Comprehensive Understanding of TCP Incast Problem

Wen Chen*, Fengyuan Ren*, Jing Xie*, Chuang Lin*, Kevin Yin† and Fred Baker†
* Tsinghua National Laboratory for Information Science and Technology
Dept. of Computer Science and Technology, Tsinghua University, Beijing, 100084, China
Email: {chenwen, renfy, xie-j13, clin}@csnet1.cs.tsinghua.edu.cn
†Cisco Systems Email: {kyin, fred}@cisco.com

***Abstract*—Since TCP Incast has been identified as a catastrophic problem in many typical data center applications, a lot of efforts have been made to analyze or solve it. The analysis work intends to model Incast problem from certain perspective, and the solutions try to solve the problem through designing enhanced mechanisms or algorithms. However, the proposed models are either closely coupled with particular protocol version or dependent on empirical observations, and the solutions cannot eliminate Incast problem entirely because the underlying issues are not identified completely. There is little work which attempts to close the gap between "analyzing" and "solving", and present a comprehensive understanding.**

In this paper, we provide an in-depth understanding of how TCP Incast problem happens. We build up an interpretive model which emphasizes particularly on describing qualitatively how various factors, including system parameters and mechanism variables, affect network performances in Incast traffic pattern, but not on calculating the accurate throughput. With this model, we give plausible explanations why the various solutions for TCP Incast problem can help, but do not solve it entirely.

***Index Terms*—TCP Incast; Modeling; Timeout; Window Size Distribution; Solutions.**

I. Introduction

As a prosperous industry, modern large-scale data center is developing quickly to enable cloud computing and bolster online services. In these online data intensive applications, the divide and conquer method is widely used in typical computing paradigms, such as MapReduce [7], Spark [18], Dryad [8], CIEL [11] and TritonSort [15], where many-to-one traffic pattern is common. For example, most of today's web search applications follow the partition/aggregation design pattern [4]. One big task is divided into pieces and assigned to many workers. Then the responses are aggregated to generate the final result. However, data simultaneously sent by many senders during the aggregation period is likely to overwhelm the switch buffer and make TCP's loss recovery mechanism lose efficacy. Then abnormal timeouts are triggered to result in catastrophic throughput collapse, which becomes lower than the link capacity by one or even two orders of magnitude. The completion time of delay-sensitive short flows is considerably extended, so the response misses the aggregator's deadline and is excluded from the final result. These special issues in Incast traffic are called TCP Incast problem, which degrades the network performance and then worsen user experience.

Since Incast was first termed in [12], lots of trails have been made. Early researches focus on modeling and analyzing. Jiao Zhang *et al.* attribute TCP Incast throughput collapse to two kinds of timeouts and provide detailed throughput estimation of a certain TCP version [19]. [6] presents a simple model to predict the throughput by fitting the experimental data instead of theoretical analysis.

On the other hand, some researches focus on solving TCP Incast problem. They can be classified into four categories roughly. (1) Adjusting system parameters: several trials have been made by changing the synchronized block size, enlarging the switch buffer size, or just increasing the capacity [14]. (2) Designing enhanced mechanisms: to reduce bandwidth wastage during timeout period, [16] reduces minimum Retransmission Timeout ($RTOmin$) to microsecond granularity. (3) Replacing loss-based TCP version: some approaches believe that delay-based algorithms improve the behavior of Incast traffic. (4) Designing new transmission protocols: ICTCP [17] and DCTCP [4] are two typical instances customized for data center network to solve Incast problem as well as to meet other additional requirements, such as low latency.

To the best of our knowledge, although there are lots of investigations both in analyzing and solving TCP Incast problem, there is little work attempting to close the gap between "analyzing" and "solving". The existing theoretical analysis either focuses on modeling the scenario of one special TCP version in Incast traffic, or is based on experimental data. They can provide some insights into understanding TCP Incast problem. However, these models are either closely coupled with particular protocol version or dependent on empirical observations, so a panoramic view is hardly captured. On the other hand, most solutions are proposed as heuristic designs. The negative impact of TCP Incast problem on performance can be constrained at some extent, but cannot be eliminated because the underlying issues may not be identified completely.

In this paper, we intend to build a simple interpretive model under proper assumptions to provide a comprehensive understanding of TCP Incast problem. The interpretive model emphasizes particularly on describing qualitatively how various factors, including system parameters and mechanism variables, affect network performances in Incast traffic pattern, but not on calculating the accurate throughput. Through observing lots of trace data in Incast scenarios, we verify that timeout is the root

978-1-4799-8381-0/15/$31.00 ©2015 IEEE

cause of TCP Incast problem, and the full window loss is the chief criminal. Then an interpretive model is built combining with experimental observation and qualitative analysis. We also redo the experiments to confirm that the model can give plausible explanations why the various solutions for TCP Incast problem can help, but do not solve it entirely.

The major contributions in this paper are threefold:

• Building a simple interpretive model to analyze the influence of system parameters and mechanism variables in flow control.

• Providing qualitative description of the impact of various factors in Incast traffic performance.

• Explaining why some solutions for TCP Incast problem work but hardly solve it entirely. The experiment results confirm our analysis.

The rest of the paper is organized as follows. The prior analysis work and solutions for TCP Incast problem are summarized in Section II. In Section III, we introduce the experimental setup and main assumptions. An interpretive model is built in Section IV, which considers all the key system parameters and mechanism variables in congestion control. In Section V, we conduct experiments to reproduce some typical solutions for Incast problem and analyze their performances. Finally the paper is concluded in Section VI.

II. Related Work

A. TCP Incast problem

Generally speaking, TCP Incast problem refers to TCP performance degradation when it runs in many-to-one traffic pattern. Initially, the catastrophic throughput collapse is observed when relatively great number of senders concurrently transmit data blocks to a single receiver, and any sender is not allowed to send the next data block until all the senders finish transmitting the current blocks [12]. Subsequently, in the delay-sensitive online applications which employ partition/aggregation computing paradigm, very few senders seriously protract the tail of flow completion time, so the system responds sluggishly or some responses are excluded from the final result due to missing the aggregator's deadline, which will impose negative impact on user experience [4].

These performance issues are related to TCP Incast problem. Seemingly, throughput and delay are different performance metrics. However, throughput collapse and flow completion time extension can both be attributed to the ill-suited mechanism in TCP flow control scheme, namely, abnormal timeouts are triggered. Generally, low throughput implies long completion time. Therefore, without special statement, the throughput is regarded as an iconic performance metric for understanding TCP Incast problem in this work.

B. Modeling

Some efforts have been made to analyze the throughput collapse phenomenon of TCP Incast by modeling.

Analytical Model: Jiao Zhang *et al*. summarize that the throughput collapse in TCP Incast is mainly caused by Block Head TimeOut (BHTO) and Block Tail TimeOut (BTTO) [19]. A goodput model of TCP Incast is built which describes the causes of the two kinds of timeouts. The model is validated by comparing with simulation data, and it can describe the features of Incast well under TCP NewReno.

Fitting Model: [6] uses empirical data to reason the dynamic system of TCP Incast. It proposes an analytical model estimating the throughput functions by fitting the experimental data. The model well explains some of the observed trends.

C. Solution

Except for modeling TCP Incast throughput, most of the investigations target to solve TCP Incast problem.

Adjusting System Parameters: Incast throughput collapse is a relatively complex case in which many system parameters are involved. Some attempts are made to alleviate TCP Incast problem by adjusting parameters, such as changing block size, enlarging buffer size, and increasing capacity [14]. Another typical example is that Facebook's researchers propose limiting the number of concurrent flows to alleviate TCP Incast problem [13].

Designing Enhanced Mechanisms: V. Vasudevan *et al.* claim that some timeouts in Incast traffic pattern are hard to be eliminated without extra mechanisms, so they suggest reducing $RTOmin$ to alleviate the throughput collapse [16]. Also as a simple and cost-effective enhanced mechanism, shrinking Maximum Transmission Unit (MTU) is proposed in [21] to mitigate TCP Incast problem.

Replacing loss-based TCP version: Some researches are interested in how the delay-based TCP algorithms behave in Incast scenario. Delay-based algorithms like Vegas [5] and FAST [14] can adjust the congestion window (*cwnd*) properly according to the delay information generated by RTT measurement. They think the slight adjustment of window sizes is beneficial to protect the buffer from overwhelming.

Designing New Transmission Protocols: ICTCP is proposed to solve Incast problem by adjusting the advertised window (*awnd*) at the receiver side by estimating the available bandwidth and RTT [17]. DCTCP employs Explicit Congestion Notification (ECN) and uses a weighted-average metric from the multi-bit information of ECN to adjust *cwnd* [4]. It can effectively avoid packet losses thus reducing the probability of timeout.

The related theoretical analysis work provides insights of Incast problem, but they either focus on modeling the scenario of one special TCP version, or depend on experimental data in special network environment. Various solutions exhibit different visions of Incast problem and provide performance gain. However, the work combining comprehensive understanding with explanations to the solutions is still absent. Our work is to fill the gap between analysis and solutions by providing an interpretive model which considers all the key system parameters and mechanism variables, and can also give plausible explanation to the existing solutions.

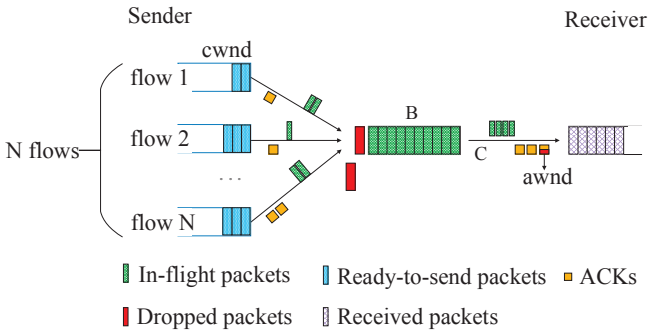

Fig. 1: Topology and parameter configurations.

III. Environment and Assumptions

A. Experimental Setup

To demonstrate the details of Incast problem in real-world environment, we build up a testbed and use tcpprobe [3] as the measurement tool. The basic topology of our testbed is made up of one switch and four end hosts. The switch function is realized by NetFPGA cards with $1Gbps$ Ethernet port, and the queue management scheme is Drop-Tail. Three of the four hosts act as servers, and the other one acts as the client. The propagation RTT without queuing delay is approximately $100\mu s$. In our experiment, each packet is $1.5KB$. Each end host can run multi-thread applications, and one thread represents one flow. The flows are distributed evenly among the three end hosts. Therefore, the number of active flows can be very large without the limitation of the number of machines.

B. Assumption

First of all, we should introduce two important concepts. One is *transmission*. One transmission begins from the senders starting to transmit new data blocks, and ends at completing this new task. The other is *round*. The first round starts from a congestion avoidance period of TCP evolution and lasts one RTT. The next round starts from the end of last round and lasts one RTT.

To abstract the model from real environment, we also need some reasonable assumptions. Assume that the packets will be dropped only when the bottleneck buffer overflows. All the packets will be dropped with equal probability. What's more, we have two assumptions which will be deeply analyzed and validated in Section IV: (1) Full window loss is the major kind of timeout that causes TCP Incast problem; (2) The distribution of window sizes in Incast traffic pattern is normal distribution.

The topology and parameter configurations in TCP Incast scenario are shown in Fig. 1. N senders transmit data blocks with size of S to the receiver. The capacity of the bottleneck is C packets per second. The buffer size is B packets. Each sender maintains its own *cwnd*, and the ACKs will bring back the *awnd* information from the receiver. Then the sending window size for the next round is the smaller one of *cwnd* and *awnd*. The key notations are summarized in Table I for the sake of terseness.

TABLE I: Key notations.

Not.	Description
x	Window size of a given flow in packets
$\bar{w}$	Average window size in packets
σ_w	Standard deviation of the window sizes in packets
p	Probability of packet loss
N	Number of flows
S	Block size in packets
C	Link capacity in packets per second
D	The Round-Trip Time in seconds
B	Buffer size in packets
R	Number of rounds in one transmission
Th	Throughput
tr	Transmission time of one flow
E_{Tr}	Expected duration of transmission
E_{TO}	Expected duration of timeout
P_{TO}	Probability of timeout
τ_{TO}	Duration of timeout

IV. Interpretive Model

The analytical model in [19] or the fitting model in [6] can accurately estimate TCP Incast throughput, and then can also provide some insights in identifying the radical reasons for TCP Incast problem. However, these models are either only applicable for special TCP version (such as NewReno in [19]), or built on experimental data in special network configurations [6]. Because the important mechanism in end-to-end congestion control and some key system parameters are not explicitly involved in the models, it is hard to get a comprehensive understanding of TCP Incast problem. In light of this, in this work, we intend to construct a concise interpretive model to capture the dominant factors in TCP Incast problem.

A. Throughput

The TCP Incast throughput collapse is mainly caused by timeouts. Through observing lots of trace data, we find that all the timeouts can be classified into two categories. One is full window loss, which occurs when an entire window is lost and there is no feedback ACK, so TCP's Fast Retransmission/Fast Recovery mechanism fails. The other happens when there are not enough ACKs to trigger retransmission. We call the former Full Loss Timeout (FLT) and the latter Lack ACK Timout (LAT).

Using the concepts of *transmission* and *round* defined in Subsection III-B, three typical scenarios illustrated in Fig. 2 are likely to appear during transmitting one data block in Incast communication pattern. Fig. 2(a) presents a normal situation, namely the flow takes a short time interval tr_1 to finish its transmission consisting of several rounds without any timeouts. Fig. 2(b) and (c) illustrate two abnormal cases, where timeouts are triggered. Fig. 2(b) encounters FLT from the start

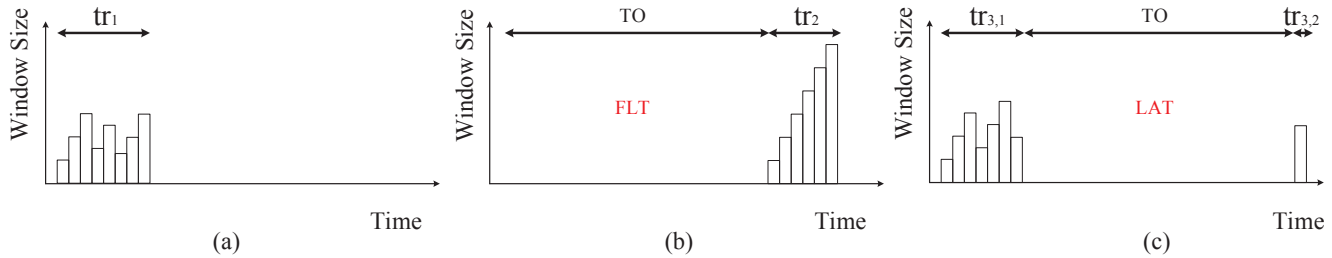

Fig. 2: The evolution of window sizes in the same Incast scenario. (a) is the normal situation that the flow finishes transmission within tr_1 time. (b) experiences FLT, so the total completion time includes TO and tr_2. (c) is encountered with LAT and its transmission time includes $tr_{3,1}$ and $tr_{3,2}$.

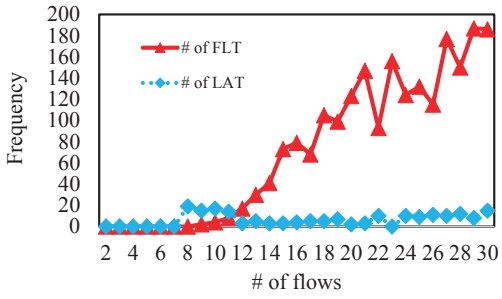

Fig. 3: Classification of timeouts.

of transmission, thus the completion time consists of TO and tr_2. The timeout shown in Fig. 2(c) is owing to LAT. The flow loses part of the block and has to retransmit the rest so that its transmission time includes two parts, $tr_{3,1}$ and $tr_{3,2}$.

Now we consider the scenario that N senders transmit data blocks with size S to the same receiver synchronously. The total amount of transmission data is $N \cdot S$. The total completion time is consisted of two parts: the time duration when link is busy, and when link is idle. Let E_{Tr} represent the expectation of the duration that the link is transmitting data, and E_{TO} as the expectation of the duration that the link is idle. Thus the throughput can be presented as follows:

$$Th \propto \frac{N \cdot S}{E_{Tr} + E_{TO}} \tag{1}$$

From Fig. 2, we can find that Tr is the union of all the transmission time of N flows:

$$Tr = \bigcup_{0<i\leq N} \bigcup_{j>0} tr_{i,j} \tag{2}$$

where i represents the $i-th$ flow, and j is the $j-th$ transmission period of the $i-th$ flow.

It is hard to calculate the precise value of Tr. However, we assume that the link capacity is fully utilized when transmitting data. Thus, the expectation of transmission time can be approximated:

$$E_{Tr} \approx \frac{N \cdot S}{C} \tag{3}$$

For the idle time, timeout appears. Although many flows may encounter timeouts, their waste time is highly overlapped. So we can approximately take one TO as the waste time in one transmission. Thus the expectation of the idle time is the probability of timeout P_{TO}, multiplied by the duration of timeout τ_{TO}:

$$E_{TO} \approx P_{TO} \cdot \tau_{TO} \tag{4}$$

Hence, the expectation of throughput can be presented as:

$$Th \propto \frac{N \cdot S}{N \cdot S/C + P_{TO} \cdot \tau_{TO}} \tag{5}$$

It is obvious that all the parameters except P_{TO} are easy to be substituted into (5). Our next work is to predict the probability of timeout. Fig. 3 presents the number of FLT and LAT as the number of concurrent flows increases. The statistical result shows that more than 90% of timeouts are FLT after 15 flows. Hence, when we predict the probability of timeout, FLT is our first concern.

B. Window Size Distribution

When studying the details of timeout, we find that most sacrificed flows in FLT are with small windows. We take 15 flows of NewReno as an example, and record the window size of each flow when timeout happens. As shown in Fig. 4, in most rounds that encounter timeouts, the loss of the whole small window causes FLT. Actually, if the window size distribution at the start of one round is polarized, the small windows are more easily squeezed out by the large ones.

To conclude the regular pattern of window size distribution, we monitor the window size of each flow every 0.1s (except the timeout duration) in ns-2 simulation platform [1]. The statistic result of window sizes is shown in Fig. 5(a). The shape of the frequency curve follows normal distribution. At 30 flows, the window size distribution also looks like normal distribution as shown in Fig. 5(b).

Then we enlarge the range from 15 to 30 flows, and check whether the normal distribution is a reasonable assumption using SPSS [2]. The input data is a collection of window sizes of all the flows at the same time. Table II shows the analysis result using Kolmogorov-Smirnov test [9], which is the principal goodness of fit test for normal and uniform data sets. In this example, the null hypothesis is that the data is normally distributed. In most cases, the values of "Sig"

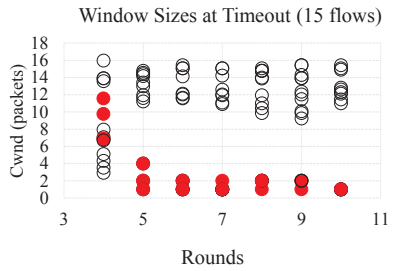

Fig. 4: Window sizes at timeout. The red solid circles represent the flows losing the whole window packets, and the hollow circles represent those not losing the whole window.

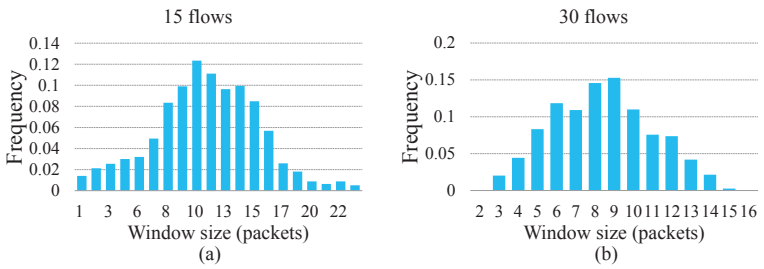

Fig. 5: Window size distributions of 15 and 30 flows in TCP NewReno.

TABLE II: The result of Kolmogorov-Smirnov test.

Num. of flows	16	17	18	19	20
Asymp. Sig.	0.241	0.306	0.436	0.157	0.054
Num. of flows	21	22	23	24	25
Asymp. Sig.	0.000	0.149	0.012	0.079	0.262
Num. of flows	26	27	28	29	30
Asymp. Sig.	0.706	0.136	0.173	0.084	0.561

columns are above 0.05, which means the null hypothesis is not rejected. Thus we can accept the assumption that the window size distribution at a certain time is normal distribution. The probability density function of normal distribution is:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{6}$$

where μ is the average and σ is the standard deviation.

C. Probability of Timeout

After describing the window size distribution in reasonable functions, we can calculate the probability of timeout considering the characteristics. Assume FLT is the dominant timeout, we have the following lemma:

Lemma 1. *In Incast scenario, the probability of timeout in one round P_{FLT}^k is proportional to $e^{\bar{w}lnp+\frac{1}{2}\sigma_w^2(lnp)^2}$, where $\bar{w}$ is the average window size at the start of one round, σ_w is the standard deviation, and p represents the probability of packet loss.*

Proof:

1) FLT of one flow: At certain time, the $i-th$ flow of $k-th$ round with x window size has the probability of:

$$p_i^k(x) = \frac{1}{\sqrt{2\pi}\sigma_w} e^{-\frac{(x-\bar{w})^2}{2\sigma_w^2}} \tag{7}$$

FLT indicates that the packets in a whole window are lost. So, for the flow with sending window size x, the probability of FLT is:

$$p_{i-FLT}^k(x) = p_i^k(x) \cdot p^x \tag{8}$$

2) FLT of one round: Any one of the flows which loses the whole window will trigger FLT in one round. Thus the probability of FLT of the $k-th$ round is:

$$\begin{aligned} P_{FLT}^k &\approx \int_0^{+\infty} p_{i-FLT}(x)dx \\ &= \int_0^{+\infty} \frac{1}{\sqrt{2\pi}\sigma_w} e^{-\frac{(x-\bar{w})^2}{2\sigma_w^2}} p^x dx \end{aligned} \tag{9}$$

Let $t = \frac{x-\bar{w}}{\sigma_w}$, then

$$\begin{aligned} P_{FLT}^k &\approx \int_{-\frac{\bar{w}}{\sigma_w}}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma_w} e^{-\frac{t^2}{2}} e^{(\sigma_w t+\bar{w})lnp} d(\sigma_w t+\bar{w}) \\ &= \frac{1}{\sqrt{2\pi}} \int_{-\frac{\bar{w}}{\sigma_w}}^{+\infty} e^{-\frac{1}{2}(t-\sigma_w \cdot lnp)^2+\bar{w}lnp+\frac{1}{2}\sigma_w^2(lnp)^2} dt \\ &= \frac{1}{\sqrt{2\pi}} e^{\bar{w}lnp+\frac{1}{2}\sigma_w^2(lnp)^2} \int_{-\frac{\bar{w}}{\sigma_w}}^{+\infty} e^{-\frac{1}{2}(t-\sigma_w \cdot lnp)^2} dt \end{aligned} \tag{10}$$

Define the error function $f_e(x)$ as:

$$f_e(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \tag{11}$$

And the relationship between error function and normal distribution is:

$$\int \frac{1}{\sigma_w\sqrt{2\pi}} e^{-\frac{(x-\bar{w})^2}{2\sigma_w^2}} dx = \frac{1}{2\sigma_w}[1 + f_e(\frac{x-\bar{w}}{\sqrt{2}\sigma_w})] + C_0 \tag{12}$$

where C_0 is a constant.

Then Eq. (10) can be rewritten as:

$$\begin{aligned} P_{FLT}^k &\approx \frac{1}{2} e^{\bar{w}lnp+\frac{1}{2}\sigma_w^2(lnp)^2} \cdot [1 + f_e(\frac{x-\sigma_w \cdot lnp}{\sqrt{2}})]|_{-\frac{\bar{w}}{\sigma_w}}^{+\infty} \\ &= \frac{1}{2} e^{\bar{w}lnp+\frac{1}{2}\sigma_w^2(lnp)^2} [1 + f_e(\frac{\bar{w}+\sigma_w^2 \cdot lnp}{\sqrt{2}\sigma_w})] \end{aligned} \tag{13}$$

Since $-1 \le f_e(x) \le 1$, we can get the upper bound of P_{FLT}^k:

$$P_{FLT}^k \le e^{\bar{w}lnp+\frac{1}{2}\sigma_w^2(lnp)^2} \tag{14}$$

In Incast traffic pattern, we can assume that the probability

of packet loss is not too small. When $lnp \geq -\frac{\bar{w}}{\sigma_w^2}$, we can have the following inequation:

$$f_e(\frac{\bar{w}+\sigma_w^2 \cdot lnp}{\sqrt{2}\sigma_w}) \geq 0$$

Thus we can also get the lower bound of P_{FLT}^k:

$$\begin{aligned} P_{FLT}^k &\approx \frac{1}{2}e^{\bar{w}lnp+\frac{1}{2}\sigma_w^2(lnp)^2}[1+f_e(\frac{\bar{w}+\sigma_w^2 \cdot lnp}{\sqrt{2}\sigma_w})] \\ &\geq \frac{1}{2}e^{\bar{w}lnp+\frac{1}{2}\sigma_w^2(lnp)^2} \end{aligned} \quad (15)$$

Combining (14) and (15) we have:

$$\frac{1}{2}e^{\bar{w}lnp+\frac{1}{2}\sigma_w^2(lnp)^2} \leq P_{FLT}^k \leq e^{\bar{w}lnp+\frac{1}{2}\sigma_w^2(lnp)^2}$$

Namely,

$$P_{FLT}^k \propto e^{\bar{w}lnp+\frac{1}{2}\sigma_w^2(lnp)^2} \quad (16)$$

Obviously, if the network congestion is so severe that $p \to 1$, then $lnp \to 0$ and $P_{FLT} \to 1$.

■

Lemma 2. *In Incast scenario, if each round has the same probability of timeout, then the probability of timeout in one transmission is $P_{TO} = 1-(1-P_{FLT}^k)^R$, where R is the number of rounds.*

Proof:

Assume one transmission has R rounds. The transmission will be and only be successful when all the rounds have no FLT. If the $k-th$ round has the probability of FLT as P_{FLT}^k, and we only consider the impact of FLT, the probability that one transmission will suffer timeout is:

$$P_{TO} = 1 - \prod_{1 \leq k \leq R}(1-P_{FLT}^k) \quad (17)$$

If every round has the same probability P_{FLT}^k, then P_{TO} can be simply presented as $1-(1-P_{FLT}^k)^R$. ■

D. Theoretical Analysis

From *Lemma 1*, we can conclude that, decreasing $\bar{w}$, σ_w and lnp can decrease P_{FLT}. When calculating p, we have an interesting proposition as below:

Proposition 1. *In Incast scenario, if the other parameters (C, S, R, D, p, $\bar{w}$, σ_w) are kept unchanged, then N is linear with B before the throughput collapse.*

Proof: Because the average window size is $\bar{w}$, $N \cdot \bar{w}$ packets will be injected into the network in one round. $C \cdot D$ packets will be served by the bottleneck link. Assume the number of arriving packets at the switch exceeds the buffer size, then $N \cdot \bar{w} - B - C \cdot D$ packets will be equally dropped. Hence, the probability of packet loss can be presented as:

$$p = \frac{N \cdot \bar{w} - B - C \cdot D}{N \cdot \bar{w}} \quad (18)$$

If $B \gg C \cdot D$, we'll get:

$$p \approx 1 - \frac{B}{N \cdot \bar{w}} \quad (19)$$

Then N is linear with B. ■

Remark: In data center networks, the characteristic of the pipeline is $B \gg C \cdot D$. For example, in a $1Gbps$ link capacity network, if $D = 100\mu s$, then $C \cdot D = 12.5KB$. However, the buffer size is often hundreds of KB or even in the unit of MB. Hence, we can approximately conclude that N is linear with B. In literature [14], A. Phanishayee also mentioned this phenomenon, namely, doubling the size of the switch's output port buffer doubles the number of servers that can transmit before Incast.

Next, we analyze the impact of $\bar{w}$ and σ_w in (16). The function actually contains two components:

$$e^{\bar{w}lnp+\frac{1}{2}\sigma_w^2(lnp)^2} = e^{\bar{w}lnp} \cdot e^{\frac{1}{2}\sigma_w^2(lnp)^2} \quad (20)$$

Since $0 < p < 1$, $lnp < 0$. Hence, $e^{\bar{w}lnp}$ is smaller than 1, and $e^{\frac{1}{2}\sigma_w^2(lnp)^2}$ is larger than 1. We define the former as the "*negative component*" and the latter "*positive component*". The increase of $\bar{w}$ and σ_w will increase the *negative component* and the *positive component* separately.

Proposition 2. *In Incast scenario, keep the other parameters (C, S, R, D, B) unchanged. When N is small, a little increase of σ_w will enlarge P_{FLT} dramatically. As N increases, $\bar{w}$ becomes the dominant factor of P_{FLT}. When N increases to certain extent that $\bar{w} \gg B/N$, P_{FLT} is irrelated with σ_w and $\bar{w}$ and has the largest value as $e^{-\frac{B}{N}}$.*

Proof: When N is not so large, p is away from 1, which makes lnp far away from 0. Then P_{FLT} is mainly influenced by its *positive component*. Because σ_w appears as the quadratic term of the exponent, P_{FLT} will increase dramatically even if σ_w increases a little. As N increases, lnp tends to 0, then we can ignore the higher order terms of lnp. So the *negative component* has greater impact on the value of P_{FLT}. As N continues to grow, B/N becomes more and more closer to 0. When $\bar{w} \gg B/N$, $\frac{B}{N\bar{w}} \to 0$, we can get the Taylor series of the exponent of *negative component* as:

$$\begin{aligned} \bar{w}lnp &\approx \bar{w}ln(1-\frac{B}{N\bar{w}}) \\ &= \bar{w}[-\frac{B}{N\bar{w}} - \frac{1}{2}(\frac{B}{N\bar{w}})^2 - \frac{1}{3}(\frac{B}{N\bar{w}})^3 - \cdots] \\ &\approx -\frac{B}{N} \end{aligned} \quad (21)$$

Ignoring the impact of *positive component*, P_{FLT} has the largest value as $e^{-\frac{B}{N}}$. ■

Remark: The influence of window size distribution with the increase of N can be divided into three stages. First, when N is small, the *positive component* is the dominant component. σ_w has great impact on the system throughput. Second, as N increases, $\bar{w}$ becomes more important and the *negative component* dominates P_{FLT}. Third, when N increases to certain extent, P_{FLT} is independent with $\bar{w}$ and

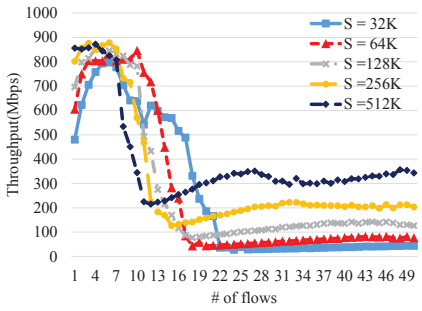

Fig. 6: The throughput of different block sizes.

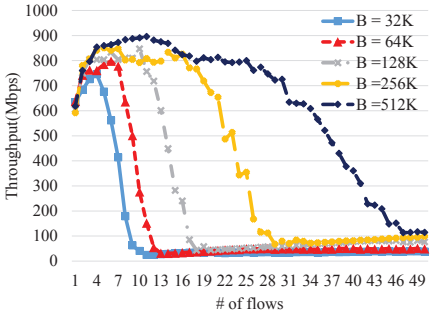

Fig. 7: The throughput of different buffer sizes.

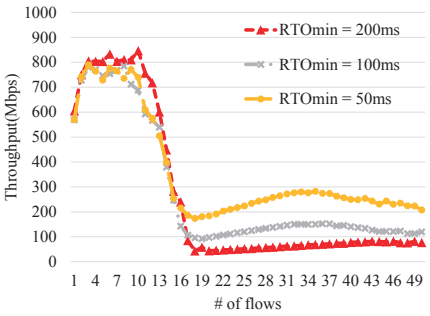

Fig. 8: The throughput of different $RTOmin$s.

σ_w, which means the window size distribution is irrelated with the throughput.

As aforementioned, in data center network, $B \gg C \cdot D$. The ideal average window size of each flow approximately equals to B/N, which means N flows share the buffer size averagely. Actually, when no Incast problem happens, the window size of each flow will fluctuate up and down the average line slightly in the steady state. However, if N is too large that $\bar{w} \gg B/N$, where $\bar{w}$ is the adjustment result of congestion control mechanism and B/N is the average value at steady state, the injecting packets will severely exceed the network capacity. When B/N is close to or smaller than 1, even if every flow injects one packet into the network, the buffer size at the bottleneck will still be overwhelmed and FLT is triggered.

Hence, our model of throughput considers all the impact of key system parameters and mechanism variables. System parameters include link capacity C, block size S and buffer size B. Mechanism variables include $RTOmin$, average window size $\bar{w}$ and standard deviation of window sizes σ_w. In the next section, considering these parameters, we'll discuss their qualitative impact on the throughput.

V. Solution Analysis

In this section, we analyze the impact of key system parameters and mechanism variables. We also reproduce several well-known solutions for TCP Incast problem and analyze how they will alleviate the throughput collapse. When we analyze a certain parameter or solution, other parameters are kept unchanged. The default parameters in our experiments are: $C = 1Gbps$, $B = 128KB$, $S = 64KB$, $RTOmin = 200ms$, and the default TCP version is NewReno. We reproduce each solution 100 times and get the average throughput as the final result.

A. Adjusting System Parameters

1) Capacity: An easy way to improve throughput under Incast scenario is to increase the link capacity. Increased link capacity implies an increase in potential packet processing rate. First, the transmission time $E_{Tr} \approx N \cdot S/C$ will be reduced. Second, increasing capacity will decrease the probability of packet loss. We can rewrite Eq. (18) as follows:

$$p = 1 - \frac{B + C \cdot D}{N \cdot \bar{w}} \tag{22}$$

Obviously, larger C results in smaller p, thus larger throughput.

2) Block Size: (5) can be rewritten as:

$$Th \propto C - \frac{C^2 \cdot E_{TO}}{N \cdot S + C \cdot E_{TO}} \tag{23}$$

When other parameters are kept unchanged, larger synchronized block size results in larger throughput.

Changing block size as a solution is proposed in [14]. In Fig. 6, after the throughput collapse, the block size of $512KB$ has the largest throughput. However, S only enlarges throughput after the collapse, it cannot avoid timeout. E_{TO} still occupies a big part of the denominator in (1), and strongly imposes negative impact on the throughput.

A particular phenomenon in Fig. 6 is that the smaller the block size, the later the throughput collapse happens. $S = 32KB$ is the latest to have an obvious throughput collapse. This seems to violate our analysis that the larger block size, the larger throughput. However, when block size increases, the number of rounds R may also increase. According to Eq. (17), increasing R means increasing the risk of full window loss in a whole transmission process. Because $32KB$ is so small that only 3 or 4 rounds of transmission will complete the task, it has the lowest risk.

3) Buffer Size: We implement Incast experiment with different buffer sizes. The results are shown in Fig. 7. An obvious tendency in Fig. 7 is that larger buffer size will support more concurrent flows. Although not so precisely, we can still observe a rough tendency that, when the buffer size doubles, the number of the supported concurrent flows also doubles. Another method to mitigate TCP Incast mentioned in [21] is to shrink MTU size. Actually, because the unit of buffer size in our model is in packets, the buffer will contain more packets with smaller MTU. Halving MTU has almost the same effect of doubling buffer.

B. Designing Enhanced Mechanisms

$RTOmin$ is the minimum time that TCP will experience under abnormal packet loss. Reducing $RTOmin$ is actually

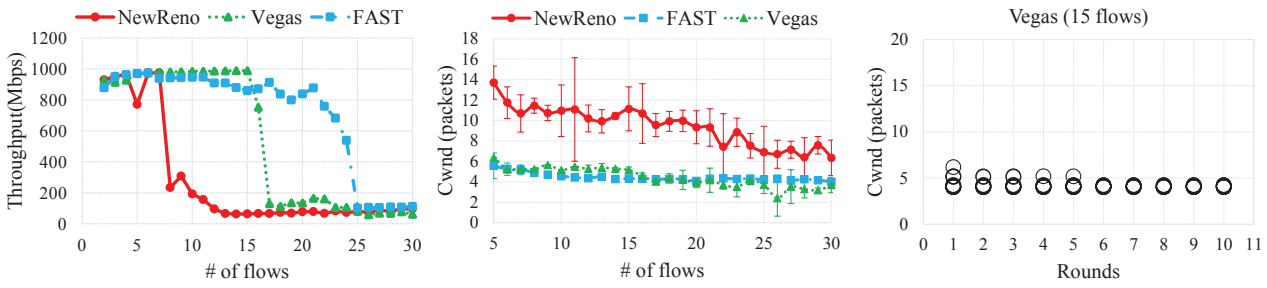

Fig. 9: (a) Throughput of NewReno, Vegas and FAST in simulation. (b) The average and standard deviation of the window sizes of NewReno, Vegas and FAST, where the dots connected with horizontal lines represent the averages and the vertical lines represent the standard deviations. (c) An illustration of window size distribution of Vegas at 15 flows. The circles represent the window sizes at the start of each round.

reducing τ_{TO} in (4). Although P_{TO} may not change, if τ_{TO} drops heavily by several orders of magnitude, the throughput collapse is alleviated. Typically, $RTOmin$ is set to $200ms$. V. Vasudevan *et al.* suggest reducing $RTOmin$ to microsecond granularity to alleviate the throughput collapse [16].

We test three values of $RTOmin$ as $50ms$, $100ms$ and $200ms$. The results are shown in Fig. 8. $RTOmin$ with $50ms$ seems to have the largest throughput. However, all the three situations have the same point of throughput collapse, which means they almost have Incast problem at the same value of N. Smaller $RTOmin$ only helps to diminish the bandwidth wastage during timeout periods, but timeout still occurs. There is concern with this method, though: setting $RTOmin$ too small may result in spurious retransmissions [20].

C. Replacing loss-based TCP version

Delay-based algorithms bring new insights into congestion control that we can sense the network congestion not only based on the packet loss, but also on the delay. Vegas [5] provides TCP the ability to anticipate congestion and adjust its transmission rate according to the information generated by RTT measurements. FAST [10] is considered to inherit the advantages of Vegas and make further improvements.

We simulate NewReno, Vegas and FAST via ns-2. The link capacity is $1Gbps$, RTT is $100\mu s$ and the block size is $100KB$. We set the buffer size to $64KB$, and each packet is $1KB$. Fig. 9(a) demonstrates that Vegas and FAST outperform NewReno. The average and standard deviation of the window sizes are shown in Fig. 9(b). NewReno has larger average and standard deviation than those of Vegas and FAST. Fig. 9(c) is an illustration of window size distribution of Vegas at 15 flows. Compared to NewReno in Fig. 4, after several rounds of adjustment, the window sizes almost converge to the same value. According to *Proposition 2*, a little increase of σ_w will increase P_{FLT} dramatically. Thus the larger σ_w of NewReno causes earlier throughput collapse compared to Vegas and FAST. Besides, NewReno has the largest $\bar{w}$, which is also the dominant factor to increase P_{FLT}.

The different σ_ws of these three TCP versions are caused by their different mechanisms at the congestion avoidance phase. Additive Increase Multiplicative Decrease (AIMD) is the congestion avoidance mechanism for NewReno. The window size will be increased by 1 if no packet loss is detected, or be halved if packet loses. Instead of simply increasing the window, Vegas has slight window size adjustment when no packet loss is detected. The window size can be increased by 1, decreased by 1, or just kept unchanged, according to the difference between the expected and measured throughput. Similar to Vegas, FAST also employs a mild adjustment law for window size. Compared to aggressive AIMD, the mild adjustment tends to result in converged window size distribution. Because of the smaller σ_w and $\bar{w}$, Vegas and FAST are less likely to have crazy increasing windows which may crowd out other small windows.

D. Designing New Transmission Protocols

Several recently proposed transmission protocols can avoid TCP Incast problem by designing new algorithms to suit the characteristic of data centers.

ICTCP [17] focuses on Incast scenario that the last hop is the bottleneck. It prevents timeout by dynamically adjusting the *awnd* at the receiver side. The feedback *awnd* will adjust the window size at the sender side to a fair value. ICTCP uses the similar concept of Vegas that it also adopts two thresholds to differentiate three cases for *awnd* adjustment. Thus the window size distribution of ICTCP also appears in small σ_w and $\bar{w}$, which results in small probability of timeout.

M. Alizadeh *et al.* propose DCTCP [4] to achieve high throughput while maintaining low latency. DCTCP sets up one threshold K at the switch and employs ECN to reflect the congestion of the network. Then each sender generates one-bit information from ECN feedback and form multi-bit information represented as a weighted-average metric α. The window size for the next round is calculated according to $cwnd \rightarrow cwnd(1-\alpha/2)$, where $0 < \alpha < 1$.

In our experiment, we set $K = 20$ packets of DCTCP as recommended. We compare the performances of TCP NewReno and DCTCP in Fig. 10(a). DCTCP can sustain much more concurrent flows than NewReno. According to

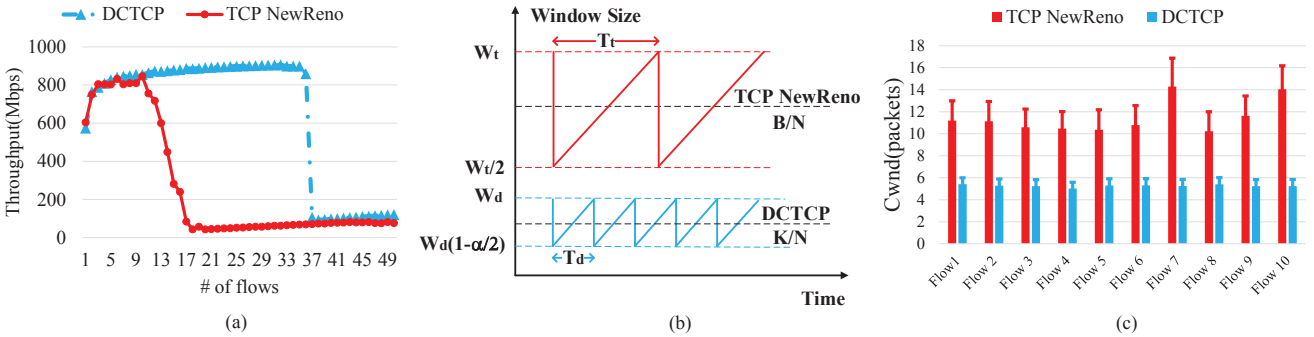

Fig. 10: (a) Throughput of NewReno and DCTCP. (b) The sawtooth waves of window size of DCTCP and TCP NewReno. (c) The average and standard deviation of the window sizes of the 10 flows.

the regulation mechanism, we draw the window size of a single DCTCP and TCP NewReno sender in Fig. 10(b). In TCP NewReno, the ideal average window size in the steady state is approximately B/N. Assume the maximum window size is W_t, the window size will fluctuate up and down the average line, while between W_t and $W_t/2$. However, because DCTCP sets K at the switch side to constrain the queue length, the ideal average window size at the steady state is approximately K/N. What's more, DCTCP calculates the window size according to $cwnd \rightarrow cwnd(1-\alpha/2)$, if the maximum value is W_d, then the amplitude of oscillation in window size is given by:

$$W_d - W_d(1-\frac{\alpha}{2}) = \frac{\alpha}{2} \cdot W_d \tag{24}$$

Since $\alpha < 1$, the window size fluctuates within a smaller range than TCP NewReno. Some theoretical analysis of the oscillation in DCTCP is conducted in [4]. It assumes α is small and can be simplified as $\alpha \approx \sqrt{2/(W_d-1)}$. Thus the amplitude of oscillation approximately equals to $\frac{1}{2}\sqrt{2(C \cdot D + K)/N}$.

We take 10 flows as an example, and calculate the average and standard deviation of the window size of each flow in Fig. 10(c). Because $K < N$, the average window sizes of DCTCP are smaller than those of TCP NewReno, and all the 10 flows have nearly the same average. Since the amplitude of window size oscillation in DCTCP is much smaller than that of TCP NewReno, the window sizes in DCTCP have smaller standard deviations. As we discussed in *Proposition 2*, larger $\bar{w}$ and σ_w induce larger probability of timeout, thus DCTCP has better performance than TCP NewReno in Incast traffic pattern.

VI. Conclusion

In this paper, we provide an in-depth understanding of how TCP Incast problem happens. Based on experimental researches, an interpretive model is presented particularly focusing on how the key system parameters and mechanism variables affect the performance of flow control system in Incast traffic pattern. We fill the gap between prior "analyzing" and "solving" work by providing qualitative analysis, rather than calculating accurate throughput in certain environment. With this model, we reproduce several methods of solving TCP Incast and give plausible explanations on their performances.

VII. Acknowledgement

The authors gratefully acknowledge the anonymous reviewers for their constructive comments. This work is supported in part by Tsinghua-Cisco joint lab and National Natural Science Foundation of China (NSFC) under Grant No. 61225011.

References

[1] The network simulator ns-2. http://www.isi.edu/nsnam/ns/.
[2] Spss statistics desktop. http://www14.software.ibm.com/download/data/web/en_US/trialprograms/W110742E06714B29.html.
[3] Tcp probe. http://www.linuxfoundation.org/collaborate/workgroups/networking/tcpprobe.
[4] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). In *SIGCOMM*, 2010.
[5] L. S. Brakmo and L. L. Peterson. TCP Veags: end to end congestion avoidance on a globle Internet. In *IEEE Journal on Selected Areas in Communications, vol. 13, no. 8, pp. 1465-80*, October 1995.
[6] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D.Joseph. Understanding TCP Incast Throughput Collapse in Datacenter Networks. In *WRENM*, 2009.
[7] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, 2004.
[8] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. In *EuroSys*, March 2007.
[9] F. J. and M. Jr. The Kolmogorov-Smirnov Test for Goodness of Fit. In *Journal of the American Statistical Association, vol. 46, no. 253, pp. 68-78*, March 1951.
[10] C. Jin, D. X. Wei, and S. H. Low. FAST TCP: Motivation, Architecture, Algorithms, Performance. In *INFOCOM*, 2011.
[11] D. G. Murray, M. Schwarzkopf, C. Smowton, S. Smith, A. Madhavapeddy, and S. Hand. CIEL: A Universal Execution Engine for Distributed Data-Flow Computing. In *NSDI*, 2011.
[12] D. Nagle, D. Serenyi, and A. Matthews. The Panasas ActiveScale Storage Cluster: Delivering Scalable High Bandwidth Storage. In *ACM/IEEE Supercomputing*, 2004.
[13] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, and et al. Scaling Memcache at Facebook. In *USENIX NSDI*, 2013.
[14] 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 *USENIX FAST*, 2008.
[15] A. Rasmussen, G. Porter, M. Conley, H. Madhyastha, R. Mysore, A. Pucher, and A. Vahdat. Tritonsort: A Balanced Large-scale Sorting System. In *NSDI*, 2011.
[16] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson1, and B. Mueller. Safe and Effective Finegrained TCP Retransmissions for Datacenter Communication. In *SIGCOMM*, August 2009.
[17] H. Wu, Z. Feng, C. Guo, and Y. Zhang. ICTCP: Incast Congestion Control for TCP in Data Center Networks. In *CoNEXT*, 2010.
[18] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster Computing with Working Sets. In *HotCloud*, March 2010.
[19] J. Zhang, F. Ren, and C. Lin. Modeling and Understanding TCP Incast in Data Center Networks. In *INFOCOM*, 2011.
[20] J. Zhang, F. Ren, L. Tang, and C. Lin. Taming TCP Incast Throughput Collapse in Data Center Networks. In *ICNP*, 2013.
[21] P. Zhang, H. Wang, and S. Cheng. Shrinking MTU to Mitigate TCP Incast Throughput Collapse in Data Center Networks. In *Third International Conference on Communication and Mobile Computing*, 2011.