TCP Congestion Control in Data Center Networks

Rohit P. Tahiliani, Mohit P. Tahiliani and K. Chandra Sekaran

1 Introduction

Internet over the past few years has transformed from an experimental system into a gigantic and decentralized source of information. Data centers form the backbone of the Internet and host diverse applications ranging from social networking to web search and web hosting to advertisements. Data Centers are mainly classified into two types [1]: the ones that aim to provide *on-line services* to users, e.g., Google, Facebook and Yahoo, and others that aim to provide *resources* to users e.g., Amazon Elastic Compute Cloud (EC2) and Microsoft Azure.

Transmission Control Protocol (TCP) is one of the most dominant transport protocols, widely used by a large variety of Internet applications and also constitutes majority of the traffic in both types of Data Centers [2]. It has been the workhorse of the Internet ever since its inception. The success of the Internet, in fact, can be partly attributed to the congestion control mechanisms implemented in TCP. Though the scale of the Internet and its usage increased exponentially in recent past, TCP has evolved to keep up with the changing network conditions and has proven to be scalable and robust.

Data Center environment, however, is largely different than that of the Internet e.g., the Round Trip Time (RTT) in Data Center Networks can be as less as 250 μs in the absence of queuing [3]. The reason is that Data Center Networks are well designed and layered to achieve high-bandwidth and low-latency. Moreover, the nature of traffic in Data Center Networks largely varies from that of the Internet traffic. Traffic in Data Center Networks is classified mainly into three types [2]: (i) Mice traffic - the

R. P. Tahiliani (✉)
Department of Computer Science and Engineering, NMAM Institute of Technology, Nitte, Karnataka, 574110 India

M. P. Tahiliani · K. C. Sekaran
Department of Computer Science and Engineering, NITK, Surathkal Karnataka, 575025 India
e-mail: tahiliani@nitk.ac.in

K. C. Sekaran
e-mail: kchnitk@gmail.com

S. U. Khan, A. Y. Zomaya (eds.), *Handbook on Data Centers,*
DOI 10.1007/978-1-4939-2092-1_15

Table 1 Data Center Traffic: applications and performance requirements

Traffic Type	Examples	Requirements
Mice traffic (< 100 KB)	Google Search, Facebook	Short response times
Cat traffic (100 KB-5 MB)	Picasa, YouTube, Facebook photos	Low latency
Elephant traffic (> 5 MB)	Software updates, Video On-demand	High throughput

queries form the mice traffic (e.g. Google search, Facebook updates, etc). Majority of the traffic in a data center network is query traffic and its data transmission volume is usually less. (ii) Cat traffic - the control state and co-ordination messages form the cat traffic (e.g. small and medium sized file downloads, etc) and (iii) Elephant traffic - the large updates form the elephant traffic (e.g. anti-virus updates, movie downloads, etc). The different traffic types in Data Center Networks, their applications and performance requirements are summarized in Table 1.

Thus, bursty query traffic, delay sensitive cat traffic and throughput sensitive elephant traffic co-exist in Data Center Networks. Therefore, the three basic requirements of the data center transport are high burst tolerance, low latency and high throughput [2]. The state-of-the-art TCP fails to satisfy these requirements together within the time boundaries because of impairments such as TCP Incast [3], TCP Outcast [4], Queue build-up [2], Buffer pressure [2] and Pseudo-Congestion Effect [5] which are discussed further in this chapter.

Recently, a few TCP variants have been proposed for Data Center Networks. The major goal of these TCP Variants is to overcome the above mentioned impairments and improve the performance of TCP in Data Center Networks. This chapter presents the background and the causes of each of the above mentioned impairments, followed by a comparative study of TCP variants that aim to overcome these impairments. Although a few other transport protocols have also been proposed for Data Center Networks, we restrict the scope of this chapter to TCP variants because TCP is the most widely deployed transport protocol in modern operating systems.

2 TCP Impairments in Data Center Networks

Although TCP constantly evolved over a period of three decades, the diversity in the characteristics of present and next generation networks and a variety of application requirements have posed several challenges to TCP congestion control mechanisms. As a result, the shortcomings in the fundamental design of TCP have become increasingly apparent. In this section, we mainly focus on the challenges faced by the state-of-the-art TCP in DCNs.

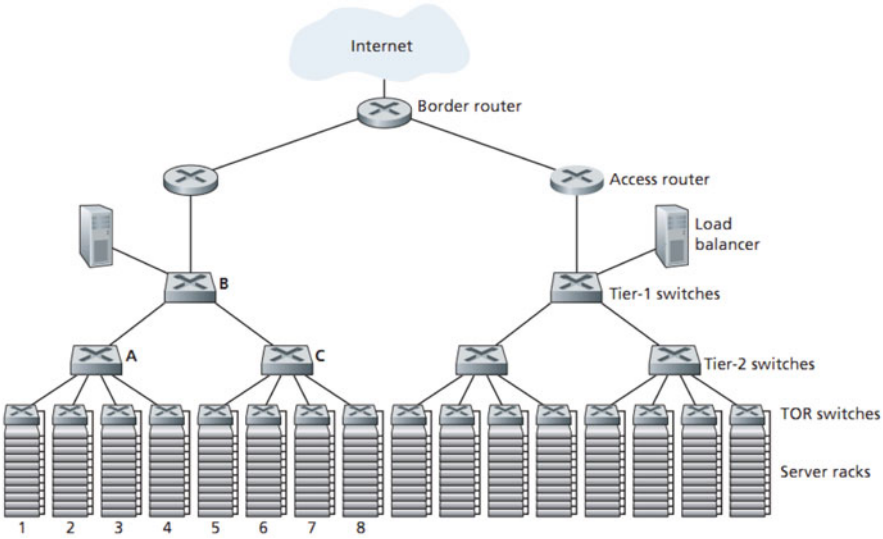

Fig. 1 Partition/aggregate application structure [8]

2.1 TCP Incast

TCP Incast has been defined as the pathological behavior of TCP that results in gross under-utilization of the link capacity in various many-to-one communication patterns [6], e.g. partition/aggregate application pattern as shown in Fig. 3. Such patterns are the foundation of numerous large scale applications like web search, MapReduce, social network content composition, advertisement selection, etc [2, 7]. As a result, TCP Incast problem widely exists in today's data center scenarios such as distributed storage systems, data-intensive scalable computing systems and partition/aggregate workflows [1].

In many-to-one communication patterns, an aggregator issues data requests to multiple worker nodes. The worker nodes upon receiving the request, concurrently transmit a large amount of data to the aggregator (see Fig. 2). The data from all the worker nodes traverse a bottleneck link in many-to-one fashion. The probability that all the worker nodes send the reply at the same time is high because of the tight time bounds. Therefore, the packets from these nodes happen to overflow the buffers of Top of the Rack (ToR) switches and thus, lead to packet losses. This phenomenon is known as *synchronized mice collide* [2]. Moreover, no worker node can transmit the next data block until all the worker nodes finish transmitting the current data block. Such a transmission is termed as *barrier synchronized transmission* [7].

Under such constraints, as the number of concurrent worker nodes increases, the perceived application level throughput at the aggregator decreases due to a large number of packet losses. The lost packets are retransmitted only after the Retransmit TimeOut (RTO), which is generally in the order of few *milliseconds*. As mentioned

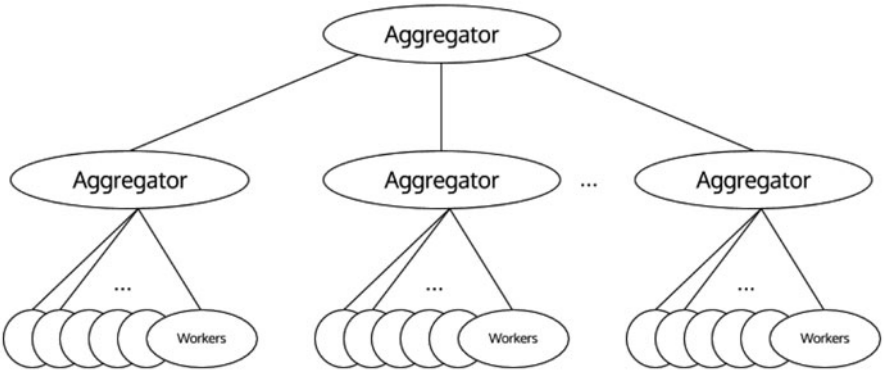

Fig. 2 Many-to-one communication pattern

earlier, mice traffic requires short response time and is highly delay sensitive. Frequent timeouts resulting out of Incast significantly degrade the performance of mice traffic as the lost packets are retransmitted after a few *milliseconds*.

It must be noted that *Fast Retransmit* mechanism may not be applicable to mice traffic applications since the data transmission volume of such traffic is quite less and hence, there are very few packets in the entire flow. As a result, the sender (or worker node) may not get sufficient duplicate acknowledgements (*dupacks*) to trigger a Fast Retransmit.

Mitigating TCP Incast: A lot of solutions, ranging from application layer solutions to transport layer solutions and link layer solutions have been proposed recently to overcome the TCP Incast problem. A few solutions suggest *revision of TCP*, others recommend to *replace TCP* while some seek solutions from layers other than the transport layer to solve this problem [1]. Ren et al. [9] provides a detailed analysis and summary of all such solutions.

2.2 TCP Outcast

When a large set of flows and a small set of flows arrive at two different input ports of a switch and compete for the same bottleneck output port, the small set of flows lose out on their throughput share significantly. This phenomenon has been termed as TCP Outcast [4] and mainly occurs in Data Center switches that employ drop-tail queues. Drop-tail queues lead to consecutive packet drops from one port and hence, cause frequent TCP timeouts. This property of drop-tail queues is termed as *Port Blackout* [4] and it significantly affects the performance of small flows because frequent timeouts lead to high latencies and thus, poor quality response times. Figure 3 shows an example scenario of port blackout where A and B are input ports whereas

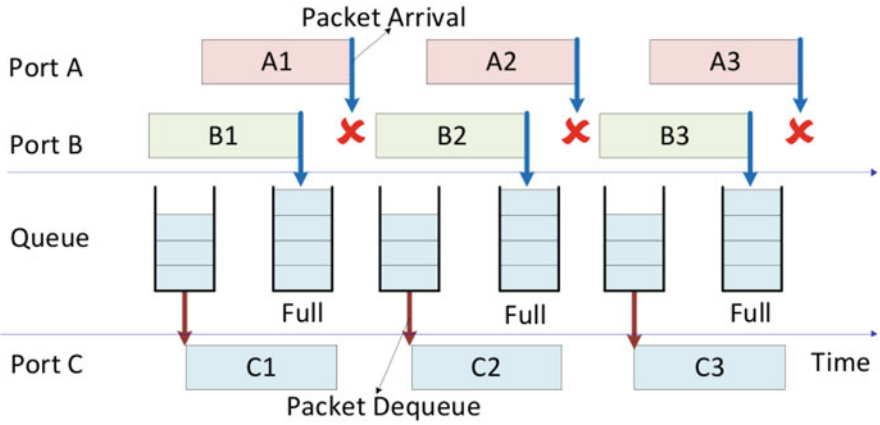

Fig. 3 Example scenario of Port Blackout [4]

C is the common output port. The figure shows that packets arriving at Port B are successfully queued whereas those arriving at Port A are dropped consecutively.

It is well known that the throughput of a TCP flow is inversely proportional to the RTT of that flow. This behavior of TCP leads to RTT-bias i.e., flows with low RTT achieve larger share of bandwidth than the flows with high RTT. However, it has been observed that due to TCP Outcast problem in Data Center Networks, TCP exhibits *Inverse RTT-bias* [4] i.e., flows with low RTT are outcasted by flows with high RTT.

The two main factors that cause TCP Outcast are: (i) the usage of drop-tail queues in switches and (ii) many-to-one communication pattern which leads to a large set of flows and a small set of flows arriving at two different input ports and competing for the same bottleneck output port. Both these factors are quite common in Data Center Networks since majority of the switches employ drop-tail queues and many-to-one communication pattern is the foundation of many cloud applications.

Mitigating TCP Outcast: One possible approach to mitigate TCP Outcast is to use queuing mechanisms other than drop-tail e.g., Random Early Detection (RED) [10], Stochastic Fair Queue (SFQ) [4], etc. Another possible approach is to minimize the buffer occupancy at the switches by designing efficient TCP congestion control laws at the end hosts.

2.3 Queue Buildup

Due to the diverse nature of cloud applications, mice traffic, cat traffic and elephant traffic co-exist in a Data Center Network. The long lasting and greedy nature of elephant traffic drives the network to the point of extreme congestion and overflows the bottleneck buffers. Thus, when both mice traffic and elephant traffic traverse

through the same route, the performance of mice traffic is significantly affected due to the presence of the elephant traffic [2].

Following are two ways in which the performance of mice traffic is degraded due to the presence of elephant traffic [2]: (i) since most of the buffer is occupied by elephant traffic, there is a high probability that the packets of mice traffic get dropped. The impact of this situation is similar to that of TCP Incast because the performance of mice traffic is largely affected by frequent packet losses and hence, the timeouts. (ii) the packets of mice traffic, even when none are lost, suffer from increased queuing delay as they are in queue behind the packets of elephant traffic. This problem is termed as Queue build-up.

Mitigating Queue Buildup: Queue build-up problem can be solved only by minimizing the queue occupancy in the Data Center Network switches. Most of the existing TCP variants employ reactive approach towards congestion control i.e., they do not reduce the sending rate unless a packet loss is encountered, and hence, fail to minimize the queue occupancy. A proactive approach instead, is desired to minimize the queue occupancy and overcome the problem of queue build-up.

2.4 Buffer Pressure

Buffer pressure is yet another impairment caused by the long lasting and greedy nature of elephant traffic. When both mice traffic and elephant traffic co-exist on the same route, most of the buffer space is occupied by packets from the elephant traffic. This leaves a very little room to accommodate the burst of mice traffic packets arising out of many-to-one communication pattern. The result is that large number of packets from mice traffic are lost, leading to poor performance. Moreover, majority of the traffic in Data Center Networks is bursty [2] and hence, packets of mice traffic get dropped frequently because the elephant traffic lasts for a longer time and keeps most of the buffer space occupied.

Mitigating Buffer Pressure: Like Queue build-up, Buffer pressure problem too can be solved by minimizing the buffer occupancy in the switches.

2.5 Pseudo-Congestion Effect

Virtualization is one of the key technologies driving the success of Cloud Computing applications. Modern Data Centers adopt Virtual Machines (VMs) to offer on-demand cloud services and remote access. These data centers are known as *virtualized data centers* [1, 5]. Though there are several advantages of virtualization like efficient server utilization, service isolation and low system maintenance cost [1], it significantly affects the environment where our traditional protocols (e.g., TCP,

Table 2 TCP Impairments in Data Center Networks and their causes

TCP impairment	Causes
TCP Incast	Shallow buffers in switches and Bursts of mice traffic resulting from many-to-one communication pattern
TCP Outcast	Usage of tail-drop mechanism in switches
Queue Buildup	Persistently full queues in switches due to elephant traffic
Buffer Pressure	Persistently full queues in switches due to elephant traffic and Bursty nature of mice traffic
Pseudo-Congestion Effect	Hypervisor scheduling latency

UDP) work. The recent study of Amazon EC2 Data Center reveals that virtualization dramatically deteriorates the performance of TCP and UDP in terms of both, throughput and end to end delay [1]. Throughput becomes unstable and the end to end delay becomes quite large even if the network load is less [1].

When more number of VMs are running on the same physical machine, the hypervisor scheduling latency increases the waiting time for each VM to obtain an access to the processor. Hypervisor scheduling latency varies from microseconds to several hundred milliseconds [5], leading to unpredictable network delays (i.e., RTT) and thus, affecting the throughput stability and largely increasing the end to end delay. Moreover, hypervisor scheduling latency can be so high that it may lead to RTO at the VM sender. Once RTO occurs, VM sender assumes that the network is heavily congested and significantly brings down the sending rate. We term this effect as *pseudo-congestion effect* because the congestion sensed by the VM sender is actually *pseudo-congestion* [5].

Mitigating Pseudo-congestion Effect There are generally two possible approaches to address the above mentioned problem. One is to design efficient schedulers for hypervisor so that the scheduling latency can be minimized. Another approach is to modify TCP such that it can intelligently detect the *pseudo-congestion* and react accordingly.

2.6 Summary: TCP Impairments and Causes

We briefly summarize the TCP impairments discussed in the above subsections and mention the causes for the same in Table 2.

3 TCP Variants for Data Center Networks

3.1 TCP with Fine Grained RTO (FG-RTO) [3]

The default value of minimum RTO in TCP is generally in the order of milliseconds (around 200 ms). This value of RTO is suitable for Internet like scenarios where the average RTT is in order of hundreds of milliseconds. However, it is significantly larger than the average RTT in data centers which is in the order of a few microseconds. Large number of packet losses due to TCP Incast, TCP Outcast, Queue build-up, Buffer pressure and pseudo-congestion effect result in frequent timeouts and in turn, lead to missed deadlines and significant degradation in the performance of TCP. Phanishayee et al. [3] show that reducing the minimum RTO from 200 ms to 200 μs significantly alleviates the problems of TCP in Data Center Networks and improves the overall throughput by several orders of magnitude.

Advantages: The major advantage of this approach is that it requires minimum modification to the traditional working of TCP and thus, can be easily deployed.

Shortcomings: The real time deployment of fine grained timers is a challenging issue because the present operating systems lack the high-resolution timers required for such low RTO values. Moreover, FG-RTOs may be not suitable for servers that communicate to clients through the Internet. Apart from the implementations issues of fine grained timers, it must be noted that this approach of eliminating drawbacks of TCP in Data Center Networks is a *reactive* approach. It tries to reduce the impact of a packet loss rather than *avoiding* the packet loss in the first place. Thus, although this approach significantly improves the network performance by reducing post-packet-loss delay, it does not alleviate the TCP Incast problem for loss-sensitive applications.

3.2 TCP with FG-RTO + Delayed ACKs Disabled [3]

Delayed ACKs are mainly used for reducing the overhead of ACKs on the reverse path. When delayed ACKs are enabled, the receiver sends only one ACK for every two data packets received. If only one packet is received, the receiver waits for delayed ACK timeout period before sending an ACK. This timeout period is usually 40 ms. This scenario may lead to spurious retransmissions if FG-RTO timers (as explained in the previous section) are deployed. The reason is that receiver waits for 40 ms before sending an ACK for the received packet and by that time, FG-RTO which is in order of few microseconds, expires and forces the sender to retransmit the packet. Thus, either the delayed ACK timeout period must be reduced to a few microseconds or must be completely disabled while using FG-RTOs to avoid such spurious retransmissions. This approach further enhances the TCP throughput in Data Center Networks.

Advantages: It has been shown in [3] that reducing the delayed ACK timeout period to 200 μs while using FG-RTO achieves far better throughput than the throughput obtained when delayed ACKs are enabled. Moreover, completely disabling the delayed ACKs while using FG-RTO further improves the overall TCP throughput.

Shortcomings: The shortcomings of this approach are exactly similar to that of TCP with FG-RTO because this approach is an undesired side-effect of the previous approach.

3.3 DCTCP: Data Center TCP [2]

Additive Increase Multiplicative Decrease (AIMD) is the cornerstone of TCP congestion control algorithms. When an acknowledgement (ACK) is received in AIMD phase, the congestion window (*cwnd*) is increased as shown in (1). This is known as Additive Increase phase of the AIMD algorithm.

$$cwnd = cwnd + \frac{1}{cwnd} \tag{1}$$

When congestion is detected either through *dupacks* or Selective Acknowledgement (SACK), *cwnd* is updated as shown in (2). This is known as Multiplicative Decrease phase of the AIMD algorithm.

$$cwnd = \frac{cwnd}{2} \tag{2}$$

DCTCP employs an efficient multiplicative decrease mechanism which reduces the *cwnd* based on the *amount of congestion* in the network rather than reducing it by half. DCTCP leverages Explicit Congestion Notification (ECN) mechanism [11] to extract multi-bit feedback on the *amount of congestion* in the network from the single bit stream of ECN marks. The next subsection describes the working of ECN in brief:

3.3.1 Explicit Congestion Notification (ECN)

Explicit Congestion Notification (ECN) [11] is one of the most popular congestion signaling mechanisms in communication networks. It is widely deployed in a large variety of operating systems at end hosts, modern Internet routers and used by a variety of transport protocols. Moreover, it has been noticed that the use of ECN in the Internet has increased by three folds in the last few years.

As shown in Fig. 4 and 5, ECN uses two bits in the IP header, namely ECN Capable Transport (ECT) and Congestion Experienced (CE), and two bits in the TCP header, namely Congestion Window Reduced (CWR) and ECN Echo (ECE), for signaling congestion to the end-hosts. ECN is an industry standard and its detailed mechanism is described in RFC 3168. Table 3 and 4 show the ECN codepoints in the TCP header and the IP header respectively and Fig. 6 shows in brief, the steps involved in the working of ECN mechanism.

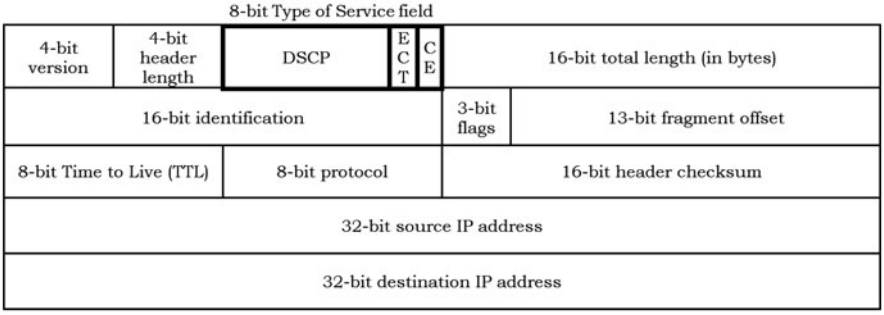

Fig. 4 ECN bits in IP header

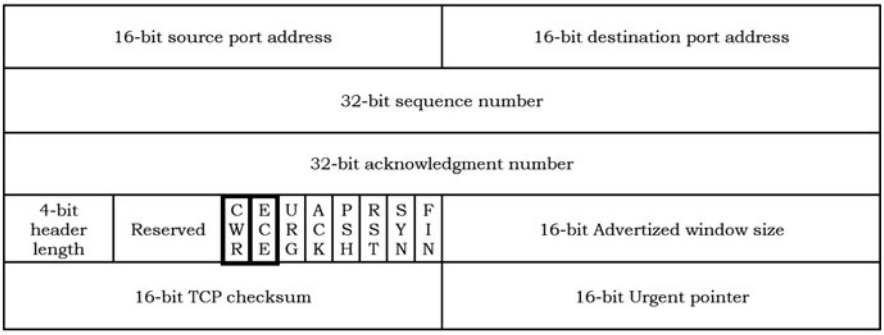

Fig. 5 ECN bits in TCP header

Table 3 ECN codepoints in the TCP header

Codepoint	CWR bit value	ECE bit value
Non ECN-set up	0	0
ECN-Echo	0	1
CWR	1	0
ECN-set up	1	1

Table 4 ECN codepoints in the IP header

Codepoint	ECT bit value	CE bit value
Non-ECT	0	0
ECT(1)	0	1
ECT(0)	1	0
CE	1	1

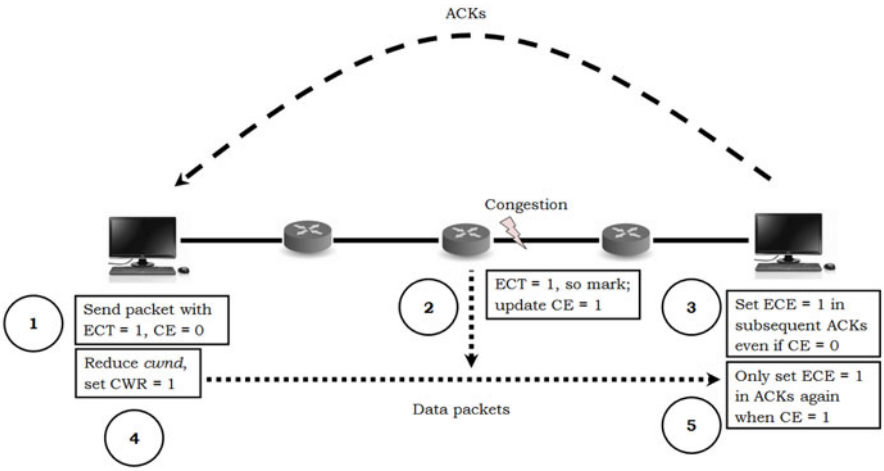

Fig. 6 Explicit congestion notification

As described in RFC 3168—the sender and the receiver must negotiate the use of ECN during the three-way handshake (See Fig. 7). If both are ECN capable, the sender marks every outgoing data packet with either ECT(1) codepoint or ECT(0) codepoint. This serves as an indication to the router that both sender and receiver are ECN capable. Whenever congestion builds up, the router marks the data packet by replacing ECT(1) or ECT(0) codepoint by the CE codepoint. When the receiver receives a marked packet with CE codepoint, it infers congestion and hence, marks *a series of* outgoing acknowledgments (ACKs) with ECE codepoint until the sender acknowledges with CWR codepoint (See Fig. 6).

The major observation here is that, even if the router marks just one data packet, the receiver continues to mark ACKs with ECE until it receives confirmation from the sender (See Step 3 of Fig. 6). This is to ensure the reliability of congestion notification; because even if the first marked ACK is lost, other marked ACKs would notify the sender about congestion. Note that this basic working of ECN aims to only notify the sender about congestion. It is not designed to provide the additional information about the *amount of congestion* to the sender.

At the receiver, counting number of packets marked by the router provides fairly accurate information about the *amount of congestion* in the network. However, conveying this information to the sender by using ECN is a complex task. One of the possible ways is to enable the sender to count the number of marked ACKs it receives from the receiver. The limitation, however, is that even if router marks just one data packet, receiver sends *a series of* marked ACKs. Hence, the number of marked ACKs counted by the sender would be much higher than the number of packets actually marked by the router. This would lead to incorrect estimation of the *amount of congestion* in the network.

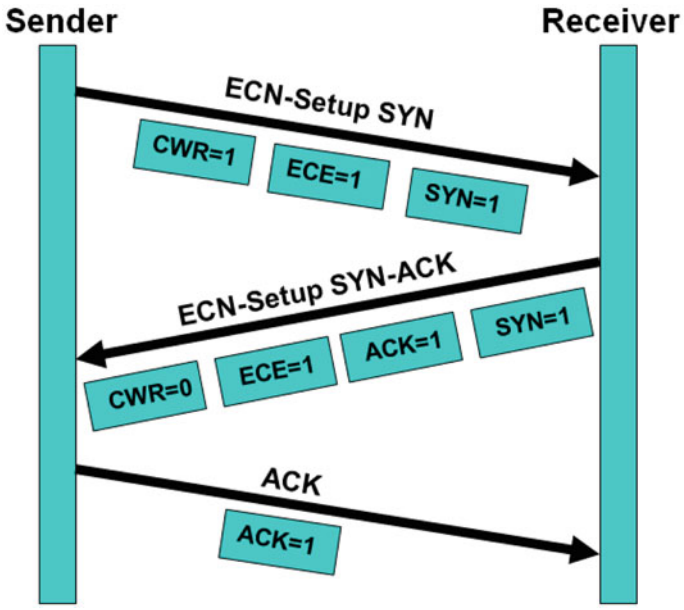

Fig. 7 ECN negotiation

To overcome this limitation, DCTCP modifies the basic mechanism of ECN. Unlike TCP receiver which sends *a series of* marked ACKs, DCTCP receiver sends a marked ACK *only when* it receives a marked packet from the router i.e., it sets ECE codepoint in the outgoing ACK *only when* it receives a packet with CE codepoint. Thus, the DCTCP sender obtains an accurate number of packets marked by the router by simply counting the number of marked ACKs it receives. Note that this modification to the original ECN mechanism reduces the reliability because if a marked ACK is lost, sender remains unaware of the congestion and does not reduce the sending rate. However, since Data Center Networks are privately controlled networks, the possibility that an ACK gets lost is negligible.

On receiving the congestion notification via ECN, the *cwnd* in DCTCP is reduced as shown in (3).

$$cwnd = cwnd \times (1 - \frac{\alpha}{2}) \tag{3}$$

where α ($0 \leqslant \alpha \leqslant 1$) is an estimate of the fraction of packets that are marked and is calculated as shown in (4). F in (4) is the fraction of packets that are marked in the previous *cwnd* and g ($0 < g < 1$) is the exponential weighted moving average constant. Thus, when congestion is low (α is near 0), *cwnd* is reduced slightly and when congestion is high (α is near 1), *cwnd* is reduced by half, just like traditional TCP.

$$\alpha = (1 - g) \times \alpha + g \times F \tag{4}$$

The major goal of DCTCP algorithm is to achieve low latency (desirable for mice traffic), high throughput (desirable for elephant traffic) and high burst tolerance (to avoid packet losses due to Incast). DCTCP achieves these goals by reacting to the *amount of congestion* rather than halving the *cwnd*. DCTCP uses a marking scheme at switches that sets the Congestion Experienced (CE) codepoint [11] in packets as soon as the buffer occupancy exceeds a fixed pre-determined threshold, K (17 % as mentioned in [12]). The DCTCP source reacts by reducing the window by a factor that depends on the fraction of marked packets: the larger the fraction, the bigger the decrease factor.

Advantages: DCTCP is a novel TCP variant which alleviates TCP Incast, Queue-up and Buffer pressure problems in Data Center Networks. It requires minor modifications to the original design of TCP and ECN to achieve these performance benefits. DCTCP employs a *proactive* behavior i.e., it tries to avoid packet loss. It has been shown in [2] that when DCTCP uses FG-RTO, it further reduces the impact of TCP Incast and also improves the scalability of DCTCP. The stability, convergence and fairness properties of DCTCP [12] make it a suitable solution for implementation in Data Center Networks. Moreover, DCTCP is already implemented in latest versions of Microsoft Windows Server operating system.

Shortcomings: The performance of DCTCP falls back to that of TCP when the degree of Incast increases beyond 35 i.e., if there are more than 35 worker nodes sending data to the same aggregator, DCTCP fails to avoid Incast and its performance falls back to that of the traditional TCP. However, authors show that dynamic buffer allocation at the switch and usage of FG-RTO can scale DCTCP's performance to handle up to 40 worker nodes in parallel.

Although DCTCP uses a simple queue management mechanism at the switch, it is ambiguous whether DCTCP can alleviate the problem of TCP Outcast. Similarly, DCTCP does not address the problem of pseudo-congestion effect in Virtualized Data Centers. DCTCP utilizes minimum buffer space in the switches, which in fact, is a desirable property to avoid TCP Outcast. However, experimental studies are required to conclude whether DCTCP can mitigate the problems of TCP Outcast and pseudo-congestion effect.

3.4 ICTCP: Incast Congestion Control for TCP [7]

Like DCTCP, the main idea of ICTCP is to *avoid* packet losses due to congestion rather than recovering from the packet losses. It is well known that a TCP sender can send a minimum of advertised window (*rwnd*) and congestion window (*cwnd*) (i.e. *min(rwnd, cwnd)*). ICTCP leverages this property and efficiently varies the *rwnd* to avoid TCP Incast. The major contributions of ICTCP are: (a) The available bandwidth is used as a quota to co-ordinate the *rwnd* increase of all connections. (b) Per flow congestion control is performed independently and (c) *rwnd* is adjusted based on

the ratio of difference between expected throughput and measured throughput over expected throughput. Moreover, live RTT is used for the throughput estimation.

Advantages: Unlike DCTCP, ICTCP does not require any modifications at the sender side (i.e. worker nodes) or network elements such as routers, switches, etc. Instead, ICTCP requires modification only at the receiver side (i.e. an aggregator). This approach is adopted to retain the backward compatibility and make the algorithm general enough to handle the Incast congestion in future high-bandwidth, low-latency networks.

Shortcomings: Although it has been shown in [7] that ICTCP achieves almost zero timeouts and high throughput, the scalability of ICTCP is a major concern i.e., the capability to handle Incast congestion when there are extremely large number of worker nodes since ICTCP employs per flow congestion control. Another limitation of ICTCP is that it assumes that both the sender and the receiver are under the same switch, which might not be the case always. Moreover, it is not known how much buffer space is utilized by ICTCP. Thus, it is difficult to conclude whether ICTCP can alleviate Queue build-up, Buffer pressure and TCP Outcast problems. Like DCTCP, ICTCP too does not address the problem of pseudo-congestion effect in Virtualized Data Centers.

3.5 IA-TCP: Incast Avoidance Algorithm for TCP [13]

Unlike DCTCP and ICTCP which use window based congestion control, IA-TCP uses rate based congestion control algorithm to control the total number of packets injected in the network. The motivation behind selecting rate based congestion control mechanism is that window based congestion control mechanisms in Data Center Networks have limitations in terms of scalability i.e., number of senders in parallel.

The main idea of IA-TCP is to limit the total number of outstanding data packets in the network so that it does not exceed the bandwidth-delay product (BDP). IA-TCP employs ACK regulation at the receiver and like ICTCP, leverages the advertised window (*rwnd*) field of the TCP header to regulate the *cwnd* of every worker node. The minimum *rwnd* is set to 1 packet. However, if large number of worker nodes send packets with respect to a minimum *rwnd* of 1 packet, the total number of outstanding packets in the network may exceed the link capacity. In such scenarios, IA-TCP adds delay, Δ, to the ACK packet to ensure that the aggregate data rate does not exceed the link capacity. Moreover, IA-TCP also uses delay, Δ, to avoid the synchronization among the worker nodes while sending the data.

Advantages: Like ICTCP, IA-TCP also requires modification only at the receiver side (i.e. an aggregator) and does not require any modifications at the sender or network elements. IA-TCP achieves high throughput and significantly improves the query completion time. Moreover, the scalability of IA-TCP is clearly demonstrated by configuring up to 96 worker nodes sending data in parallel.

Shortcomings: Similar to the problem of ICTCP, it is not clear how much buffer space is utilized by IA-TCP. Thus, experimental studies are required to conclude whether IA-TCP can mitigate Queue build-up, Buffer pressure and TCP Outcast problems. Like DCTCP and ICTCP, studies are required in Virtualized Data Center environments to analyze the performance of IA-TCP with respect to the problem of pseudo-congestion effect.

3.6 D^{2}TCP: Deadline-aware Datacenter TCP [14]

D^{2}TCP is a novel TCP-based transport protocol which is specifically designed to handle high burst situations. Unlike other TCP variants which are deadline-agnostic, D^{2}TCP is deadline-aware. D^{2}TCP uses a *distributed* and *reactive* approach for bandwidth allocation and employs a novel deadline-aware *congestion avoidance* algorithm which uses ECN feedback and deadlines to vary the sender's *cwnd* via a gamma-correction function [14].

D^{2}TCP does not maintain per flow information and instead, inherits the distributed and reactive nature of TCP while adding deadline-awareness to it. Similarly, D^{2}TCP employs its congestion avoidance algorithm by adding deadline-awareness to DCTCP. The main idea, thus, is that far-deadline flows back-off aggressively and the near-deadline flows back-off only a little or not at all.

Advantages: The novelty of D^{2}TCP lies in the fact that it is deadline-aware and reduces the fraction of missed deadlines up to 75 % as compared to DCTCP. In addition, since it is designed upon DCTCP, it avoids TCP Incast, Queue build-up and has high burst tolerance.

Shortcomings: The shortcomings of D^{2}TCP are exactly similar to those of DCTCP: scalability and whether it is robust against TCP Outcast as well as pseudo-congestion effect. However, since it is deadline-aware, it would be interesting to analyze the robustness of D^{2}TCP against the pseudo-congestion effect in Virtualized Data Centers.

3.7 TCP-FITDC [15]

TCP-FITDC is an adaptive delay-based mechanism to *prevent* the problem of TCP Incast. Like D^{2}TCP, TCP-FITDC is also a DCTCP-based TCP variant which benefits from the novel ideas of DCTCP. Apart from utilizing ECN as an indicator of network buffer occupancy and buffer overflow, TCP-FITDC also monitors changes in the queueing delay to estimate variations in the available bandwidth.

If there is no marked ACK received during the RTT, TCP-FITDC infers that the queue length in the switch is below the marking threshold and hence, increases the *cwnd* to improve the throughput. If marked ACKs are received during the RTT,

cwnd is decreased to control the queue length. TCP-FITDC maintains two separate variables called rtt_1 and rtt_2 for unmarked ACKs and marked ACKs respectively. By analyzing the difference between these two types of ACKs, TCP-FITDC gets more accurate estimate of the network conditions. The *cwnd* is then reasonably decreased to maintain low queue length.

Advantages: TCP-FITDC gets a better estimate of the network conditions by coupling the information received via ECN and the information obtained by monitoring the RTT. Thus, it has better scalability than DCTCP and scales up to 45 worker nodes in parallel. It avoids TCP Incast, Queue build-up and has high burst tolerance because it is built upon DCTCP.

Shortcomings: The shortcomings of TCP-FITDC are similar to those of DCTCP, except that it improves the scalability of DCTCP. Unlike D^2TCP, TCP-FITDC is deadline-agnostic and like all above mentioned TCP variants, it does not address TCP Outcast and pseudo-congestion effect problems.

3.8 TDCTCP [16]

TDCTCP attempts to improvise the working of DCTCP (and so, is DCTCP-based) by making three modifications. First, unlike DCTCP, TDCTCP not only decreases, but also increases the *cwnd* based on the *amount of congestion* in the network i.e, instead of increasing the *cwnd* as shown in (1), TDCTCP increases the *cwnd* as shown in (5). Thus, when the network is lightly loaded, the increment in *cwnd* is high; and vice-versa.

$$cwnd = cwnd * \left(1 + \frac{1}{1 + \frac{\alpha}{2}}\right) \tag{5}$$

Second, TDCTCP resets the value of α after every delayed ACK timeout. This is done to ensure that α does not carry the stale information about the network conditions, because if the stale value of α is high, it restricts the *cwnd* increment and thereby degrades the overall throughput. Lastly, TDCTCP employs an efficient approach to dynamically calculate the delayed ACK timeout with a goal to achieve better fairness.

Advantages: TDCTCP achieves 26–37 % better throughput and 15–20 % better fairness than DCTCP in a wide variety of scenarios ranging from single bottleneck topologies to multi-bottleneck topologies and varying buffer sizes. Moreover, it achieves better throughput and fairness even at very low values of K i.e., the ECN marking threshold at the switch. However, there is a slight increase in the delay and queue length while using TDCTCP as compared to DCTCP.

Shortcomings: Although TDCTCP improves throughput and fairness, it does not address the scalability challenges faced by DCTCP. Like most of other TCP variants discussed, TDCTCP too is deadline agnostic and does not alleviate the problems of TCP Outcast and pseudo-congestion effect.

3.9 TCP with Guarantee Important Packets (GIP) [17]

TCP with GIP mainly aims to improve the network performance in terms of goodput by minimizing the total number of timeouts. Timeouts lead to dramatic degradation in the network performance and affect the user perceived delay. The authors of TCP with GIP focus on avoiding mainly two types of timeouts in the network: (i) the timeouts caused due to the loss of full window of packets. These types of timeouts are termed as Full window Loss TimeOuts (FLoss-TOs) and (ii) the timeouts caused due to the lack of ACKs. These types of timeouts are termed as Lack of ACKs TimeOuts (LAck-TOs).

FLoss-TOs generally occur when the total data sent by all the worker nodes exceeds the available bandwidth in the network and thus, a few unlucky flows end up loosing all the packets of the window. On the other hand, LAck-TOs mainly occur when the transmission is *barrier synchronized transmission*. In such transmissions, the aggregator will not request the worker nodes to transmit the next stripe units until all the worker nodes finish sending their current ones. If a few packets get dropped at the end of the stripe unit, they cannot be recovered until the RTO fires because there may not be sufficient *dupacks* to trigger Fast Retransmit.

TCP with GIP introduces *flags* in the interface between the application layer and the transport layer. These *flags* indicate whether the running application follows many-to-one communication pattern or not. If the running application follows such a communication pattern, TCP with GIP redundantly transmits the last packet of the stripe unit at most three times and each worker node decreases its initial *cwnd* at the head of the stripe unit. On the other hand, if the running application does not follow many-to-one communication pattern, TCP with GIP works like a standard TCP.

Advantages: TCP with GIP achieves almost zero timeouts and higher goodput in a wide variety of scenarios including with and without the background UDP traffic. Moreover, the scalability of TCP with GIP is much more than any other TCP variant discussed above i.e., it scales well up to 150 worker nodes in parallel.

Shortcomings: TCP with GIP does not address the queue occupancy problem resulting out of the presence of elephant traffic. As a result, the Queue Buildup, Buffer pressure and TCP Outcast problems remain unsolved because all these problems arise due to the lack of the buffer space in the switches. Although timeouts are eliminated by TCP with GIP, but flows may miss the specified deadlines because of queueing delay. Moreover, the hypervisor scheduling latency is not taken into consideration and thus, the problem of pseudo-congestion effect also remains open. Note that high latencies introduced by hypervisor scheduling algorithm may also prevent flows from meeting the specified deadlines.

3.10 PVTCP: Para Virtualized TCP [5]

PVTCP proposes an efficient solution to the problem of pseudo-congestion effect. This approach does not require any changes to be done in the hypervisor. Instead, the basic working of TCP is modified to accept the latencies introduced by the hypervisor scheduler. An efficient approach is suggested to capture the *actual* picture of every packet transmission involving the hypervisor-introduced-latencies and then determine RTO more accurately to filter out pseudo-congestion effect.

Whenever the hypervisor introduces scheduling latency, sudden spikes can be observed during the regular measurements of RTT. PVTCP detects these sudden spikes and filters out the negative impact of these spikes by proper RTT measurement and RTO management. While calculating average RTT, PVTCP ignores the measurement of a particular RTT if a spike is observed in that RTT.

Advantages: PVTCP solves the problem of pseudo-congestion effect without requiring any changes in the hypervisor. By detecting the unusual spikes, accurately measuring RTT and proper management of RTO, PVTCP enhances the performance of Virtualized Data Centers.

Shortcomings: The scalability of PVTCP is ambiguous and thus, whether it can solve TCP Incast effectively or not is unclear. The queue occupancy while using PVTCP is not taken into consideration which may further lead to problems such Queue build-up, Buffer pressure, TCP Outcast and missed deadlines.

4 Summary: TCP Variants for DCNs

Table 5 summarizes the comparative study of TCP variants proposed for Data Center Networks. Apart from the novelty of the proposed TCP variant, the table also highlights the deployment complexity of each protocol. The protocols which require modifications in sender, receiver and switch are considered as hard to deploy. The ones which require modification only at the sender or receiver are considered as easy to deploy. Data Center Networks, however, are privately controlled and managed networks and thus, the former ones may also be treated as easy to deploy.

Apart from the above mentioned parameters, the summary also includes which problems amongst TCP Incast, TCP Outcast, Queue build-up, Buffer pressure and pseudo-congestion effect are alleviated by each TCP variant. The details regarding the tools used/approach of implementation adopted by the authors are also listed.

Table 5 Summary of TCP Variants proposed for Data Center Networks

TCP Variants proposed for Data Center Networks	Modifies Sender	Modifies Receiver	Modifies Switch	Solves TCP Incast	Solves TCP Outcast	Solves Queue build-up	Solves Buffer pressure	Is Dead-line Aware	Detects pseudo-congestion	Implementation
TCP with FG-RTO	✓	x	x	✓	x	x	x	x	x	Testbed and *ns*-2
TCP with FG-RTO + Delayed ACKs disabled	✓	x	x	✓	x	x	x	x	x	Testbed and *ns*-2
DCTCP	✓	✓	✓	✓	x	✓	✓	x	x	Testbed and *ns*-2
ICTCP	x	✓	x	✓	x	x	x	x	x	Testbed
IA-TCP	x	✓	x	✓	x	x	x	x	x	*ns*-2
D^2TCP	✓	✓	✓	✓	x	✓	✓	✓	x	Testbed and *ns*-3
TCP-FITDC	✓	✓	✓	✓	x	✓	✓	x	x	Modeling and *ns*-2
TDCTCP	✓	✓	✓	✓	x	✓	✓	x	x	OMNeT++
TCP with GIP	x	✓	x	✓	x	x	x	x	x	Testbed and *ns*-2
PVTCP	✓	✓	x	✓	x	x	x	x	✓	Testbed

5 Open Issues

Although several modifications have been proposed to the original design of TCP, there is an acute need to further optimize the performance of TCP in DCNs. A few open issues are listed below:

- Except D^2TCP, all other TCP variants are deadline-agnostic. Meeting deadlines is the most important requirement in DCNs. Missed deadlines may lead to violations of Service Level Agreements (SLAs) and thus, incur high cost to the organization.
- Most of the Data Centers today employ virtualization for efficient resource utilization. Hypervisor scheduling latency ranges from microseconds to hundreds of milliseconds and hence, may hinder in successful completion of flows within the specified deadline. While making modifications to hypervisor is one viable solution, designing an efficient TCP which is deadline-aware and automatically tolerates hypervisor scheduling latency is a preferred solution.
- A convincing solution to TCP Outcast problem is unavailable. An optimal solution to overcome TCP Outcast must ensure minimal buffer occupancy at the switch. Since RED is implemented in most of the modern switches - it can be used to control the buffer occupancy. The parameter sensitivity of RED, however, poses further challenges and complicates the problem.

6 Concluding Remarks

Data Centers in the present scenario house a plethora of Internet applications. These applications are diverse in nature and have various performance requirements. Majority of these applications use many-to-one communication pattern to gain performance efficiency. TCP, which has been a mature transport protocol of Internet since past several decades, suffers from performance impairments such as TCP Incast, TCP Outcast, Queue build-up, Buffer pressure and Pseudo-congestion effect in Data Center Networks.

This chapter described each of the above mentioned impairment in brief along with the causes and possible approaches to mitigate them. Moreover, it presents a comparative study of TCP variants which have been specifically designed for Data Center Networks and the advantages and shortcomings of each TCP variant are highlighted.

References

1. J. Zhang, F. Ren, and C. Lin "Survey on Transport Control in Data Center Networks," *IEEE Network*, vol. 27, no. 4, pp. 22–26, 2013.
2. M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan "Data Center TCP (DCTCP)," *SIGCOMM Computer Communications*

Review, vol. 40, no. 4, pp. 63–74, Aug. 2010. [Online]. Available: http://doi.acm.org/10.1145/1851275.1851192
3. V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller "Safe and effective Fine-grained TCP Retransmissions for Datacenter Communication," *SIGCOMM Computer Communications Review*, vol. 39, no. 4, pp. 303–314, Aug. 2009. [Online]. Available: http://doi.acm.org/10.1145/1594977.1592604
4. P. Prakash, A. Dixit, Y. C. Hu, and R. Kompella "The TCP Outcast Problem: Exposing Unfairness in Data Center Networks," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 30–30. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228339
5. L. Cheng, C.-L. Wang, and F. C. M. Lau "PVTCP: Towards Practical and Effective Congestion Control in Virtualized Datacenters," in *21st IEEE International Conference on Network Protocols*, ser. ICNP 2013. IEEE, 2013.
6. Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph "Understanding TCP Incast Throughput Collapse in Datacenter Networks," in *Proceedings of the 1st ACM workshop on Research on Enterprise Networking*, ser. WREN '09. New York, NY, USA: ACM, 2009, pp. 73–82. [Online]. Available: http://doi.acm.org/10.1145/1592681.1592693
7. H. Wu, Z. Feng, C. Guo, and Y. Zhang "ICTCP: Incast Congestion Control for TCP in Data Center Networks," in *Proceedings of the 6th International Conference*, ser. Co-NEXT '10. New York, NY, USA: ACM, 2010, pp. 13:1–13:12. [Online]. Available: http://doi.acm.org/10.1145/1921168.1921186
8. J. F. Kurose and K. W. Ross, *Computer Networking: A Top Down Approach*.Addison-Wesley, 6th ed., 02/2012, ISBN-13: 978-0-13-285620-1, 2012.
9. Y. Ren, Y. Zhao, P. Liu, K. Dou, and J. Li "A survey on TCP Incast in Data Center Networks," *International Journal of Communication Systems*, pp. n/a–n/a, 2012. [Online]. Available: http://dx.doi.org/10.1002/dac.2402
10. S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, August 1993. [Online]. Available: http://dx.doi.org/10.1109/90.251892
11. K. K. Ramakrishnan and S. Floyd, "The Addition of Explicit Congestion Notification (ECN) to IP," 2001, rFC 3168.
12. M. Alizadeh, A. Javanmard, and B. Prabhakar "Analysis of DCTCP: Stability, Convergence and Fairness," in *Proceedings of the ACM SIGMETRICS, Joint International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '11. New York, NY, USA: ACM, 2011, pp. 73–84. [Online]. Available: http://doi.acm.org/10.1145/1993744.1993753
13. J. Hwang, J. Yoo, and N. Choi "IA-TCP: A Rate Based Incast-Avoidance Algorithm for TCP in Data Center Networks," *ICC 2012*, 2012.
14. B. Vamanan, J. Hasan, and T. Vijaykumar "Deadline-aware Datacenter TCP (D^2TCP)," *SIGCOMM Computer Communications Review*, vol. 42, no. 4, pp. 115–126, Aug. 2012. [Online]. Available: http://doi.acm.org/10.1145/2377677.2377709
15. J. Wen, W. Zhao, J. Zhang, and J. Wang "TCP-FITDC: An Adaptive Approach to TCP Incast Avoidance for Data Center Applications," in *Proceedings of the 2013 International Conference on Computing, Networking and Communications (ICNC)*, ser. ICNC '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 1048–1052. [Online]. Available: http://dx.doi.org/10.1109/ICCNC.2013.6504236
16. T. Das and K. M. Sivalingam, "TCP Improvements for Data Center Networks," in *Communication Systems and Networks (COMSNETS), 2013 Fifth International Conference on*. IEEE, 2013, pp. 1–10.
17. J. Zhang, F. Ren, L. Tang, and C. Lin "Taming TCP Incast Througput Collapse in Data Center Networks," in *21st IEEE International Conference on Network Protocols*, ser. ICNP 2013. IEEE, 2013.