Mitigating TCP Incast Issue in Cloud Data Centres using Software-Defined Networking (SDN): A Survey

Zawar Shah
Whitireia Community Polytechnic
450 Queen Street, Auckland
New Zealand
[e-mail :zawar.shah@whitireia.ac.nz]

Received April 10, 2017; revised June 28, 2017; accepted May 3, 2018; published November 30, 2018

Abstract

Transmission Control Protocol (TCP) is the most widely used protocol in the cloud data centers today. However, cloud data centers using TCP experience many issues as TCP was designed based on the assumption that it would primarily be used in Wide Area Networks (WANs). One of the major issues with TCP in the cloud data centers is the Incast issue. This issue arises because of the many-to-one communication pattern that commonly exists in the modern cloud data centers. In many-to-one communication pattern, multiple senders simultaneously send data to a single receiver. This causes packet loss at the switch buffer which results in TCP throughput collapse that leads to high Flow Completion Time (FCT). Recently, Software-Defined Networking (SDN) has been used by many researchers to mitigate the Incast issue. In this paper, a detailed survey of various SDN based solutions to the Incast issue is carried out. In this survey, various SDN based solutions are classified into four categories i.e. TCP Receive Window based solutions, Tuning TCP Parameters based solutions, Quick Recovery based solutions and Application Layer based solutions. All the solutions are critically evaluated in terms of their principles, advantages, and shortcomings. Another important feature of this survey is to compare various SDN based solutions with respect to different performance metrics e.g. maximum number of concurrent senders supported, calculation of delay at the controller etc. These performance metrics are important for deployment of any SDN based solution in modern cloud data centers. In addition, future research directions are also discussed in this survey that can be explored to design and develop better SDN based solutions to the Incast issue.

Keywords: Software-Defined Networking (SDN), Incast Issue, Cloud Data Centres, Congestion Control, Transmission Control Protocol (TCP).

http://doi.org/10.3837/tiis.2018.11.001 **ISSN : 1976-7277**

1. Introduction

Data centres form the backbone for cloud operators to provide their clients with access to cloud services that range from simple process like online storage to that of computationally sensitive services running on virtual machines. More and more individuals and enterprises are taking advantage of the different cloud service models (i.e. Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS)) that are available to them. The rapid growth in the usage of services provided by cloud computing has led to the creation of new data centres by cloud providers e.g. Amazon plans to establish a new data centre in Paris in 2017 [1]. Virtualization is the foundation on which cloud providers build their business. The hypervisor, also called Virtual Machine Monitor (VMM), is an integral component of virtualization that creates and runs Virtual Machines (VMs) [2]. In cloud data centers, the majority of flows tend to be short, whereas the majority of packets belong to a few long-lived large flows. The short flows known as mice are related to bursty, latency-sensitive applications like search results. The long-lived flows known as elephants are usually large transfers, such as backups or back-end operations [3]. For efficient communication, cloud data centers must employ data transfer techniques and protocols that are not only highly reliable but also capable of maintaining high throughput. The reliable Transmission Control Protocol (TCP) is the widely used transport protocol for data transfer within the cloud data centres [3][4][5][6].

TCP was designed based on the assumption that it would primarily be used in Wide Area Networks (WANs) and is therefore, not optimized to be used in a typical data centre environment that consists of limited-size switch buffers, low propagation delays and high-speed links [3][4]. For example, the minimum Retransmission Timeout (RTO) timer of TCP is set to 200ms which is too large for a data centre environment that has very low Round Trip Times (RTTs) [4][5]. Another major issue with TCP is the fact that it does not take into account the extent of congestion and reduces congestion window to half or to 1 Maximum Segment Size (MSS) as a result of a packet loss [3][7]. These issues with TCP can severely impact the throughput that is needed to perform various tasks inside the cloud data centers. Consequetly, TCP suffers from many issues inside the cloud data centers. One of the major issues with TCP inside the cloud data centers that has been widely discussed in the literature is the Incast issue [3][4][5][6].

Incast issue arises because of the partition/aggregation workflow that is present in the cloud data centers. The aggregator divides the query and sends it to various worker nodes. After resolving the query, worker nodes send packets to the aggregator at the same time which results in a many-to-one communication pattern. The switch (with limited buffer capacity) present between the worker nodes and the aggregator is not able to store all the packets because of this many-to-one communication pattern and this results in packet loss at the switch. The aggregator now has to wait till TCP retransmits the loss packets. This results in TCP throughput collapse which increases the time required to complete the query [3][4][5][6][8]. Many techniques have been proposed in the literature to address the Incast problem. Some solutions to the Incast issue that have been used by many studies in the literature, as a benchmark, are Data Center TCP (DCTCP) [3], Incast Congestion Control TCP (ICTCP) [9], Deadline-Aware Data Center TCP (D^2TCP) [10], Receiver Window Queue (RWNDQ) [11] etc. Recently, Software-Defined Networking (SDN) [12][13] based solutions to mitigate the Incast issue have also been discussed in the literature [6][14][15][16].

In this work, we carry out a detailed survey of various SDN based solutions proposed in

the current literature to mitigate the TCP Incast issue. We classify the various SDN based solutions into four categories i.e. TCP receive window based solutions, Tuning TCP parameters based solutions, Quick recovery based solutions and Application layer based solutions. We discuss principles, working details of all the four types of solutions and then crtically evaluate each of them by discussing their weaknesses. All SDN based solutions are evaluated for various performance metrics that are critical for deployment in the cloud data centers e.g. support of maximum number of concurrent senders, comparison with non-SDN solutions, calculation of delay at the controller etc. Future research directions are also discussed that outline promising investigation areas. We note that few surveys [4][5] have been carried out in the existing literature regarding the TCP incast issue. However, these surveys [4][5] discuss only non-SDN based solutions for TCP Incast issue. To the best of author's knowledge, there is no survey in the existing literature that classifies and critically evaluates various SDN based solutions to the Incast issue.

The main contributions of this work are (i) To categorise various SDN based solutions proposed in the existing literature to mitigate the TCP Incast issue. (ii) To discuss working principles and details of various SDN based solutions proposed in the existing literature to mitigate the TCP Incast issue. (iii) To critically evaluate various categories of SDN based solutions proposed in the existing literature to mitigate the TCP Incast issue. (iv) To compare different SDN based solutions to mitigate Incast issue based on various perfromance metrics. (v) To propose future research directions that can be explored as a source of motivation towards development and deployment of new SDN based solutions for the TCP Incast issue.

The rest of this survey is organized as follows. In section 2, TCP Incast issue and previous surveys carried out regarding TCP Incast issue are discussed. SDN is briefly discussed in section 3. In section 4, four types of SDN based solutions are discussed and critically evaluated. In section 5, comparison of various SDN based solutions in terms of different performance metrics is carried out and future research directions are proposed. Finally, conclusion of the survey is provided in section 6.

2. Background and Related Work

The cloud data centers today consist of large number of traffic flows with different characteristics and requirements. The traffic inside the cloud data center includes short lived, delay sensitive mice flows (e.g. traffic from large scale web applications, distributed fie storage etc) and long lived, delay insensitive, bandwidth sensitive elephant flows (e.g. traffic from backups and VM migration etc) [3]. Mice flows are generated inside the cloud data centers because of applications that consist of many-to-one communication pattern e.g. large scale web applications, distributed fie storage (e.g. Hadoop Distributed File System (HDFS)), data procesing applications (e.g. MapReduce) [3][17][18]. In all these applications, the query is broken down into smaller queries by the aggregator and are sent to worker nodes. The worker nodes are connected to a shallow buffer switch (mostly 3-4 MB memory [19]) which is in turn connected to the aggregator. The aggregator receives all the replies from workers and then combines them into final result. However, sending of infromation by worker nodes to aggregator creates a synchronized many-to-one communication pattern that quickly fills up the buffer at the switch which also has traffic from elephant flows passing through it. This exhaustion of buffer at the switch causes packet loss [3][4][5]. Fast retranmission phase of TCP, that requires sending Tripple duplicate ACKs, also does not trigger due to lack of subsequent packets arriving at the sender. Consequently, TCP at the sender waits for the minimum RTO timer of TCP to expire before loss packets can be retransmitted. RTO timer

of TCP is generally set to 200ms in most operating systems [3][19]. However, this value is suitable for WAN links but not for data centers where RTTs are in the order of microseconds [3][17][18]. This delay leads to underutilization of the network and throughput collapse (called as TCP Incast throughput collapse) which leads to long Flow Completion Time (FCT) for mice flows. The increase in FCT makes the mice flows to miss their deadlines and consequently are unacceptable to the aggregator. This results in bad response quality [3][4][5][9][20]. Studies [19][20][21] suggest that this delay caused by Incast issue can severely impact the revenue of the cloud operator e.g. every 100ms of latency cost Amazon 1% in sales [21], similarly, extra 0.5 seconds in search page generation time dropped the traffic of Google by 20% [21]. The incast scenario is shown in **Fig. 1** [3][4][6].

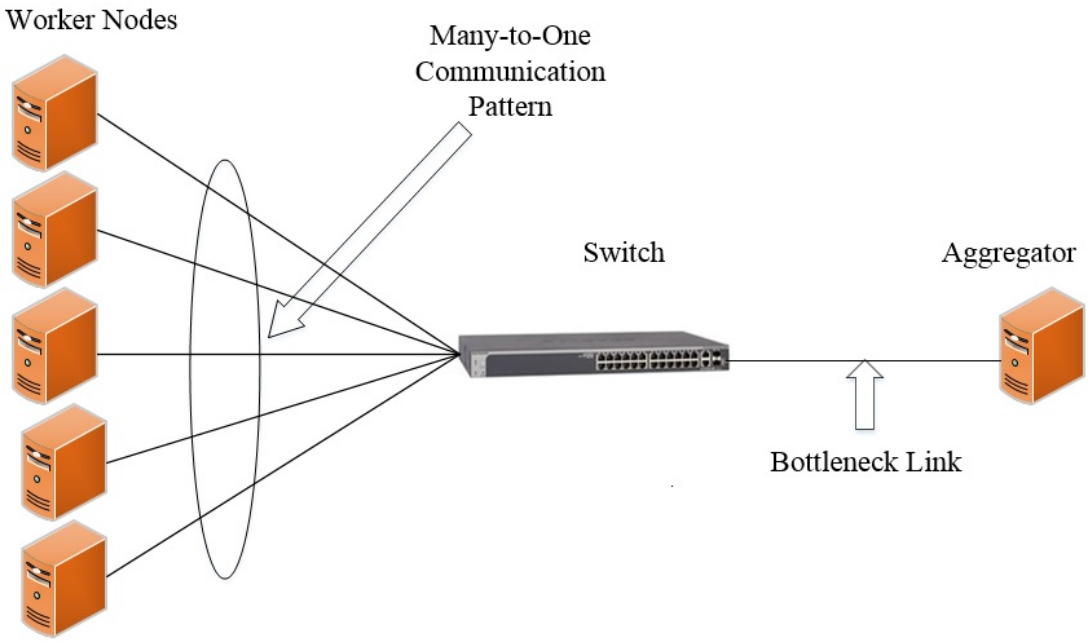

Fig. 1. TCP Incast Scenario.

Some studies [4][5][22][23][24][25] in the current literature have carried out a survey to classify and analyze various non-SDN based solutions to mitigate the Incast issue. In [4] authors categorized non-SDN based solutions from the aspect of different layers of the TCP/IP model i.e. solutions based on link layer, transport layer and application layer. However, authors in [4] do not provide any future research directions in this area. A survey on various issues regarding transport control in data center networks is carried out in [5]. Authors discussed the TCP Incast issue and provided a survey of various non-SDN based solutions to mitigate this issue. In [5], authors classified the non-SDN based solutions in three categories i.e. revising TCP parameters, replacing TCP with other protocols and solving Incast at other layers of the TCP/IP model. Authors briefly proposed and explained few future research directions in this area e.g. to design a lossless transport protocol to eliminate timeouts. Similarly, authors in [22][23] discussed and critically evaluated various non-SDN solutions (e.g. DCTCP etc.) to mitigate the Incast issue in cloud data centers. No future research directions are presented in [22], however, authors in [23] discussed future research directions and proposed the use of SDN in cloud data centers to mitigate the Incast issue. In [23], authors suggested that SDN can be used to tune different TCP parameters in

real-time with respect to the current network state and prevent buffers from queuing up too much data. However, no survey of SDN based solutions to mitigate the Incast issue is carried out in [23]. In [24], a survey on various issues related to network resource management in cloud data centers is carried out. Authors have also briefly explained various solutions to mitigate these issues. The use of SDN to mitigate the Incast issue in cloud data centers is presented in [24]. Authors explained that SDN controller can be used to dynamically update various TCP parameters (e.g. minimum RTO) and this can help in the mitigation of the Incast issue. A survey of existing transport layer solutions proposed for mitigating the TCP Incast issue is also carried out in [25]. In [25], various solutions to TCP Incast (e.g. DCTCP, ICTCP etc.) issue are critically evaluated using different criteria i.e. modifications to the TCP stack, support from the switch, congestion control algorithm etc.

It is noted that no study exists in the current literature that has carried out a detailed survey of various SDN based solutions that have been proposed in the last few years to mitigate the Incast issue. This study aims to fill in this research gap by categorizing and critically evaluating various SDN based solutions. Another novelty of this work is to compare different SDN based solutions in terms of various performance metrics and propose future research directions.

3. Software-Defined Networking (SDN)

SDN is a new approach to computer networks that differs from traditional networking due to the fact that it separates the data and control planes [12][13]. Data plane deals with the logic and tables that help in the forwarding of incoming packets based on characteristics like IP address, Virtual Local Area Networks Identification (VLAN ID) etc. Control plane deals with the protocols, logic and algorithms that are used to program the data plane. In SDN, unlike traditional networks, the control plane is not present on the devices but is moved to a centralized controller. The separation of the control plane and the data plane is realized by means of a well-defined programming interface between the switches and the controller. The controller exercises direct control over the state in the data-plane elements (e.g. switches etc.) via this well-defined Application Programming Interface (API). The most notable example of such an API is OpenFlow. An OpenFlow switch has one or more tables (called as flow table) of packet-handling rules. Each rule matches a subset of the traffic and performs certain actions (e.g. dropping, forwarding etc.) on the traffic. SDN was developed to facilitate innovation and enable simple programmatic control of the network data-path. The separation of the forwarding hardware from the control logic allows easier deployment of new protocols and applications, straightforward network visualization and management [12][13]. SDN based solutions to mitigate the Incast issue in cloud data centers are discussed in the next section.

4. Existing SDN based Solutions

SDN based solutions to mitigate Incast have attracted a lot of attention in the last few years because of the popularity and growth of infrastructure capable of supporting SDN in cloud data centers. In this section, working details of various SDN based solutions are provided. An important feature of our work is to critically evaluate these solutions and highlight the issues in them. We divide the various SDN based solutions discussed in the literature into four broad categories, which are: TCP receive window based solutions, Tuning TCP parameters based solutions, Quick recovery based solutions and Application layer based

solutions. This classification is shown in **Fig. 2.**

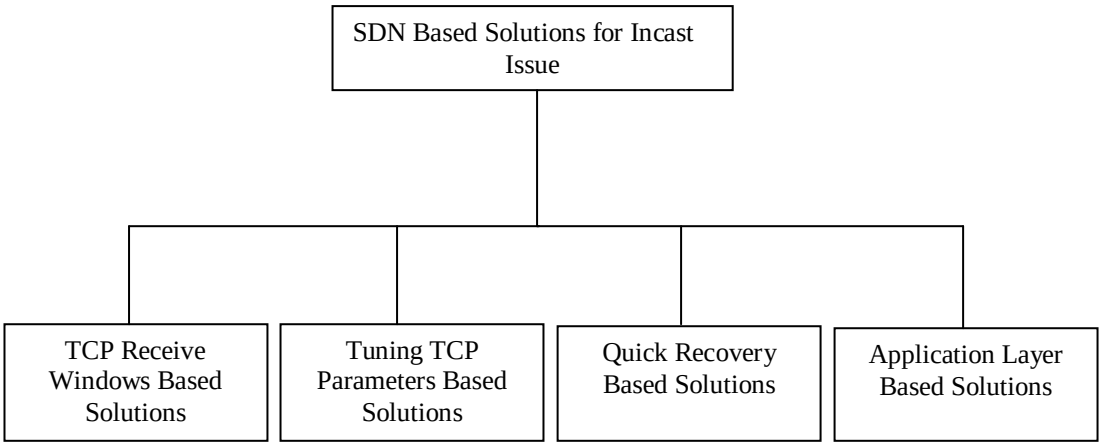

Fig. 2. Classification of SDN based Solutions for Incast Issue

4.1 TCP Receive Window Based Solutions

The key idea of these solutions is to take advantage of the centralized architecture of SDN and use the controller and SDN switch to calculate new sending rate for the senders in the presence of congestion. This new sending rate is added to the receive window field of the TCP ACK header and is sent to the senders to throttle the sending rate. This adjustment in sending rate prevents switch buffers to overflow. The solutions in this category need either the SDN switch or some other entity e.g. the hypervisor to modify the TCP packet. These solutions are discussed below;

4.1.1 Software-Define Network Based TCP (SDTCP)

SDN based TCP (SDTCP) is presented in [6][26] to mitigate the TCP Incast issue. On detecting congestion, this mechanism reduces the sending rate of elephant flows so that mice flows can be accommodated and packet loss can be avoided. The congestion is detected when the queue length of the SDN switch is above the certain threshold and at this instance, the SDN switch sends a congestion notification message to the controller. This notification message depending on the congestion level at the switch can be of three types i.e. low congestion, medium congestion and high congestion. The controller has a flow selection module that has all the details of elephant flows and mice flows. The controller selects all the elephant flows and estimates their current bandwidth. It then reduces the sending rate of these elephant flows depending on the network congestion level as indicated by the SDN switch. For example, the sending rate (*awnd*) of elephant flows determined by the controller on receipt of medium level congestion notification from the switch is given as:

$$awnd_j = \max(\frac{W_{swnd}}{2}, 1MSS), (j \in G) \quad (1)$$

Where G is the number of elephant flows and W_{swnd} is the capacity of bottleneck link that is equally shared by each flow. W_{swnd} is given as:

$$W_{swnd} = \frac{C.RTT + Q(t)}{N} \tag{2}$$

Where C is the capacity of bottleneck link, $Q(t)$ is the queue length at time t and N is the number of elephant and mice flows. The controller then updates the flow table entries at the SDN switch and a new flow table entry called as advertised window (*awnd*) is added which represents the new calculated sending rate. When the TCP ACK packet arriving at the SDN switch matches the *awnd* entries, the *awnd* value of the ACK packet is updated with this new value. The *awnd* is determined by:

$$awnd_i^{'} = \min(awnd_i, awnd_r), (i \in G) \tag{3}$$

where $awnd_r$ = value of *awnd* of TCP ACK packet arriving at the switch
$awnd_i$ = value of *awnd* in the flow table for the *i-th* elephant flow

After receiving the modified ACK packet, the sender adjusts its sending window (*swnd*) based on the following equation

$$swnd = \min(cwnd, awnd) \tag{4}$$

where *cwnd* is the congestion window. Like in basic TCP, *swnd* is normally defined by cwnd. This means that SDTCP temporarily reduces the sending rate of all the elephant flows and in this way the queue at the switch is not filled up to cause packet loss for both elephant and mice flows. Authors in [6][26] carry out experiments to test SDTCP and compare it with TCP and DCTCP. Experiments are carried out by using many-to-one communication topology with a TCP RTO of 30ms. The experimental results show that SDTCP provides high throughput to mice flows and does not starve elephant flows. It also provides lower FCT than both TCP and DCTCP. Experimental results presented in [26] are based on 40 senders and 40 receivers with 1Gbps throughput. Another work [27] compares SDTCP with TCP Reno [28] and TCP Cubic [29] in a smaller network of 10 senders and 10 receivers with lesser capacity of 250Mbps. Authors in [27] found that with smaller network size and lesser capacity, TCP Reno and TCP Cubic provide similar throughput performance as SDTCP.

4.1.2 SDN based Incast Congestion Control via Queue-based Monitoring

The motivation behind the SDN based Incast Congestion Control via Queue-based Monitoring (SICCQ) as presented in [30][31] is to reduce the effect of Incast issue in data centers while simultaneously not effecting the throughput of elephant flows. Another motivation behind SICC is not to modify the basic TCP mechanism on sender/ receiver and the SDN switch. The controller in SICC keeps track of TCP SYN packets of all connections (source destination pair) and a weighted moving average of buffers occupancy at the SDN switch. The controller also keeps track of the new minimum number of extra bytes added by any new connection. If the new connection added can result in buffer overflow then it sends an *Incast-on* message to the hypervisor of the senders. However, if more space is available in the buffer (more than 20% of buffer size) it then sends an *Incast-off* message. The hypervisor keeps track of TCP ACK packets and if *Incast-on* flag is turned on, it then rewrites the receive window value in the ACK packets to 1MSS. On receiving *Incast-off* message, the controller turns off the Incast flag. The setting of receive window to 1MSS reduces the

sending rate of all senders. This prevents packet loss and TCP retransmission timeouts for mice flows (especially at the start of connection when recovery due to packet loss is not possible because of unavailability of Three Duplicate ACKs), therefore, mitigating TCP Incast issue. Incast flag is turned off when buffer space is available and then all the flows restore their data rate by resuming their existing congestion window values. Simulations and experiments carried out in [30][31] show that SICCQ reduces the Incast by improving FCT of mice flows and at the same time its impact on throughput of elephant flows is minimal.

The main contribution of SICCQ is that it does not modify the basic TCP mechanism and the SDN switches that are used in the data center today. However, it does require the hypervisor to be modified to rewrite the ACK packets and keep track of the Incast flag. Moreover, the results presented in [30][31], show that the performance of SICCQ is comparable to DCTCP but worse than RWNDQ. For example, for a single-rooted topology (with elephant to mice flow ratio of 1:3), SICCQ improves the completion time of mice flows on average; it has less variation in response times and its performance is similar to DCTCP. However, it does not perform better than RWNDQ.

4.1.3 Scalable Congestion Control Protocol (SCCP)

The main goal of SCCP [14] is to avoid packet loss via buffer flow even in the presence of large number of flows in the cloud data centers. SCCP keeps track of packets coming in and going out of each port of the SDN switch. For each packet leaving the port, it keeps track of the number of TCP flows (by detecting TCP SYN/FIN flags). For each packet entering the port, it computes the *Fair-Share (FS)* and updates the value of the TCP receive window field in the header only if the new calculated value is less than the old value of receive window. *FS* is defined as the size of each flow that the switch port can support. *FS* is calculated by taking into account the Bandwidth Delay Product (BDP) of the port and the number of flows passing through the port. *FS* on i^{th} switch port is given as:

$$FS_i = \frac{BDP_i}{N_i} \tag{5}$$

where N_i is the number of flows passing through the i^{th} switch port. BDP_i is given as:

$$BDP_i = LinkCapacity_i \times CommonRTT \tag{6}$$

where value of *CommonRTT* is given as 300µs or 400µs [14]. The receive window value in TCP header is updated by each switch in the end-to-end path between the sender and the receiver. In this way, the sender comes to know about the *FS* of the bottleneck port and it can update the sending rate accordingly. The bottleneck port may change with traffic dynamically entering the network. The *FS* is updated accordingly and reported to the senders. This mitigates the Incast issue as packet loss due to the buffer overflow in the bottleneck port is avoided. SCCP has shown significant performance gains than both TCP New Reno and DCTCP. In a single root scenario, it achieve very low FCT (8.4ms) for mice flows even in the presence of 400 workers. In a multiple root scenario, where 90 workers are attached to each of the five roots, it again outperforms both TCP New Reno and DCTCP. Another advantage of SCCP is the fact that it does not maintain per flow information (which can cause significant overhead) but it just keeps record of number of flows (N) passing through each switch port. SCCP, however, does require the SDN switch to do extra processing by checking for the receive window value and then update it with new value, if

required.

4.1.4 SED: An SDN based Explicit Deadline Aware TCP

SED [19] divides the flows into two types i.e. deadline flows and non-deadline flows. Non-deadline flows are given a base sending rate of 1MSS. However, spare bandwidth is given to as many deadline flows as possible. In SED, the SDN switch keeps track of its buffer and sends a congestion notification message to the controller if the buffer size is above a certain threshold. The controller creates a Global Information Flow Table (GIF). This table holds records of many flows like size and identification number of flow, deadline, remaining flow size etc. GIF table is sorted according to the deadline of the flow, flows with earliest deadlines comes first in the table. At the heart of SED is a Receive window determination algorithm that is used by the controller. The value of receive window as determined by this algorithm is placed in the TCP ACK header which is later on used by the TCP senders. The receive window determination algorithm keeps track of total window T_{win} (which is the sum of all sending window sizes of all TCP flows that are passing through the switch at a certain time). T_{win} is given as:

$$T_{win} = \sum_{i \varepsilon N} W_i(t) \tag{7}$$

where *Wi (t)* is the window size of TCP flow *i* at time t.

Another parameter taken into account by this algorithm is the window allocation to a deadline flow. This window allocation is determined as follows:

$$WindowAlloc = \frac{d}{t} \times RTT_{avg} \tag{8}$$

Where *d* is the size of the remaining data that is to be transmitted, *t* is the remaining time until deadline and RTT_{avg} is the average RTT of all flows. The algorithm also keeps track of the total allocation (T_{alloc}) which is incremented after every allocation (both for deadline and non-deadline flows). The algorithm first assigns base sending rate of 1MSS to all the non-deadline flows. After that, if T_{alloc} is less than T_{win}, it allocates window to all the deadline flows according to equation 8. This ensures that all flows meet their deadlines. A flow is dropped if its deadline is missed. If T_{alloc} is less than T_{win} after the initial allocation, then reallocation to non-deadline flows is performed in a fair share manner. SED is compared with TCP, DCTCP and D^{2}TCP in [19]. Experiments carried out in [19] with many-to-one communication pattern with varying number of deadline and non-deadline flows show that SED provides high goodput and low FCT than TCP, DCTCP and D^{2}TCP. However, goodput of SED decreases sharply when number of concurrent senders are more than 40. SED also requires the controller to maintain GIF table, which can cause overhead in the presence of large number of flows.

4.1.5 Issues with TCP Receive Window Based Solutions

All the TCP receive window based solutions either require switch or hypervisor to check for the receive window value in the header of the TCP ACK packet and then update it with new value, if required. This requires modification to existing switches and hypervisors. Moreover, all the solutions, except SCCP, have to maintain per flow information about different flows

present in the network. This can incur significant overhead in the presence of large number of flows. To implement these solutions, the widely used OpenFlow protocol [12][13] also has to be extended to enable the controller and switch to perform various actions e.g. congestion notification trigger sent by the switch to the controller etc. The issues, advantages and working details of these solutions are summarized in **Table 1**. Different software specifications (e.g. SDN controller etc.) used by authors to carry out performance analysis of SDTCP [6][26], SICCQ [30][31], SCCP [14] and SED [19] are also mentioned in **Table 1**.

4.2 Solutions based on Tuning TCP Parameters

Other SDN based solutions discussed in the literature take advantage of the fact that SDN controller can collect many network statistics to align TCP parameters according to the particular data center environment. Some solutions only tune initial TCP parameters [32] while some tune various parameters [15][33]. The fine-tuning of TCP parameters avoids buffer overflow and packet loss at SDN switch. These solutions are discussed below;

4.2.1 Omniscient TCP (OTCP)

Omniscient TCP (OTCP) [15] is a SDN based approach to mitigate the Incast issue by tuning various TCP parameters. OTCP takes into account the global view of the network to collect different network parameters e.g. latency, throughput, buffer size of SDN switch. OTCP measures the end-to-end latency using the OpenFlow Discovery Protocol (OFDP). The buffer size of SDN switch (in bytes) is calculated by the controller by sending an ofp_queue_get_config packet. These measurements of various network parameters are then used to calculate different TCP specific parameters like initial value of *RTO* (RTO_{min}), maximum value of *RTO* (RTO_{max}), initial value of congestion window ($CWND_{init}$) and maximum value of congestion window ($CWND_{max}$). $CWND_{max}$ is calculated by taking into account the BDP of the route between two end systems (S1, S2). $CWND_{max}$ is given as:

$$CWND_{\max}(S1->S2) = BDP_{S1->S2} \tag{9}$$

where *BDP* is calculated by:

$$BDP_{S1->S2} = RTT_{S1->S2}.T_R \tag{10}$$

RTT is the round trip time and T_R is the sending rate between two hosts S1 and S2. Similarly, $CWND_{init}$ is calculated by dividing the $CWND_{max}$ with the number of active flows on the link. $CWND_{init}$ is given as:

$$CWND_{init} = \min(1, \frac{CWND_{\max}}{\beta}) \tag{11}$$

The value of β in equation 11 can be calculated by the controller by sending ofp_flow_stats_request to the SDN switch. The end systems get all the calculated TCP parameters by connecting to the controller. In this way, OTCP aligns the TCP congestion control parameters with the data center environment where it is used. It mitigates Incast by tuning the congestion window to the BDP of the network and this prevents packet loss at the bottleneck switch. OTCP has low FCT compared to TCP when used in a partition/aggregate work flow. For mice flows, it provides 12 times improvement than TCP. However, experiments carried out in [15] also show that in the presence of elephant flows, OTCP

suffers from queue build up and it cannot totally mitigate Incast issue. Consequently, authors have to use it with DCTCP in order to avoid the queue buildup, which ultimately leads to decrease in FCT.

Table 1. Summary of TCP Receive Window Based Solutions

Solution	Working Details	Congestion Control	Issues	Advantages	Software Specifications
SDTCP [6][26]	Controller keeps track of the bandwidth of all elephant flows and uses switch to modify advertised window (*awnd*) in the TCP ACK packets.	Sending rate depends on the congestion level and is calculated by the controller.	Overhead is caused by the maintenance of Per-flow information.	Performs better than TCP and DCTCP in larger networks with high data rate.	Mininet, OpenFlow, FloodLight Controller.
SICCQ [30][31]	*Incast-on* message is sent to the hypervisor by the controller of the senders to indicate congestion.	The hypervisor keeps track of TCP ACK packets and if *Incast-on* flag is turned on, it then rewrites the receive window value in the ACK packets to 1MSS.	Hypervisor needs to be modified to rewrite the ACK packets.	No modification is needed to the TCP and the switches. Low FCT than TCP and DCTCP.	Ryu Controller, OpenFlow.
SCCP [14]	Computes the Fair-Share (FS) for each incoming packet and then update the value of the TCP receive window field in the header.	*FS* is calculated by diving the BDP of the port with the number of flows passing through that port. All switches in the end-to-end path calculate and update FS.	TCP ACK packets have to be modified by the SDN switch.	No per flow information needs to be maintained.	Network Simulator-3 (NS-3), OpenFlow, Open vSwitch controller
SED [19]	Non-deadline flows are given a base sending rate of 1MSS. However, spare bandwidth is given to as many deadline flows as possible.	The deadline flows are allocated window by taking into account the size of the remaining data that is to be transmitted, remaining time until deadline and average RTT of all flows.	Overhead to maintain per flow information in the Global Information Flow (GIF) table.	Performs better than TCP, DCTCP and D^2TCP.	Mininet, OpenFlow, FloodLight Controller.

4.2.2 OpenTCP

Authors in [33] propose OpenTCP which is a TCP adaptation framework that is specially designed to operate in data centers that support SDN. It allows network administrators to tune different TCP parameters according to their policies e.g. congestion control policy. It also allows network administrators to choose between various TCP variants to use in their data center e.g. DCTCP [3], TCP Cubic [29]. In OpenTCP an application called as 'Oracle' runs at the controller. This application collects information about the network from the SDN switch. The 'Oracle' also takes into account the policies defined by the administrator before sending the information to the end hosts via SDN switch. End hosts then either update TCP parameters or choose a variant of TCP to use. Authors in [33] deploy OpenTCP in a real data center as a proof of concept and results show that in the presence of mice flows, OpenTCP can provide better FCT (64% shorter) and less packet drops than TCP when used in a many-to-one communication topology. Also, OpenTCP can be used to take advantage of any TCP variant e.g. DCTCP. OpenTCP, however, is only a framework to reduce Incast and more work on various aspects are needed e.g. designing congestion control policy etc.

4.2.3 Tuning Initial TCP Parameters

Another technique proposed in [32] to mitigate Incast problem is to tune only the initial values of TCP parameters i.e. initial value of *RTO* (RTO_{min}) and initial value of congestion window ($CWND_{init}$). In this approach, the controller calculates the RTO_{min}, $CWND_{init}$ and then sends them to all the senders. The controller calculates RTO_{min} by taking into account the propagation delays and time required by each buffer to offload. RTO_{min} is given as:

$$RTO_{\min} = \sum_{i=1}^{n} L_i + \sum_{i=1}^{n} \frac{B_i}{T_i} \qquad (12)$$

Where L is the propagation delay, B is the buffer size of switch and T is the throughput. Similarly, $CWND_{init}$ is matched to the BDP of the network and is given as:

$$CWND_{init} = \min_{i \varepsilon R} = \frac{T_i}{N_i} \times \sum_{i=1}^{n} L_i \qquad (13)$$

Where N is the number of active TCP flows. Experiments carried out in [32] show that tuning RTO_{min} and $CWND_{init}$ according to the conditions of the data centre environment provides low FCT than using the default values. The results presented in [32] reveal that for mice flows, TCP with optimized RTO_{min} and $CWND_{init}$ values to 14ms and 1 segment, respectively, provide eight times less FCT than with default values of 200ms and 10 segments for RTO_{min} and $CWND_{init}$, respectively. However, further performance evaluation of this mechanism in the presence of elephant flows is needed.

4.2.4 Issues with Solutions based on Tuning TCP Parameters

The benefit of tuning TCP parameters based solutions is that they do not make any modification to the basic TCP mechanism. However, in all these solutions the controller has to collect various network statistics to calculate different network parameters. The controller also has to send the information to all the senders. This requires extra processing on the

controller, which can affect its performance in the presence of large number of flows. The wastage of bandwidth is also an issue when controller sends information to all the senders. Another aspect that needs further investigation is determination of the time it takes for the controller to collect, compute and share the TCP parameters to all the senders. This latency can affect the FCT. Moreover, collecting various network parameters for each TCP flow in these solutions require the switch to maintain a flow table entry for every flow so that TCP statistics are reported to the controller on request. This may not be suitable for SDN switches that have fewer table entries and cannot store information related to all the flows. The issues, advantages and working details of these solutions are summarized in **Table 2.** Different software specifications (e.g. SDN controller etc.) used by authors to carry out performance analysis of OTCP [15], OpenTCP [33] and Tuning Initial TCP Parameters [32] are also mentioned in **Table 2.**

4.3 Quick Recovery based Solutions

In the presence of packet loss, TCP sender waits for the arrival of three duplicate ACKs or the timer to expire before it can retransmit the lost packets. This increases the FCT. In Quick recovery based solutions, lost packets are retransmitted quickly without leaving the communication link idle for a long time. This results in higher throughput and low FCT. More details on these solutions are given below;

4.3.1 Retransmission-enhanced SED (RSED)

RSED is an extension of SED that is discussed in section 4.1.4. The basic idea behind RSED as mentioned by authors in [19] is that packet loss ultimately occurs because the number of flows are very large in a typical data center. Therefore, authors propose that the TCP sender, in the event of loss packets, should retransmit quickly instead of waiting for the timer to expire. In the event of packet loss, the SDN switch encapsulates the dropped packet in an OpenFlow Packet-In message and sends it to the controller. The controller has a GIF table (as mentioned in section 4.1.4) that holds records of many flows like size and identification number of flow, deadline, remaining flow size etc. The controller extracts the dropped packet from the Packet-In message and then sends Triple Duplicate ACKs to the sender (source) of the dropped packet. The switch forwards the Triple Duplicate ACKs to the sender based on the entry in its routing table. The sender on receiving the Triple Duplicate ACKs retransmits quickly without waiting for the timer to expire**.** RSED is compared with SED, TCP, DCTCP and D^2TCP in [19]. Experiments carried out in [19] with many-to-one communication pattern (six senders transmitting flows to one receiver) show that RSED performs better than SED, TCP, DCTCP and D^2TCP. For example, it provides better goodput and low FCT than SED, TCP, DCTCP and D^2TCP even with number of flows equal to 100. However, the issue with RSED is that in the presence of heavy congestion on the switch the Triple Duplicate ACKs sent by the controller may also get lost. The sender will then retransmit only when the timer expires.

Table 2. Summary of Tuning TCP Parameters Based Solutions

Solution	Working Details	Congestion Control	Issues	Advantages	Software Specifications
OTCP [15]	Fine-tunes various TCP parameters e.g. latency, throughput, buffer size of switch.	Tunes the congestion window to the BDP of the route between two systems to prevent packet loss.	Suffers from queue build up in the presence of elephant flows.	Twelve times improvement in FCT than TCP for mice flows. It can also be used with DCTCP to further reduce the FCT.	Mininet, OpenFlow, Go Controller (Specially designed to manage Open vSwitch software switches).
OpenTCP [33]	Different TCP parameters are tuned according to policies and gives choice between various TCP variants to use in the data center.	Defining congestion control policies (e.g. which network parameters to collect etc.) are left on the network operator.	Only a framework to reduce Incast. More work is needed on various aspects e.g. designing congestion control policy etc.	Performs better than TCP in terms of FCT.	No specific controller is mentioned. Oracle (application running at the controller).
Tuning Initial TCP Parameters [32]	Tunes only minimum retransmission timer and initial congestion window.	Initial Congestion window matches the BDP of the network to prevent packet loss.	Extra processing at the controller. More investigation in the presence of elephant flows is needed.	Eight times lower FCT than non-tuned retransmission timer and non-tuned initial congestion window.	Network Simulator-3 (NS-3). No specific controller is mentioned.

4.3.2 Explicit Packet Drop Notification (PDN)

In Explicit PDN technique [34], SDN switch explicitly sends the details of the dropped packet to the sender so that it is retransmitted. When the packet loss occurs in the switch because of congestion, the switch removes few important fields from the packet that are needed to recover the lost data. These important fields are source and destination IP addresses, source and destination port numbers, sequence numbers and length of payload. This data of size 20 bytes is called as Packet Drop Notification Data (PDND). PDND is stored locally by the switch in a separate queue called as notification queue. SDN infrastructure (controller and switches) enables the PDND to reach the sender whose data packet was lost. When the TCP flow in the SDN switches are setup by the controller, all the SDN switches in the network are then also required to create a reverse path so that the PDND reaches the correct sender. PDND is attached to the frame header when any frame to

the next hop that is destined to the same source traverses the switch. In this way, PDND is carried along the path to the original sender of data. The sender on receiving the PDND quickly retransmits the actual packet that was lost. Simulations carried out in [34] show that TCP with PDN provides low FCT even in the presence of large number of senders. The results in [34] show that TCP suffers from packet loss and timeouts (with a delay of 200ms) when number of senders are more than 72. However, in the same scenario, TCP with PDN retransmits quickly with no timeouts and therefore, results in low FCT.

4.3.3 Issues with Quick Recovery based Solutions

Quick recovery based solutions require the SDN switch to keep track of lost packets. This is an overhead for SDN switch and is impractical with large packet loss, which may occur with large number of senders. Both RSED and Explicit PDN mechanisms require the modern SDN switches to be modified. In RSED, SDN switches have to send message to controller when packet loss occurs; this feature is not supported by modern SDN switches. Switches also not support modifying frame header (addition of 20 bytes) and maintaining separate queues for PDND in Explicit PDN mechanism. In RSED, Triple Duplicate ACKs that are sent to the sender results in extra processing by the controller that is already loaded with other tasks to perform in the network.

Both RSED and Explicit PDN presented in [19] and [34], do not determine the total delay it takes for the sender to retransmit when a packet loss occurs. In RSED, this delay consists of notifying the controller of packet loss by switch, generation of Triple Duplicate ACKs by the controller and arrival of Triple Duplicate ACKs at the sender. This delay needs further investigation because if this delay is larger than the RTO timer at the sender then there is no advantage of sending Triple Duplicate ACKs. Similarly, in Explicit PDN mechanism this delay consists of storing PDND in a separate queue, keeping track of frames heading to the direction of the sender, adding twenty bytes to frame header and arrival of PDND to the sender. This delay also needs further study because if this delay is larger than the RTO timer at the sender then there is no advantage of sending Explicit PDN. The issues, advantages, and working details of RSED [19] and Explicit PDN [34] are summarized in **Table 3.** Different software specifications (e.g. SDN controller etc.) used by authors to carry out performance analysis of RSED [19] and Explicit PDN [34] are also mentioned in **Table 3.**

4.4 Application Layer based Solutions

Application layer based solutions either restrict the number of senders who want to send data at the same time [35] or modify the existing SDN architecture by running an application on the controller that helps in decoupling the policy resolution layer from the policy enforcement layer in network service appliances e.g. Firewalls [36]. These solutions do not modify the TCP congestion control mechanism. The details of these solutions are given below;

4.4.1 Send in Group (SIG)

Send in Group (SIG) [35] avoids packet loss and buffer overflow at the switch by not allowing all the senders to transmit data at the same time. It divides the senders into smaller groups and then let the each group transmit one after the other. SDN controller creates the flow table and sends it to the SDN switch, which forwards it to all the senders. When multiple senders want to send data to one receiver simultaneously, the controller invokes the SIG mechanism. It calculates the number of groups N_g of the senders so that the Incast issue is mitigated.

Table 3. Summary of Quick Recovery Based Solutions

Solution	Working Details	Congestion Control	Issues	Advantages	Software Specifications
RSED [19]	Controller sends a Triple Duplicate ACK to the TCP sender so that it can retransmit quickly.	Same as SED.	Generating Triple Duplicate ACK is extra load on the controller.	RSED provides better goodput and low FCT than SED, TCP, DCTCP and D^2TCP.	Mininet, OpenFlow, FloodLight Controller.
Explicit PDN [34]	Switch sends specific details of packet that was lost to the sender so that it can be retransmitted quickly.	Same as TCP.	Modern switches do not support modifying Frame header.	TCP with Explicit PDN provides low FCT than TCP without Explicit PDN in the presence of large number of senders.	Network Simulator-3 (NS-3). No specific controller is mentioned.

If number of senders that simultaneously want to send data to same receiver is N, N_g is then given as:

$$N_g = \left\lceil \frac{N}{K} \right\rceil \tag{14}$$

Where K is adapted from [8], where authors predict the start of Incast in terms of maximum concurrent senders with a specific data block size. Therefore, K is the number of concurrent senders where the network goodput is maximum and any further increase in this number causes the goodput collapse. The value of K as calculated by [8] varies with data block size. The values of K are 13, 7 and 4 for 16KB, 32KB and 64KB data blocks, respectively. The controller chooses K based on number of concurrent senders and data block size. It calculates N_g and then give instructions to the SDN switch. The SDN switch then instructs each sender group and they take turns to send data. SIG avoids packet loss and simulations carried out in [35] show that with 32KB and 64KB data block, TCP throughput collapse did not occur when number of senders are increased from 13 and 7, respectively. The TCP throughput was

also very stable with no fluctuations (because of no packet loss). However, authors in [35] have only considered throughput as a performance metric of SIG. FCT is another important metric that should be discussed and investigated. Moreover, authors carry out simulations for only 30 concurrent senders, which is not a large number. Performance of SIG should be investigated with large number of concurrent senders. SIG is theoretically a good way to prevent packet loss and consequently avoids Incast, however, more research needs to be conducted to prove its efficiency.

4.4.2 EnforSDN

EnforSDN [36] is a novel way of using network service appliances e.g Firewalls, Intrusion Detection Systems (IDSs) and Intrusion Prevention Systems (IPSs) etc. into an environment that uses SDN. In SDN, policy configuration i.e. control decisions are made by a central controller based on the global knowledge and it is delivered to programmable switches that implement policy resolution and enforcement. However, this approach leads to many issues [36] e.g. latency due to processing time at the service appliance, network propagation delay on path towards switches and queuing delay in the forwarding devices that have large traffic passing through them. EnforSDN differs from SDN due to the fact that it decouples the policy resolution layer from the policy enforcement layer in network service appliances. The decoupling of policy resolution and policy enforcement removes the above mentioned shortcomings in SDN resulting in decrease of appliance load and reduction of the network load over the links surrounding the appliances which ultimately leads to mitigation of the Incast issue [36]. An application called *EnforSDN manager* runs on top of the controller that connects with policy resolution instances running on appliances. This communication channel between *EnforSDN manager* and policy resolution instance enables the appliance to inform the *EnforSDN manager* of its policy (which could be blocking, logging, modifying or rate limiting the flow) and also request it to enforce this policy. The *EnforSDN manager* then configures one or more switches to enforce the policy. The appliance can decide which policy to enforce locally and which policy to be enforced remotely. Results presented in [36] show that in a many-to-one communication pattern, EnforSDN reduces the load on the link towards the firewall and mitigates the Incast issue. EnforSDN enabled firewall provides 400%-500% improvement in throughput than regular firewall. However, authors in [36] have only used firewall as an example to implement EnforSDN. More research is needed to extend this mechanism to incorporate other network appliances like IDSs, IPSs etc.

4.4.3 Issues with Application Layer based Solutions

Both SIG [35] and EnforSDN [36] require extra processing at the controller. SIG requires controller to calculate the value of N_g and then send it to the SDN switch. EnforSDN, however, modifies the SDN architecture and it requires an added application (i.e. *EnforSDN manager*) to run on the controller that communicates with the instances running on the network service appliances. The issues, advantages and working details of these solutions are summarized in **Table 4.** Different software specifications (e.g. SDN controller etc.) used by authors to carry out performance analysis of SIG [35] and EnforSDN [36] are also mentioned in **Table 4.**

5. Comparison of Different Solutions and Future Research Directions

In the last section, all SDN based solutions to mitigate TCP Incast issue are discussed and various issues associated with these solutions are highlighted. However, it is also important to compare different solutions proposed in the literature in terms of various performance metrics.

These performance metrics help to determine the usefulness of any solution in terms of deployment in cloud data centers. A comparison of different solutions in terms of various performance metrics along with future research directions are discussed in this section. Performance comparison of various solutions are summarized in **Table 5.**

Table 4. Summary of Application Layer Based Solutions

Solution	Working Details	Congestion Control	Issues	Advantages	Software Specifications
SIG [35]	Divides the senders into smaller groups and then let each group transmit one after the other.	Controls congestion by not allowing all senders to transmit simultaneously.	Supports only 30 concurrent senders. Extra processing at the controller.	Higher throughput than TCP.	NOX Controller, OpenFlow.
EnforSDN [36]	Decouples the policy resolution layer from the policy enforcement layer in network service appliances.	Decoupling of policy resolution layer from the policy enforcement layer reduces the network load over the links surrounding the network appliances.	SDN architecture needs to be modified. Implemented only for firewalls.	EnforSDN enabled firewall provides 400%-500% improvement in throughput than regular firewall.	Mininet, OpenFlow, Open vSwitch controller.

5.1 Maximum Number of Concurrent Senders

An important performance metric is the maximum number of concurrent senders supported by a particular solution before throughput collapses because of the Incast issue. There can be hundreds of concurrent senders inside the data centers that take part in many-to-one communication work flow. In many-to-one communication pattern, increasing the number of concurrent senders increases the onset of throughput collapse because of the Incast issue. Therefore, it is an important metric to consider when evaluating any Incast solution. This survey finds that solutions proposed in the literature do not support a large number of senders, only SCCP [14] can support 400 simultaneous senders while OpenTCP [33] was

implemented in a real data center with thousands of nodes but it is not clear how many concurrent senders it can support. Most of the solutions support less number of concurrent senders e.g. SED [19] and OTCP [15] can support 40 and 100 simultaneous senders, respectively. Future research should be directed in designing solutions that can support large number of concurrent senders.

5.2 Comparison with Non-SDN based Solutions

In order to gauge the performance of all the SDN based solutions, it is important to carry out their performance comparison (in terms of throughput, FCT, number of concurrent senders supported etc.) with other non-SDN based solutions. It is found that majority of the studies have taken the performance of DCTCP [3] as a benchmark to compare the performance of their solutions. However, DCTCP [3] suffers with many issues e.g. it shows poor scalability and cannot cope with more than 35 concurrent senders [4][5]. Other solutions e.g. ICTCP [9] and RWNDQ [11] have shown better performance than DCTCP [3]. Future research work should carry out comparison with solutions other than DCTCP [3] so that the true advantage of using SDN in cloud data centers can be highlighted.

5.3 Comparison with SDN based Solutions

It is interesting to note that none of the existing solutions discussed in section 4 have compared their work with other SDN based solutions. As mentioned above, majority of the solutions have compared their work with non-SDN based solutions (i.e. DCTCP [3]). However, in future performance comparison with other SDN based solutions should also be carried out.

5.4 Number of Mice and Elephant Flows

Another important performance metric is the number of mice and elephant flows considered by the authors when they evaluate their respective solutions. Studies [3][17][37] suggest that elephant and mice flows are in the ratio of 1:3 in a typical cloud data center. We consider this ratio and evaluate various solutions. It is found that only [30][31] have considered this ratio (with 126 mice flows and 42 elephant flows) when they evaluate the performance of SICCQ. Most of the studies have either not considered elephant flows or have only considered one elephant flow, which are not representatives of a typical cloud data center. Future SDN based solutions to mitigate Incast issue should be evaluated by taking elephant and mice flows in the ratio of 1:3.

5.5 Calculation of Delay at the Controller

Critical evaluation of all the four types of solutions in section 4 reveal that SDN controller plays an important role in implementing any SDN based solution. It performs various tasks to implement these solutions e.g. it runs congestion control algorithms in TCP receive window based solutions, calculates various TCP parameters in Tuning TCP parameters based solutions etc. All these tasks require extra processing by the controller and this can affect its performance. It is therefore, important to calculate the time the controller takes to perform various tasks in a particular SDN based solution. However, it can be noted from **Table 5** that only authors in [26] and [33] have calculated the delay at the controller for

SDTCP and OpenTCP, respectively. Rest of the solutions have not considered this important performance metric. Future research in this area should consider calculating delay at the controller in performing various tasks.

5.6 Fairness between Mice and Elephant flows

Another important performance metric is the fairness between mice and elephant flows. Elephant flows and mice flows co-exist in a typical cloud data center. However, a particular solution may allocate more data rate to mice flows and this can starve the elephant flows in the long run. It is therefore, important to determine whether this metric (fairness) is taken into
account by various solutions discussed in section 4. It can be noted from **Table 5** that few solutions (e.g. SDTCP [6][26], SICCQ [30][31], SCCP [14] and OTCP [15]) have considered this performance metric and results show all these four solutions provide fairness in allocating data rate to mice and elephant flows. Most of the studies have ignored this metric when evaluating their solutions. Future research in this area should also consider this metric when evaluating the proposed solution.
Survey results clearly indicate that all the four types of solutions suffer from various issues that make them impractical to be deployed in a cloud data center. From **Table 5**, it is also noted that TCP receive window based solutions like SDTCP [6][26], SCCP [14], SICCQ [30][31] and Tuning TCP based solution like OTCP [15], OpenTCP [33] have the potential to be deployed in a real cloud data center as these solutions address most of the performance metrics discussed above. However, all these solutions still need further development and performance evaluation before they can be deployed in a cloud data center. The research directions given in this section will enable future researchers to address the shortcomings in the current SDN based solutions discussed in the literature. It will also help them to design SDN based solutions to mitigate Incast issue by taking into account various performance metrics that are required for deployment in cloud data centers.

6. Conclusion

In this survey, various SDN based solutions proposed in the existing literature to mitigate the TCP Incast issue are classified into four types i.e. TCP receive window based solutions, Tuning TCP parameters based solutions, Quick recovery based solutions and Application layer based solutions. The working principles, advantages and issues associated with all the solutions are discussed in detail. All the solutions are also critically evaluated and compared in terms of various performance metrics that are important for deployment in cloud data centres. It is noted that all the SDN based solutions suffer from various issues; and need more development and performance evaluation before they can be deployed in cloud data centres. Various research directions are also proposed in this survey that will enable future researchers to propose better SDN based solutions to mitigate the Incast problem.

Table 5. Comparison of Different SDN Based Solutions

Taxonomy of SDN Based Solutions	Solution	Maximum Number of Concurrent Senders	Comparison with Non-SDN Solutions	Number of Mice and Elephant Flows	Delay at the Controller	Fairness between Various Flows
TCP Receive Window Based Solutions	SDTCP [6][26]	120	DCTCP	119 mice,1 elephant	Yes	Yes
	SICCQ [30][31]	145	DCTCP, RWNDQ	126 mice, 42 elephant	No	Yes
	SCCP [14]	400	DCTCP	400 mice, 1 elephant	N/A	Yes
	SED [19]	40	DCTCP, D^2TCP	5 mice, 1 elephant	No	No
Tuning TCP Parameters Based Solutions	OTCP [15]	100	DCTCP	90 mice, one elephant	No	Yes
	OpenTCP [33]	Not Clearly Mentioned (3,864 total nodes in SciNet HPC data center).	No	Mix of both (Not Mentioned clearly)	Yes	No
	Tuning Initial TCP Parameters [32]	10	No	10 mice, 0 elephant	No	No
Quick Recovery Based Solutions	RSED [19]	100	DCTCP, D^2TCP	5 mice, 1 elephant	No	No
	Explicit PDN [34]	150	No	150 mice, 0 elephant	No	No
Application Layer Based Solutions	SIG [35]	30	No	30 mice, 0 elephant	No	No
	EnforSDN [36]	32	No	Mice and Elephant flows are randomly distributed	No	No

References

[1] New AWS Region in France. Retrieved on May 5, 2018. Article (CrossRef Link)

[2] Farzad Sabahi, "Secure Virtualization for Cloud Environment Using Hypervisor-based Technology," *International Journal of Machine Learning and Computing*, vol. 2, no. 1, pp. 39, 2012. Article (CrossRef Link)

[3] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta and Murari Sridharan, "Data Center TCP (DCTCP)," in *Proc. of ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 63-74, 2010. Article (CrossRef Link)

[4] Yongmao Ren, Yu Zhao, Pei Liu, Ke Dou and Jun Li, "A Survey on TCP Incast in Data Center Networks," *International Journal of Communication Systems*, vol. 27, no. 8, pp.1160-1172, 2014. Article (CrossRef Link)

[5] Jiao Zhang, Fengyuan Ren, and Chuang Lin, "Survey on Transport Control in Data Center Networks," *IEEE Network*, vol. 27, no. 4, pp. 22-26, 2013. Article (CrossRef Link)

[6] Lu Yifei, Zhen Ling, Shuhong Zhu and Ling Tang, "SDTCP: Towards Datacenter TCP Congestion Control with SDN for IoT Applications," *Sensors*, Vol.17, no. 1, 2017. Article (CrossRef Link)

[7] J. F. Kurose and K. W. Ross, "Computer Networking: A Top-Down Approach Featuring the Internet," *3rd Edition, Pearson Education*, Inc., 2005.

[8] Yanpei Chen, Rean Griffit, David Zats and Randy H. Katz, "Understanding TCP Incast and Its Implications for Big Data Workloads," *University of California at Berkeley, Technical Report*, 2012. Article (CrossRef Link)

[9] Haitao Wu, Zhenqian Feng, Chuanxiong Guo and Yongguang Zhang, "ICTCP: Incast Congestion Control for TCP in Data-Center Networks," *IEEE/ACM Transactions on Networking*, vol. 21, no. 2 pp. 345-358, 2013. Article (CrossRef Link)

[10] Balajee Vamanan, Jahangir Hasan and T. N. Vijaykumar, "Deadline Aware Data Center TCP (D^2TCP)," in *Proc. of ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 115-126, 2012. Article (CrossRef Link)

[11] Ahmed M Abdelmoniem and Brahim Bensaou, "Efficient Switch-assisted Congestion Control for Data Centers: An Implementation and Evaluation," in *Proc. of Proceedings IEEE International Performance Computing and Communications Conference (IPCCC)*, pp. 1-8, 2015. Article (CrossRef Link)

[12] Bruno Astuto A. Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka and Thierry Turletti, "A Survey of Software-defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 3, pp. 1617-1634, 2014. Article (CrossRef Link)

[13] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky and Steve Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp.14-76, 2015. Article (CrossRef Link)

[14] Jaehyun Hwang, Joon Yoo, Sang-Hun Lee and Hyun-Wook Jin, "Scalable Congestion Control Protocol Based on SDN in Data Center Networks," in *Proc. of IEEE Global Communications Conference (GLOBECOM)*, pp. 1-6, 2015. Article (CrossRef Link)

[15] Simon Jouet, Colin Perkins and Dimitrios Pezaros, "OTCP: SDN-Managed Congestion Control for Data Center Networks," in *Proc. of IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pp. 171-179, 2016. Article (CrossRef Link)

[16] Chunghan Lee, Yukihiro Nakagawa, Kazuki Hyoudou, Shinji Kobayashi, Osamu Shiraki and Takeshi Shimizu, "Flow-Aware Congestion Control to Improve Throughput under TCP Incast in Datacenter Networks," in *Proc. of IEEE Computer Software and Applications Conference*, vol. 3, pp. 155-162, 2015. Article (CrossRef Link)

[17] Theophilus Benson, Aditya Akella and David A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild," in *Proc. of Proceedings of the ACM SIGCOMM Conference on Internet measurement*, pp. 267-280, 2010. Article (CrossRef Link)

[18] Gill Phillipa, Navendu Jain and Nachiappan Nagappan, "Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications," in *Proc. of ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 350-361, 2011. Article (CrossRef Link)
[19] Yifei Lu, "SED: An SDN-based explicit-deadline-aware TCP for cloud Data Center Networks," *Tsinghua Science and Technology, vol.* 21, no. 5, pp. 491-499, 2016. Article (CrossRef Link)
[20] Christo Wilson, Hitesh Ballani, Thomas Karagiannis and Ant Rowtron, "Better Never than Late: Meeting Deadlines in Datacenter Networks," in *Proc. of ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 50-61, 2011. Article (CrossRef Link)
[21] Cost of Latency. Retrieved on May 5, 2018. Article (CrossRef Link)
[22] B. Thiruvenkatama and Mukesh Krishnana, "Survey on TCP Incast Problem in Datacenter Networks," *International Journal of Control Theory and Applications*, vol. 9, no. 60, 2016. Article (CrossRef Link)
[23] Fung Po Tso, Simon Jouet, and Dimitrios P. Pezaros, "Network and Server Resource Management Strategies for Data Centre Infrastructures: A Survey," *Computer Networks*, vol. 106, pp. 209-225, 2016. Article (CrossRef Link)
[24] Guan Xu, Jun Yang, and Bin Dai, "Challenges and Opportunities on Network Resource Management in DCN with SDN," in *Proc. of IEEE International Conference on Big Data*, pp. 1785-1790, 2015. Article (CrossRef Link)
[25] Prasanthi Sreekumari and Jaeil Jung, "Transport Protocols for Data Center Networks: A Survey of Issues, Solutions and Challenges," *Photonic Network Communications*, vol. 31, no. 1, pp. 112-128, 2016. Article (CrossRef Link)
[26] Lu Yifei and Shuhong Zhu, "SDN-based TCP Congestion Control in Data Center Networks," in *Proc. of IEEE International Performance Computing and Communications Conference (IPCCC)*, pp. 1-7, 2015. Article (CrossRef Link)
[27] James Roberts, Johnny Skandalakis, Richard Foard and Jason Choi, "A Comparison of SDN based TCP Congestion Control with TCP Reno and CUBIC," *Technical Report*, 2016. Article (CrossRef Link)
[28] Jitendra Padhye, Victor Firoiu, Donald F. Towsley and James F. Kurose, "Modeling TCP Reno Performance: A Simple Model and its Empirical Validation," *IEEE/ACM Transactions on Networking (ToN)*, vol. 8, no. 2, pp. 133-145, 2000. Article (CrossRef Link)
[29] Sangtae Ha, Injong Rhee and Lisong Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64-74, 2008. Article (CrossRef Link)
[30] Ahmed M Abdelmoniem, and Brahim Bensaou, "SDN-based Incast Congestion Control Framework for Data Centers: Implementation and Evaluation," *Technical Report*, Hong Kong University of Sciences and Technology, 2016. Article (CrossRef Link)
[31] Ahmed M. Abdelmoniem, Brahim Bensaou and Amuda James Abu, "Mitigating Incast-TCP Congestion in Data Centers with SDN," *International Annals of Telecommunications Journal, Springer - Special Issue on Cloud Communications and Networking*, 2017. Article (CrossRef Link)
[32] Simon Jouet and Dimitrios P. Pezaros, "Measurement-based TCP Parameter Tuning in Cloud Data Centers," in *Proc. of IEEE International Conference on Network Protocols (ICNP),* pp. 1-3, 2013. Article (CrossRef Link)
[33] Monia Ghobadi, Soheil Hassas Yeganeh and Yashar Ganjali, "Rethinking End-to-End Congestion Control in Software Defined Networks," in *Proc. of Proceedings of the ACM Workshop on Hot Topics in Networks*, pp. 61-66, 2012. Article (CrossRef Link)
[34] Yu Xia, Ting Wang, Zhiyang Su and Mounir Hamdi, "Preventing Passive TCP Timeouts in Data Center Networks with Packet Drop Notification," in *Proc. of International Conference on Cloud Networking (CloudNet)*, pp. 173-178, 2014. Article (CrossRef Link)
[35] Jianhua Xu, Hongxiang Guo, Jian Wu, Jintong Lin, DongXu Zhang, Gang Chen, Xingping Zhang and Chao Chen, "SIG: Solution to TCP Incast in SDN Network Based Openflow Protocol," in *Proc. of Asia Communications and Photonics Conference*, pp. AW4I-5. Optical Society of America, 2013. Article (CrossRef Link)

[36] Yaniv Ben-Itzhak, Katherine Barabash, Rami Cohen, Anna Levin and Eran Raichstein, "EnforSDN: Network Policies Enforcement with SDN," in *Proc. of IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 80-88, 2015. Article (CrossRef Link)
[37] Ahmed M. Abdelmoniem and Brahim Bensaou, "Reconciling Mice and Elephants in Data Center Networks," in *Proc. of IEEE International Conference on Cloud Networking (CloudNet)*, pp. 119-124, 2015. Article (CrossRef Link)

Zawar Shah completed his M.EngSc and PhD degree in Telecommunications from the University of New South Wales (UNSW), Sydney, Australia in 2004 and 2009, respectively. He was the head of Telecommunications and Computer Networks group at School of Electrical Engineering and Computer Science (SEECS), which is a constituent college of National University of Sciences and Technology (NUST), Pakistan. He is currently working as a Senior Lecturer at Whitireia Community Polytechnic, Auckland, New Zealand. His research interests include Quality of Service issues in Wireless Networks, Software Defined Networking, Cloud Computing, Network Architectures and Protocols.