 7-9 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 link capacity between each sender and the intermediate switch equals to C.

Obviously, the model well characterizes the general tendency of TCP incast, which indicates that the two types of TOs, BTTO and BHTO, indeed are the essential causes of TCP incast. Besides, we find 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 Figure 7-9, 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 $W_m = \lfloor \frac{CD+B}{N} + \frac{1}{2} \rfloor + 1$. Then the expected number of packets successfully transmitted in a CA

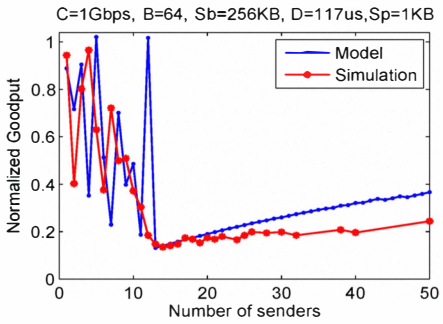

Figure 7. Normalized goodput with 64KB buffer

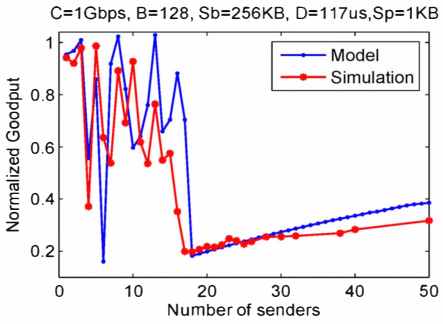

Figure 8. Normalized goodput with 128KB buffer

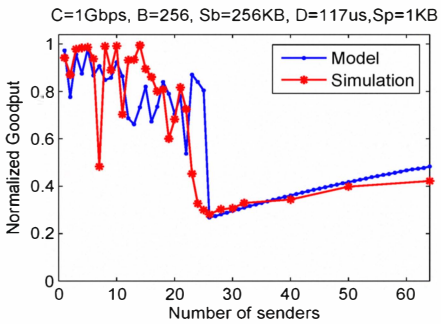

Figure 9. Normalized goodput with 256KB buffer

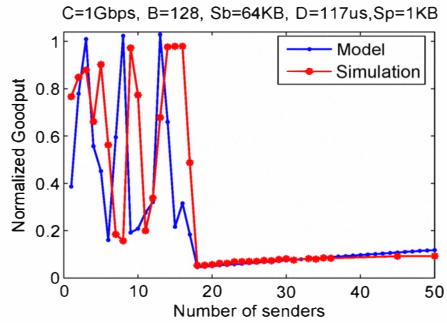

Figure 10. Normalized goodput with 64KB block

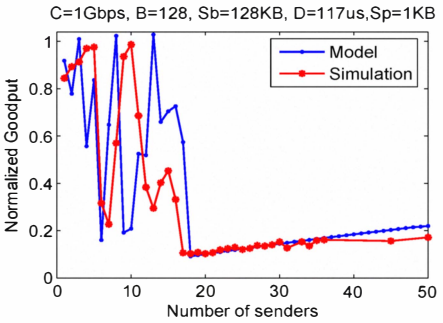

Figure 11. Normalized goodput with 128KB block

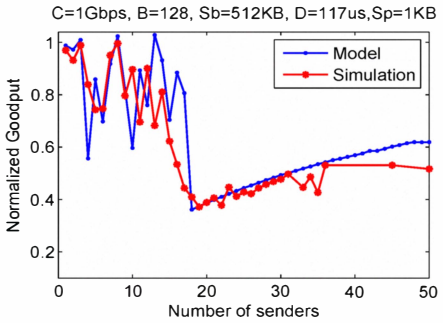

Figure 12. Normalized goodput with 512KB block

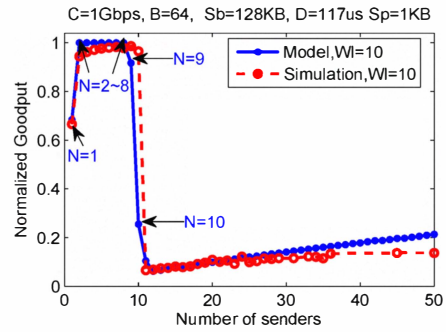

Figure 13. Normalized goodput with window limitation $W_l = 10$

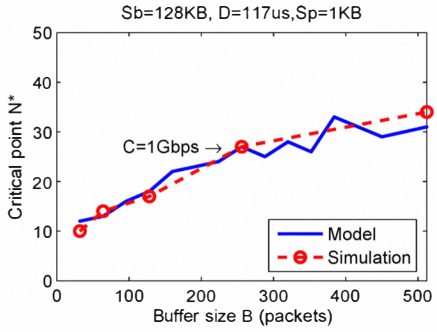

Figure 14. Critical point N^* with different B and $C = 1$Gbps

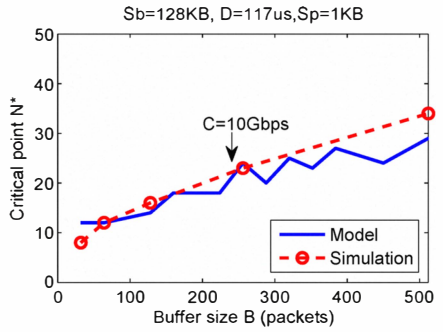

Figure 15. Critical point N^* with different B and $C = 10$Gbps

period, Y_N^C, increases. When $N < N^*$, the probability of a TO period has a negative correlation with 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. TCP incast suffers quite low goodput when N equals to 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 to 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. While the number of senders N increases. Hence, goodput slowly increases. 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.

2) Different Synchronized Data Block S_b: Figures 10-12 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 $N < N^*$, 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 of TO will have a decline tendency and consequently the goodput will increase. When $N > N^*$, since in average one TO happens in each block. Thereby, when 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.

B. With Window Limitation

Figure 13 shows the impact of the advertised window of the receiver on the goodput. We select a typical $W_l = 10$, According to Eq. (31)-(34), 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 Figure 13 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 $W_l \leq W_m^{N_i}$.

C. Parameter Analysis

N^* is a quite important point since it is the critical point of the goodput collapse. We conduct a series of simulations with different N, B, C, and solve the value of N^* using our analytical model. The results are presented in Figure 14-15. 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 with a particular N. Therefore, larger C does not delay the onset of goodput collapse. With respect to buffer size B, larger buffer can temporarily keep more packets to prevent them from being dropped. Hence, larger B can delay the onset of goodput collapse.

V. Conclusions

In this paper, we build an analytical model to understand the essential causes of TCP incast, which is a crucial issue in data center networks due to its catastrophic performance deterioration as the number of senders becomes large. The existing investigations on it try to find a good solution to TCP incast. However, they either need high cost, such as substituting TCP by UDP, or only can temporarily mitigate goodput drop, such as reducing RTO_{min}. To solve TCP incast substantially, the fundamental reasons should be firstly explored. In our work, we find 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, which is caused by one of the last three packets in a block being dropped, happens when the number of concurrent senders is small. While BHTO is caused by dropping the first window of a block totally. It happens when the number of concurrent senders becomes large. At last, we validate the proposed model by comparing it with simulation data, finding that our model well characterizes the goodput of TCP incast. The insights provided by the proposed model will be helpful for developing more effective solutions to TCP incast at low cost.

Acknowledgement

This work is supported in part by the National Natural Science Foundation of China (NSFC) under Grant No.60773138, 60971102, the National Grand Fundamental Research Program of China (973) under Grant No. 2009CB320504, 2010CB328105, and National Science and Technology Major Project of China (NSTMP) under Grant No. 2009ZX03006-001-003 and 2009ZX03006-003-01.

References

[1] 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. of FAST*, 2008.
[2] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson1, and B. Mueller, "Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication," in *Proc. of ACM SIGCOM*, Aug.2009, pp. 303–314.
[3] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D.Joseph, "Understanding TCP Incast Throughput Collapse in Datacenter Networks," in *Proc. of ACM WREN*, 2009.
[4] D. Nagle, D. Serenyi, and A. Matthews, "The Panasas ActiveScale Storage Cluster: Delivering Scalable High Bandwidth Storage," in *Proc. of the 2004 ACM/IEEE conference on Supercomputing*, 2004, pp. 53–62.
[5] 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. S. andJohn D. Strunk, E. Thereska, M. Wachs, and J. J. Wylie, "Ursa Minor: Versatile Cluster-based Storage," in *Proc. of FAST*, 2005.
[6] S. Ghemawat, H. Gobioff, and S. Leung, "The Google File System," in *Proc. of SOSP*, 2003, pp. 29–43.
[7] "Presentation Summary-high Performance at Massive Scale Lessons Learned at Facebook." [Online]. Available: http://idleprocess.wordpress.com/2009/11/24/presentation-summary-high-performance-at-massive-scale-lessons-learned-at-facebook/
[8] J. Dean, S. Ghemawat, and G. Inc, "Mapreduce: Simplified data processing on large clusters," in *Proceedings of OSDI*, 2004.
[9] C. Minkenberg, A. Scicchitano, and M. Gusat, "Adaptive Routing for Convergence Enhanced Ethernet," in *Proceedings of HPSR*, 2009, pp. 34–41.
[10] J.Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," in *Proc. of ACM SIGCOM*, Sep.1998, pp. 303–314.
[11] N. Parvez, A. Mahanti, and C. Williamson, "An Analytic Throughput Model for TCP NewReno," in *Proc. of IEEE/ACM TON*, vol. 18, no. 2, April 2010, pp. 448–461.
[12] E. Altman, K. Avrachenkov, and C. Barakat, "A Stochastic Model of TCP/IP with Stationary Random Losses," in *Proceedings of ACM SIGCOMM*, Aug. 2000, pp. 231–242.
[13] M. Goyal, R. Guerin, and R. Rajan, "Predicting TCP Throughput from Non-invasive Network Sampling," in *Proceedings of IEEE INFOCOM*, Mar. 2002, pp. 180–189.
[14] A. Kumar, "Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link," in *IEEE/ACM TON*, vol. 6, no. 4, Aug. 1998, pp. 485–498.
[15] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm," in *RFC 3782*, Apr. 2004.