symmetry

MDPI

Article

Gentle Slow Start to Alleviate TCP Incast in Data Center Networks

Sheeba Memon [1], Jiawei Huang [1,*] and Hussain Saajid [2]

[1] School of Information Science and Engineering, Central South University, Changsha 410083, China; sheeba.memon@csu.edu.cn

[2] School of Computer Science and Engineering, Dalian University of Technology, Dalian 116024, China; sunny_sau@hotmail.com

* Correspondence: jiaweihuang@csu.edu.cn; Tel.: +86-0731-8883-0212

Received: 30 December 2018; Accepted: 25 January 2019; Published: 27 January 2019

Abstract: Modern data center networks typically adopt symmetric topologies, such as leaf-spine and fat-tree. When a large number of transmission control protocol (TCP) flows in data center networks send data to the same receiver, the congestion collapse, called TCP Incast, frequently happens because of the huge packet losses and Time-Out. To address the TCP Incast issue, we firstly demonstrate that adjusting the increasing speed of the congestion window during the slow start phase is crucially important. Then we propose the Gentle Slow Start (GSS) algorithm, which adjusts the congestion window according to real-time congestion state in a gentle manner and smoothly switches from slow start to congestion avoidance phase. Furthermore, we present the implementation and design of Gentle Slow Start and also integrate it into the state-of-the-art data center transport protocols. The test results show that GSS effectively decreases the Incast probability and increases the network goodput by average 8x.

Keywords: congestion control; Incast; slow start; data center networks; TCP

1. Introduction

Currently, data centers are becoming the essential infrastructure in the brand new era of cloud computing. With the ascent of distributed computing and storage, the network facility suppliers, such as Amazon, Google, and Microsoft, build their data centers to deal with online applications, including Hadoop, MapReduce, and Web searching. These applications generally apply a data extensive communication model, where the large amount of data is delivered among the distributed processing components [1,2].

The fundamental features of data center networks (DCNs) are a high-speed link (>1 Gbps) and low propagation delay (around 100 μs). Unfortunately, due to the limitation of TCP protocol, TCP is not fully appropriate for data center networks. When many concurrent TCP flows reach a switch, the switch buffer is exhausted in a small epoch of time, resulting in severe packet losses and even 200 ms idle time of retransmission Time-Out (RTO) [3,4]. This throughput collapse is called TCP Incast.

Many novel transport layer protocols are proposed to resolve the TCP Incast issue [3–7]. These protocols share similar architecture and features—in order to reduce the large number of packets in the bottleneck link, they reduce the TCP congestion windows when detecting congestion. These transport layer techniques alleviate the influence of the Incast problem in TCP. However, these TCP solutions have a common problem. All protocols just concentrate on the congestion avoidance phase, while ignoring the aggressively increasing sending rate in the slow start phase.

In data centers, most flows are small in size. For example, more than 80% of flows in data mining applications are usually small in length (e.g., 10 KB) [8]. It indicates that most flows are small enough

to finish their transmissions in the slow start phase. However, the slow start phase is very aggressive because of the congestion window's exponential growth [9]. In the slow start phase, the sender expands the congestion window by 1 for each Acknowledgement (ACK), which easily results in packet loss and even TCP Time-Out. In typical data center applications, if one flow experiences Time-Out under the high flow concurrency, the other flows have to wait for the unlucky flow to initiate the next round of transmission. Thus, the TCP Incast greatly degrades the overall performance in data center networks.

In this paper, we demonstrate that by only giving attention to the congestion avoidance phase, the current TCP protocols for data centers are not able to tackle the TCP Incast problem in the slow start phase. Through theoretical evaluation and empirical studies, we show that adjusting the increasing speed of congestion window in the slow start phase is vital in resolving the TCP Incast. We then propose the Gentle Slow Start (GSS) algorithm, which smoothly adjusts the congestion window in a gentle manner and effectively reduces the Incast probability. Specifically, our major contributions are given below:

(1) An extensive simulation-based study is conducted to explore the impact of TCP slow start on Incast problem. We experimentally and theoretically show why regulating the congestion window in slow start phase is more efficient in avoiding Incast under heavy congestion.
(2) We propose the Gentle Slow Start algorithm, which adjusts increasing speed of congestion window according to the real-time state of network congestion and smoothly switches slow start to congestion avoidance phase. Our design is very simple and can be easily deployed on various TCP protocols.
(3) We integrate Gentle Slow Start into the state-of-the-art data center transport protocols, such as DCTCP [5], D2TCP [10], and L2DCT [11]. The test results reveal that GSS substantially decreases the Incast probability for various TCP protocols. Gentle Slow Start greatly especially decreases the number of Time-Out events and helps DCTCP, TCP NewReno, D2TCP, and L2DCT to obtain up to 8x goodput.
(4) The rest of the paper is structured as follows. In Section 2, the motivation is described. The architecture of Gentle Slow Start is described in Section 3. In Section 4, through network simulator 2 (NS2) simulations the performance of Gentle Slow Start is evaluated. In Section 5, we review the related works. Finally, the paper is concluded in Section 6.

2. Motivation

In this section, to perceive the difficulties in dealing with data center transport protocols, we initially compare two phases of traditional TCP congestion window updating mechanisms. Secondly, we present the traffic characters in DCNs. Moreover, we illustrate that the Incast probability is successfully decreased by adjusting the congestion window in the slow start phase.

2.1. Congestion Avoidance and Slow Start

The mechanism of updating the TCP congestion window has two phases—congestion avoidance and slow start. Figure 1 presents the congestion avoidance and slow start phases. In the slow start phase, the algorithm works by increasing the TCP window by one segment for each Acknowledgement (ACK), meaning the TCP window size doubled for each Round Trip Time (*RTT*).

When congestion window (cwnd) has reached the threshold of slow start, the congestion avoidance phase starts. If all cwnd segments have been acknowledged in one *RTT*, cwnd is increased by one only. Compared with congestion avoidance, slow start is much more aggressive in increasing the congestion window. In the slow start phase, it is very easy to exhaust the available bandwidth leading to packet losses due to the exponential growth in the congestion window.

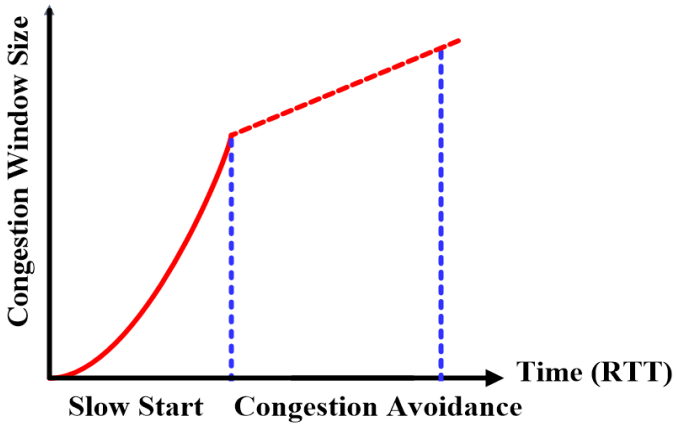


Figure 1. Example of congestion avoidance and slow start phase.

2.2. Traffic Characters in Data Center

In data centers, the network traffic shows the following three characters. Firstly, most flows are small in size (i.e., $\leq$100 KB) [1,8]. Some tiny flows are even smaller than 10 KB. This character means that the lifetime of most flows in data centers is dominated within their slow start phases.

Secondly, the mice flows always appear in the burst pattern. For example, the measurement results in [12,13] show that the network traffics in the Memcached systems exhibit obvious shortness and prevalence of bursts, easily bringing about heavy congestion for the switches with shallow buffer.

Thirdly, these mice flows also show the large-scale property, not only in numerous covered physical nodes but also in high concurrency. Given the Memcached clusters of Facebook as examples, these applications usually use tens of thousands of servers and more than 100,000 flows are activated per second [13].

2.3. Modeling Analysis of Slow Start

To examine the performance of highly concurrent TCP flows, we firstly analyze the influence of the congestion window w in the slow start phase and flow concurrency n on the Incast probability P. When n concurrent flows share the bottleneck link, we get the packet loss rate p as

$$p = 1 - \frac{nw}{B}, \tag{1}$$

where w and B are the congestion window size and buffer size, respectively. Since the packet loss of full congestion window is the main reason causing the Time-Out and TCP Incast problem [14,15], the Time-Out probability P_{TO} is calculated as the probability of full window of packet losses, that is

$$P_{TO} = p^w = (1 - \frac{nw}{B})^w. \tag{2}$$

As TCP Incast occurs when at least one of the flows encounters Time-Out in n flows, we get the Incast probability P as

$$P = 1 - (1 - P_{TO})^n = 1 - \left(1 - \left(1 - \frac{nw}{B}\right)^w\right)^n. \tag{3}$$

When the switch buffer is full, the number of in-flight packets in a single round trip time equals to the buffer size B. Then, the average congestion window of each flow is B/n. Since all flows expand their congestion windows by 2 in the subsequent round trip time, the packet loss ratio p will become 0.5 when half of packets are dropped.

Figure 2 plots the Incast probability P, while n and w are increased for 25 TCP flows when the switch buffer is full. The Incast probability P becomes larger due to the high increasing speed of

congestion in the slow start phase. This result shows that in high flow concurrency, the huge number of in-flight packets cause Time-Out easily.

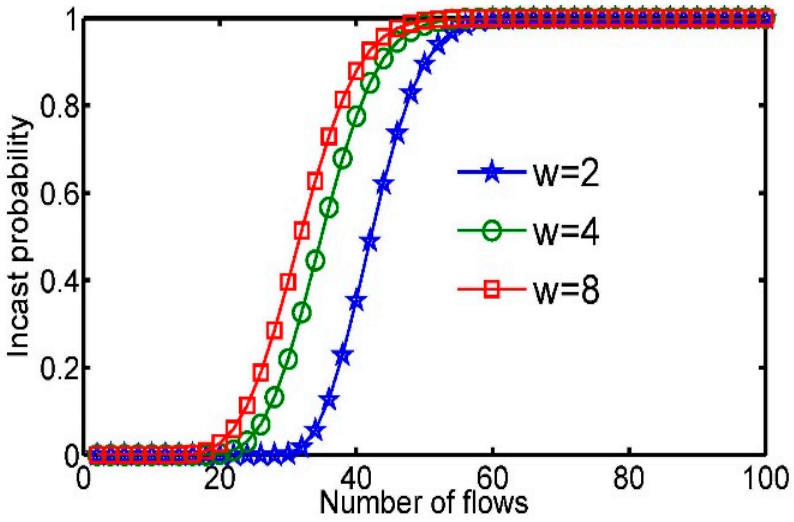


Figure 2. Incast probability with increasing n and w.

2.4. Summary

Our observation leads us to conclude that (i) TCP slow start is very aggressive because of the exponentially increasing congestion window, and (ii) mice flows widely exist, and those flows can finish their transmissions in the slow start phase. These outcomes inspire us to handle the TCP Incast issue by using Gentle Slow Start (GSS), which utilizes information of congestion from the transport layer to adjust the increasing speed of congestion window in slow start phase.

3. Gentle Slow Start

Design and details of Gentle Slow Start are presented in this section. We first show that adjusting the increasing speed of the congestion window effectively avoids Incast. We then describe the detailed design, including how to get the congestion factor and adjust the congestion window during the slow start phase, according to the real-time congestion state. Finally, we describe the implementation of Gentle Slow Start.

3.1. Adjusting the Increasing Speed of the Congestion Window

We use bandwidth delay product (BDP) to represent the packets number that may be served in the link pipeline and switch buffer as

$$BDP = \frac{C * RTT + B}{MSS}, \tag{4}$$

where MSS is the packet size, B and C are the buffer size and total link capacity, respectively. Assume that there are n simultaneous flows, we get the average size of congestion window w as

$$w = \frac{BDP}{n}. \tag{5}$$

To avoid Time-Out and accommodate as many concurrent flows as possible, the congestion window should be increased in a gentle manner. Here, we take the additive increasing as an example. The congestion window w for the next round of RTT is $w + 1$. Then P is calculated as

$$P = 1 - (1 - \left(1 - \frac{n * (w+1) - BDP}{n * (w+1)}\right)^{w})^{n}. \tag{6}$$

The Incast probability is depicted in Figure 3. The number of TCP flows n and the size of congestion window w are changed. When Gentle Slow Start is deactivated, with the growing number of flows n, the Incast probability grows larger because of heavy congestion. Moreover, we discover that it is an easy way to experience TCP Incast under a smaller bandwidth link. If the bandwidth link is 1 Gbps, when n is 65, the Incast probability reaches to 1. While the bandwidth link becomes 10 Gbps, when n is 78, the Incast probability becomes 1. However, if Gentle Slow Start is used, the maximum number of flows without introducing Incast increases greatly, displaying that the Time-Out event is impressively mitigated.

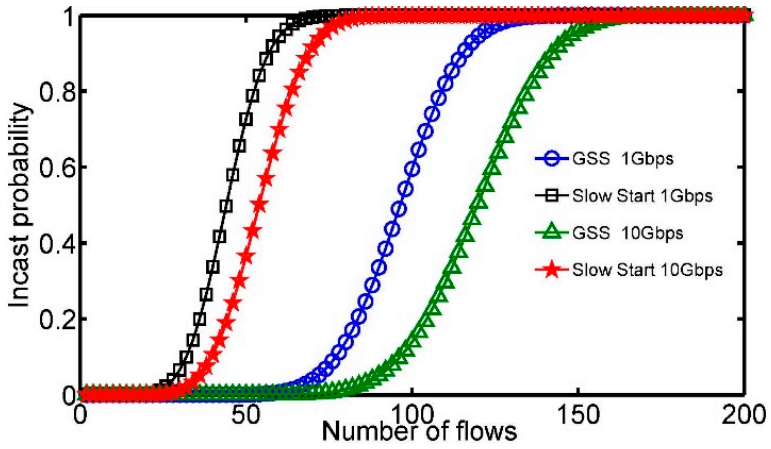


Figure 3. Incast probability with varying number of flows.

3.2. Protocol Design

The design of Gentle Slow Start contains several key challenges. First, we need to estimate congestion state. Second, we need to design the adjustment strategy for the congestion window, which shifts gently from slow start to congestion avoidance phase.

(1) Estimate congestion state: Gentle Slow Start employs congestion information from round trip time RTT to calculate congestion factor α to determine congestion state.

$$\alpha = \frac{RTT - RTT_{\min}}{RTT_{\max} - RTT_{\min}}, \tag{7}$$

where RTT, $RTT_{\max}$, and $RTT_{\min}$ are the current RTT, maximum RTT, and minimum RTT, respectively.

(2) Adjustment strategy of congestion window: In our design, Gentle Slow Start updates the congestion window size $cwnd_{i+1}$ of the $(i+1)_{th}$ RTT as

$$cwnd_{i+1} = cwnd_i + cwnd_i^{1-\alpha} \tag{8}$$

Here, we use α to control the congestion window's increasing speed. With several values of α the congestion window's increasing speed could be clarified by two cases:

Case 1: When α is close to 0, it means that the current network is getting less congested. Then, the increasing speed becomes fast and finally exponential increasing is used.

Case 2: When α is close to 1, we might infer that the current network becomes congested. Gentle Slow Start should not increase the congestion window aggressively. Then, the current increasing speed of the congestion window converges with additive increasing.

Algorithm 1 shows our design, which only modifies a few lines in the TCP codes.

Algorithm 1 Adjust congestion window algorithm

$n \leftarrow n*$
$cwnd \leftarrow 2$
On receiving a packet from TCP flow i
if Receiving an ACK **then**
    update $RTT_{\min}$, $RTT_{\max}$ and RTT
        $\alpha \leftarrow \frac{RTT - RTT_{\min}}{RTT_{\max} - RTT_{\min}}$
        if Receiving all the ACK in the previous congestion window **then**
            $cwnd_{\,i+1} = cwnd_i + cwnd_i^{1-\alpha}$
        end if
else
    go to congestion avoidance
end if

3.3. Implementation

We should note that there are three key steps in our implementation. First, we simply adjust the congestion window by updating α through measuring $RTT_{\min}$, $RTT_{\max}$, and current RTT. Many researchers may argue that accurate RTT is hard to be precisely measured in DCNs because of the host delay and virtualization. Recently, however, several approaches are proposed to get accurate RTT [16, 17]. For example, Timely-RTT measures RTT by timestamps, which is provided by special hardware NIC. Thus, in the real implementation, we could adopt a similar approach to get accurate RTT.

The second one is that most of the protocols initialize the congestion window in an aggressive manner. For example, the initial size of the congestion window is set from 10 to 50 [18], which is very aggressive for highly concurrent TCP flows. Here, we set the initial congestion window as 2.

The last key point is that the Gentle Slow Start algorithm is quite easy to implement in current data center TCP protocols, such as DCTCP, L2DCT, and D2TCP. All the problems regarding implementation could be resolved without any great modification in the transport protocols. We simply adjust the congestion window during slow start phase in the original transport protocol. With Gentle Slow Start, the Incast problem could be avoided automatically in the transport protocols, and achieves higher network goodput.

4. Performance Evaluation

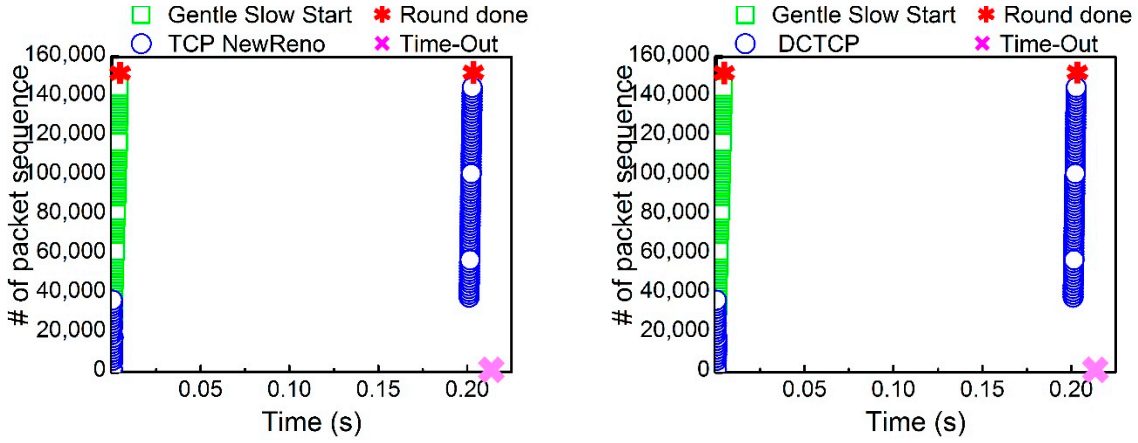
In this section, we conduct the performance evaluation of DCTCP and TCP NewReno with Gentle Slow Start. Notice that TCP NewReno is the most utilized TCP variant, while DCTCP is notable for data center networks transport protocol. The performance of Gentle Slow Start is tested in two typical applications (i.e., Map Reduce and Web search application). At last, we test Gentle Slow Start with different typical data center protocols (i.e., DCTCP, D2TCP, and L2DCT) under different flow concurrency.

By using large-scale NS2 simulation tests, multiple flows are sent to a single aggregator through one ToR (top of the rack) switch, and the switch buffer can accommodate 250 packets. DCTCP Marking threshold is set to 20, as referred to in [5]. For the sake of homogenous topology, a 10 Gbps bandwidth is assigned to all links with 100 µs round trip propagation delay. Like the default setting in most Linux Kernels, $RTO_{\min}$ is set to 200 ms. We use goodput as the performance metric, which is measured at the application layer.

4.1. Basic Performance

We first test the basic performance of Gentle Slow Start. In order to fetch data, the aggregator sends requests to 70 workers, while in return, the aggregator gets responses from individual workers. Figure 4 represents the packet sequence where the aggregator received the responses successfully. Both TCP NewReno and DCTCP in Figure 4a,b encounter Time-Out at around 0.20 s. Few flows

experience the Time-Out event, and the entire transfer process is deferred by such flows for 200 ms. However, with the Gentle Slow Start, we find that there is no Time-Out, and 70 workers are done in less time, about 0.01 s, significantly enhancing the network performance.

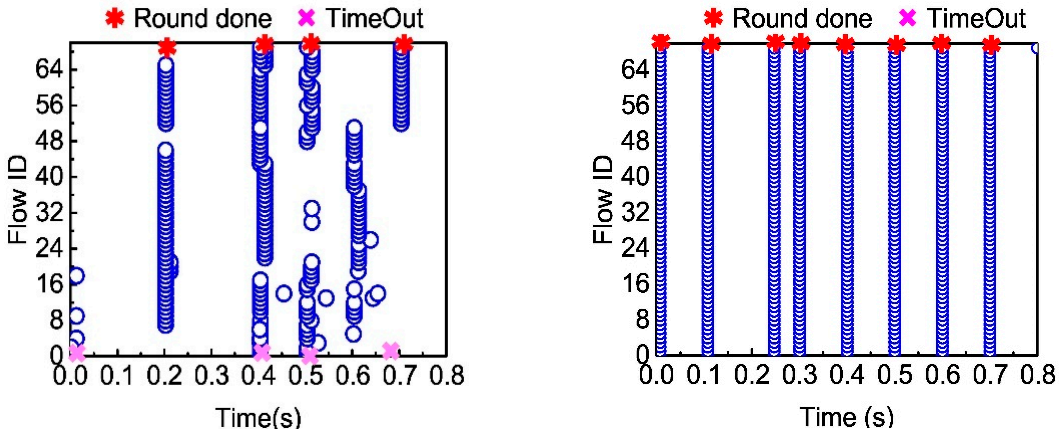


(**a**) GSS with and without TCP NewReno (**b**) GSS with and without DCTCP

Figure 4. Packet sequence of successfully received responses.

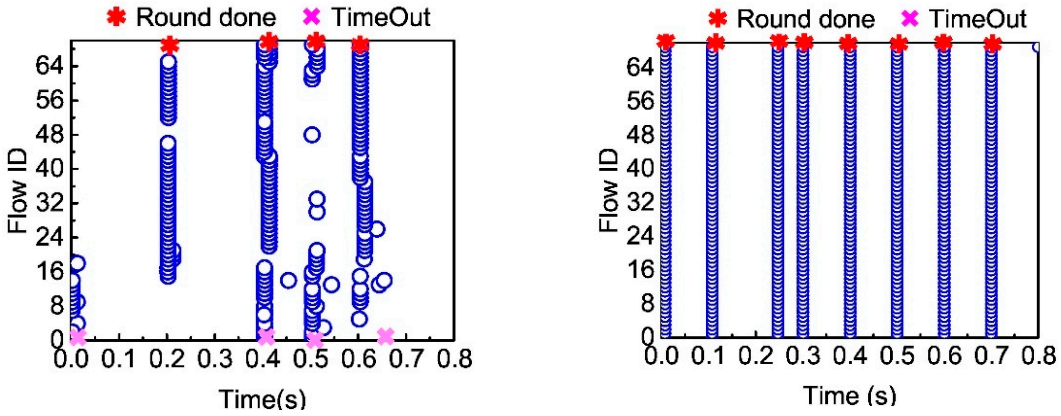
To make this scenario more realistic, we change the number of requests from a single round to multiple rounds to simulate the transmission process of an Incast scene. Figure 5 shows the completion time when each flow ends the transferring of its data, with or without Gentle Slow Start. As showed in Figure 5a,b, by adjusting the increment size of the congestion window during the slow start phase, the entire transfer time of TCP NewReno has 7x reduction. With Gentle Slow Start, 8 rounds of requests finish in 0.22 s, while only a single round of requests can be done without the Gentle Slow Start. From Figure 5c,d, we notice that the Gentle Slow Start also significantly speeds up the whole transfer for DCTCP. Without the Gentle Slow Start, the network goodput is greatly reduced if workers simultaneously deliver data to a single aggregator.

The network goodput with the increasing number of flows is shown in Figure 6. While the number of flows exceeds 20, TCP NewReno experiences Incast. DCTCP uses ECN as an exact congestion sign, and hence gets high goodput till the number of flows increases to 50. With the increasing of flow concurrency, DCTCP still encounters goodput collapse. By regulating the increment size of congestion window during Slow Start phase, it is observed that both DCTCP and TCP NewReno with Gentle Slow Start can support many more concurrent flows and achieve high goodput.



(**a**) TCP NewReno without GSS (**b**) TCP NewReno with GSS

Figure 5. *Cont.*



(**c**) DCTCP without GSS (**d**) DCTCP with GSS

Figure 5. Completion time of different rounds.

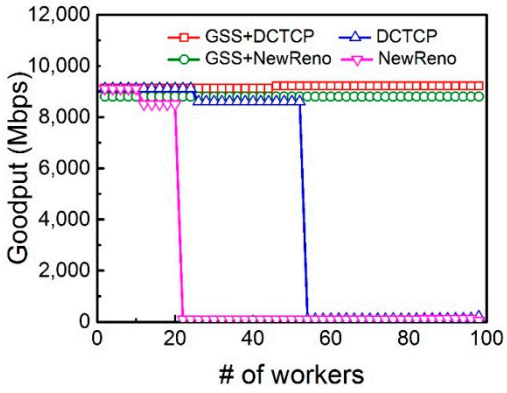
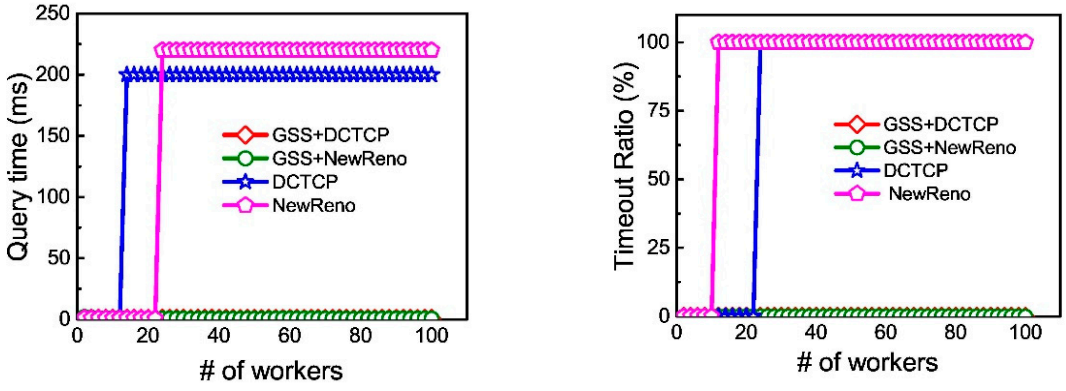


Figure 6. Goodput with varying number of flows.

4.2. Web Search Application

We examine the performance of DCTCP and TCP NewReno with GSS in the web search application. The test metric is the query time, which is the delay between queries dispatched by the aggregator and all responses are received.

We use n to denote the number of total workers and 1024 KB/n is the return data size of every worker. We record the query time of up to 100 workers, as shown in Figure 7a. During experiments, both DCTCP and NewReno with GSS show low query time in all cases. However, because of the Incast congestion, TCP NewReno's query time is raised up to 220 ms, while DCTCP's query time increases to around 200 ms.



(**a**) Search Response Time (**b**) Time-Out Ratio

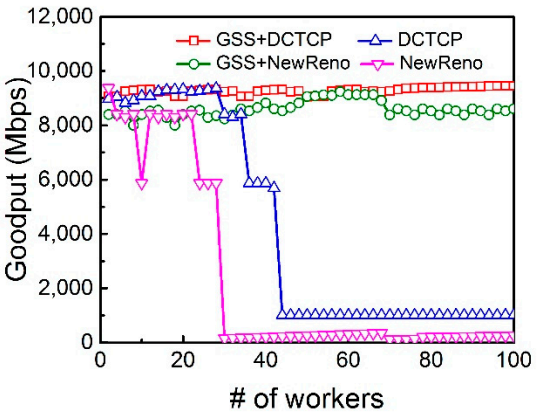
Figure 7. Web search application.

We also record the Time-Out ratio, that is the fraction of queries that experience at least one Time-Out, as shown in Figure 7b. When the number of workers exceeds 13, TCP NewReno suffers at least one Time-Out event. When the number of workers is more than 20, DCTCP begins to suffer from Time-Out. On the other hand, DCTCP with GSS and TCP NewReno with GSS do not experience Time-Out in any cases, which means that Gentle Slow Start successfully avoids Time-Out.

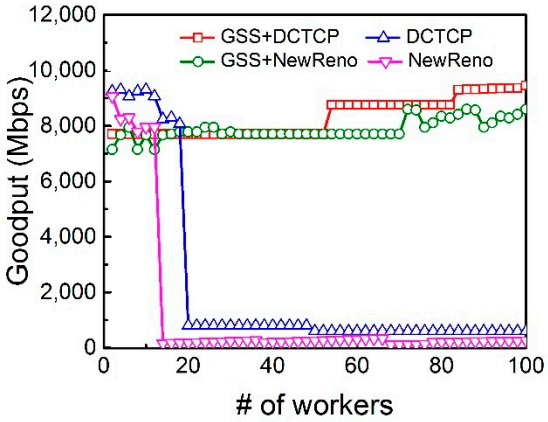
4.3. MapReduce Application

In MapReduce application, an aggregator generates queries to multiple senders, and each of them immediately responds with Sever Request Unit (SRU). To test the network performance, we evaluate the performances of DCTCP and TCP NewReno with GSS. The SRU size is 256 KB.

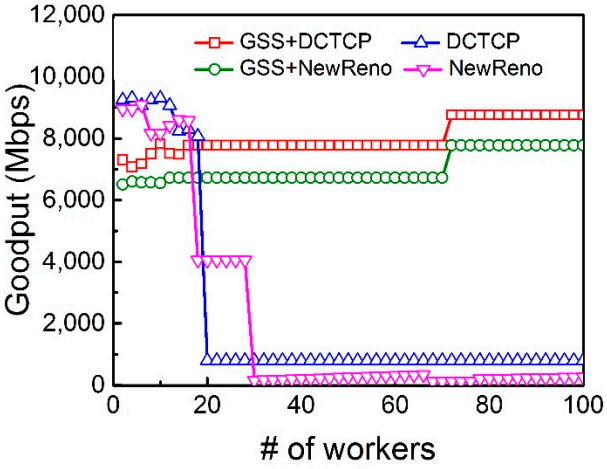
Figure 8a shows that when the number of workers increases, both DCTCP with GSS and NewReno with GSS obtain higher goodput. The performance of DCTCP is better than TCP NewReno. Nonetheless, DCTCP still experiences goodput collapse once the number of flows exceeds 20.



(**a**) TCP flow with no background flow



(**b**) TCP flow with one background flow



(**c**) TCP flows with two background flows

Figure 8. MapReduce application.

Next, in order to test the impact of background workload, we add up to 2 long TCP flows. The TCP background flows are sent by the workers that are just sending background flows. The network goodputs with 1 and 2 background flows are shown in Figure 8b,c, respectively. The goodput of DCTCP and NewReno are less than that without background flows. Fortunately, with Gentle Slow Start enabled, both DCTCP and TCP NewReno obtain higher goodput than the original DCTCP and TCP NewReno. This is because that the Gentle Slow Start avoids congestion through adjusting the increment size of the congestion window during the slow start phase as the number of workers increases.

4.4. Performance Results with Different Data Center Protocols

We test the performance of different protocols (i.e., DCTCP, D2TCP and L2DCT) with or without Gentle Slow Start (GSS) in this section. In this test, the aggregator server receives one flow from each individual worker. The network goodput with up to 100 workers is shown in Figure 9.

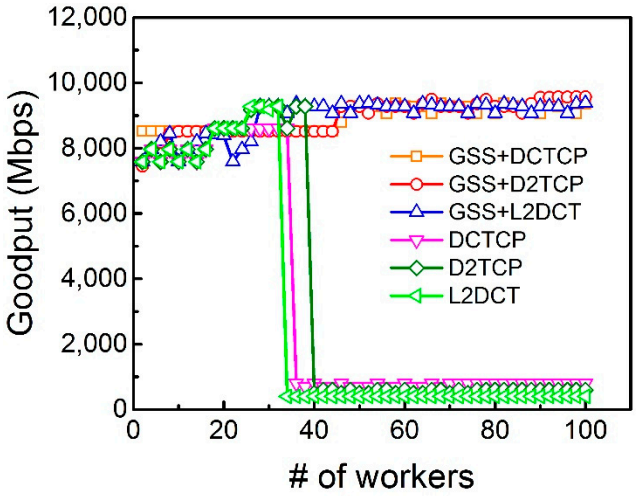


Figure 9. Network goodput with increasing number of workers.

We find that when the number of workers exceeds 35, DCTCP, D2TCP, and L2DCT experience great performance degradation. Fortunately, with Gentle Slow Start enabled, all these protocols achieve quite high goodput. For example, for the 100 workers, Gentle Slow Start working together with the three protocols achieves 35× incremental increase in network goodput.

5. Related Work

The researchers have reported the noteworthy defer gap between RTO_{min} (e.g., no less than 200 ms) and RTT (e.g., hundreds of microsecond). Different proposals aim at upgrading TCP protocol to avoid TCP throughput collapse. In order to achieve exact congestion control [5], DCTCP uses Explicit Congestion Notification (ECN) to tune the size of the congestion window. DCTCP adopts a simple yet effective active queue management scheme, whereby the switch marks the packets with the Congestion Experienced (CE) code point when the buffer occupancy exceeds a pre-determined threshold. Based on the marked packets fraction, DCTCP tunes the size of the congestion window and obtains good performance, such as high throughput and low queueing delay.

In order to control the aggregate throughput, ICTCP [7] adaptively tunes the advertisement window. Moreover, several researchers tried to avoid the problem of TCP Incast by extending the switch buffer size, changing block size, and shrinking Maximum Transmission Unit (MTU) [19]. In practice, Facebooks researchers alleviate the problem of TCP Incast by limiting the number of concurrent flows [20]. Packet Slicing [21] utilizes simple ICMP messages to regulate the IP packet size with low overhead, letting the switch buffer facilitate extra packets. Similarly, Proactive ACK Control (PAC) [22] prevents Incast congestion by throttling the sending rate of ACKs on the receiver. A session-aware mechanism [23] is proposed to slow down the leading flows at the presence of congestion, thus, the lagging ones will have a greater chance to catch up to obtain improved goodput and session completion time. In [24], the authors provided a wide study to examine the basic reason of degraded performance of highly concurrent HTTP connections in data center networks. To avoid the TCP Incast problem, the probe packets are used to detect the available bandwidth at the beginning of ON periods in HTTP congestions. Furthermore, in [25] the authors proposed an Adaptive Pacing (AP) scheme, which adjusts the micro-bursts dynamically based on the flow concurrency and real-time network congestion state.

The TCP Incast problem is also solved at other layers [26,27]. PLATO [28] adopts a packet labeling scheme to prevent labeled packets from being dropped. This solution can maintain the ACK-clocking and avoid timeout. However, PLATO requires modification on the switch, which is not readily deployed. Adaptive Request Scheduling (ARS [29] adjusts the number of requests in the application layer according to the congestion information from the transport layer. Proactive Incast Congestion Control (PICC) [30] limits the number of concurrent flows connected to the front-end server to avoid the Incast congestion through data placement. Furthermore, several researchers employ a coding-based approach to prevent TCP Incast [31,32]. However, transmitting lots of redundant packets inevitably reduces the link utilization.

Compared with the improved TCP protocols concentrating on the adjustment of the congestion window in the congestion avoidance phase, our solution Gentle Slow Start deals the TCP Incast issue by adjusting the increasing speed of the congestion window during the slow start phase. This key difference enables the Gentle Slow Start to reduce the probability of Time-Out in high concurrent flows, where current protocols become ineffective. Gentle Slow Start could be integrated directly into the state-of the-art TCP protocols designed for data centers. Furthermore, Gentle Slow Start only adjusts the sender-side, which makes it easy to be deployed.

6. Conclusions and Future Work

We design and implement GSS, the Gentle Slow Start for data center networks to mitigate TCP Incast problem. Gentle Slow Start adjusts the increasing speed of the congestion window during the slow start phase according to the real-time congestion state in a gentle manner. Gentle Slow Start not only significantly reduces the Time-Out probability in highly concurrent scenarios but also smoothly switches slow start to congestion avoidance phase.

The key feature of Gentle Slow Start is broad applicability. In other words, Gentle Slow Start could be integrated directly into other transport protocols and achieves great improvement in performance. We integrated Gentle Slow Start with TCP NewReno, DCTCP, L2DCT, and D2TCP. The experimental results indicate that these protocols with Gentle Slow Start enabled significantly alleviation of the TCP Incast problem. The network goodput is increased remarkably by up to 8 times.

In the future work, we will deploy Gentle Slow Start protocol in TCP stack protocol and use the Linux-based testbed to evaluate GSS performance in the real system. Furthermore, we will optimize the congestion window adjustment in the GSS algorithm.

Author Contributions: S.M. and J.H. conceived and designed the whole system; S.M. and J.H. designed the algorithm and model; S.M. and H.S. conducted the experiment; S.M. and J.H. wrote the research paper.

Funding: This work is supported by National Natural Science Foundation of China (61872387, 61572530, 61872403) and CERNET Innovation Project (Grant No. NGII20160113, NGII20170107).

Conflicts of Interest: I declare that none of the authors have a conflict of interest.

References

1. Benson, T.; Akella, A.; Maltz, D.A. Network traffic characteristics of data centers in the wild. In Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, Melbourne, Australia, 1–3 November 2010; pp. 267–280.
2. Ananthanarayanan, G.; Kandula, S.; Greenberg, A.G.; Stoica, I.; Lu, Y.; Saha, B.; Harris, E. Reining in the Outliers in Map-Reduce Clusters using Mantri. In Proceedings of the of the 9th USENIX Conference on Operating Systems Design and Implementation, Vancouver, BC, Canada, 4–6 October 2010; p. 24.
3. Vasudevan, V.; Phanishayee, A.; Shah, H.; Krevat, E.; Andersen, D.G.; Ganger, G.R.; Gibson, G.A.; Mueller, B. Safe and effective fine-grained TCP retransmissions for datacenter communication. In Proceedings of the ACM SIGCOMM Computer Communication Review, Barcelona, Spain, 16–21 August 2009; pp. 303–314.
4. Zhang, J.; Ren, F.; Lin, C. Survey on transport control in data center networks. *IEEE Netw.* **2013**, *27*, 22–26. [CrossRef]

5. Alizadeh, M.; Greenberg, A.; Maltz, D.A.; Padhye, J.; Patel, P.; Prabhakar, B.; Sengupta, S.; Sridharan, M. Data center Tcp (Dctcp). In Proceedings of the ACM SIGCOMM 2010 Conference, New Delhi, India, 30 August–3 September 2010.
6. Zhang, J.; Ren, F.; Yue, X.; Shu, R.; Lin, C. Sharing bandwidth by allocating switch buffer in data center networks. *IEEE J. Sel. Areas Commun.* **2014**, *32*, 39–51. [CrossRef]
7. Wu, H.; Feng, Z.; Guo, C.; Zhang, Y. ICTCP: Incast congestion control for TCP in data-center networks. *IEEE/ACM Trans. Netw. (ToN)* **2013**, *21*, 345–358.
8. Dong, P.; Yang, W.; Tang, W.; Huang, J.; Wang, H.; Pan, Y.; Wang, J. Reducing transport latency for short flows with multipath TCP. *J. Netw. Comput. Appl.* **2018**, *108*, 20–36. [CrossRef]
9. Ha, S.; Rhee, I. Taming the elephants: New TCP slow start. *Comput. Netw.* **2011**, *55*, 2092–2110. [CrossRef]
10. Vamanan, B.; Hasan, J.; Vijaykumar, T. Deadline-aware datacenter TCP (D2TCP). In Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Helsinki, Finland, 13–17 August 2012; pp. 115–126.
11. Munir, A.; Qazi, I.; Uzmi, Z.; Mushtaq, A.; Ismail, S.N.; Safdar Iqbal, M.; Khan, B. Minimizing flow completion times in data centers. In Proceedings of the 2013 Proceedings IEEE INFOCOM, Turin, Italy, 14–19 April 2013; pp. 2157–2165.
12. Cheng, W.; Ren, F.; Jiang, W.; Qian, K.; Zhang, T.; Shu, R. Isolating Mice and Elephant in Data Centers. *arXiv*, **2016**; arXiv:1605.07732.
13. Atikoglu, B.; Xu, Y.; Frachtenberg, E.; Jiang, S.; Paleczny, M. Workload analysis of a large-scale key-value store. In Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems, London, UK, 11–15 June 2012; pp. 53–64.
14. Phanishayee, A.; Krevat, E.; Vasudevan, V.; Andersen, D.G.; Ganger, G.R.; Gibson, G.A.; Seshan, S. Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems. In Proceedings of the 6th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, 26–29 February 2008; pp. 1–14.
15. Chen, W.; Ren, F.; Xie, J.; Lin, C.; Yin, K.; Baker, F. Comprehensive understanding of TCP Incast problem. In Proceedings of the 2015 IEEE Conference on Computer Communications, Kowloon, Hong Kong, 26 April–1 May 2015; pp. 1688–1696.
16. Mitta, R.; Dukkipati, N.; Blem, E.; Wassel, H.; Ghobadi, M.; Vahdat, A.; Wang, Y.; Wetherall, D.; Zats, D. TIMELY: RTT-based Congestion Control for the Datacenter. In Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, London, UK, 17–21 August 2015; pp. 537–550.
17. Lee, C.; Park, C.; Jang, K.; Moon, S.B.; Han, D. Accurate Latency-based Congestion Feedback for Datacenters. In Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference, Santa Clara, CA, USA, 8–10 July 2015; pp. 403–415.
18. Dukkipati, N.; Refice, T.; Cheng, Y.; Chu, J.; Herbert, T.; Agarwal, A.; Jain, A.; Sutin, N. An argument for increasing TCP's initial congestion window. *Comput. Commun. Rev.* **2010**, *40*, 26–33. [CrossRef]
19. Zhang, P.; Wang, H.; Cheng, S. Shrinking MTU to mitigate tcp incast throughput collapse in data center networks. In Proceedings of the 2011 Third International Conference on Communications and Mobile Computing, Qingdao, China, 18–20 April 2011; pp. 126–129.
20. Nishtala, R.; Fugal, H.; Grimm, S.; Kwiatkowski, M.; Lee, H.; Li, H.C.; McElroy, R.; Paleczny, M.; Peek, D.; Saab, P.; et al. Scaling Memcache at Facebook. In Proceedings of the NSDI, Lombard, IL, USA, 2–5 April 2013; pp. 385–398.
21. Huang, J.; Huang, Y.; Wang, J.; He, T. Adjusting Packet Size to Mitigate TCP Incast in Data Center Networks with COTS Switches. *IEEE Trans. Cloud Comput.* **2018**. [CrossRef]
22. Bai, W.; Chen, K.; Wu, H.; Lan, W.; Zhao, Y. Pac: Taming TCP incast congestion using proactive ack control. In Proceedings of the IEEE 23rd International Conference on Network Protocols (ICNP), Raleigh, NC, USA, 21–24 October 2014; pp. 385–396.
23. Li, D.; Li, S.; Du, Z. Session-aware congestion control for TCP Incast in datacenter networks. In Proceedings of the 2016 IEEE Symposium on Computers and Communication (ISCC), Messina, Italy, 27–30 June 2016; pp. 1227–1232.
24. Zhang, T.; Wang, J.; Huang, J.; Chen, J.; Pan, Y.; Min, G. Tuning the Aggressive TCP Behavior for Highly Concurrent HTTP Connections in Intra-datacenter. *IEEE/ACM Trans. Netw.* **2017**, *25*, 3808–3822. [CrossRef]

25. Zou, S.; Huang, J.; Zhou, Y.; Wang, J.; He, T. Flow-Aware Adaptive Pacing to Mitigate TCP Incast in Data Center Networks. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017; pp. 2119–2124.
26. Ren, Y.; Zhao, Y.; Liu, P.; Dou, K.; Li, J. A survey on TCP Incast in data center networks. *Int. J. Commu. Syst.* **2014**, *27*, 1160–1172. [CrossRef]
27. Ruan, C.; Wang, J.; Jiang, W.; Huang, J.; Min, G.; Pan, Y. FSQCN: Fast and Simple Quantized Congestion Notification in Data Center Ethernet. *J. Netw. Comput. Appl.* **2017**, *83*, 53–62. [CrossRef]
28. Shukla, S.; Chan, S.; Tam, A.S.; Gupta, A.; Xu, Y.; Chao, H.J. TCP PLATO: packet labelling to alleviate time-out. *IEEE J. Sel. Areas Commu.* **2014**, *32*, 65–76. [CrossRef]
29. Huang, J.; He, T.; Huang, Y.; Wang, J. ARS: Cross-layer adaptive request scheduling to mitigate TCP incast in data center networks. In Proceedings of the IEEE INFOCOM 2016—The 35th Annual IEEE International Conference on Computer Communications, San Francisco, CA, USA, 10–14 April 2016; pp. 1–9.
30. Wang, H.; Shen, H. Proactive Incast Congestion Control in a Datacenter Serving Web Application. In Proceedings of the INFOCOM, Honolulu, HI, USA, 15–19 April 2018.
31. Jiang, C.; Li, D.; Xu, M.; Zheng, K. A coding-based approach to mitigate TCP Incast in data center networks. In Proceedings of the 32nd International Conference on Distributed Computing Systems Workshops (ICDCSW), Macau, China, 18–21 June 2012; pp. 29–34.
32. Jiang, C.; Li, D.; Xu, M. LTTP: an LT-code based transport protocol for many-to-one communication in data centers. *IEEE J. Sel. Areas Commu.* **2014**, *32*, 52–64. [CrossRef]