Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2019) 27: 2068 – 2080
© TÜBİTAK
doi:10.3906/elk-1712-232

Queue length feedback-based solution of TCP Incast in data center networks

Hasnain AHMED[*], **Muhammad Junaid ARSHAD**
Department of Computer Science & Engineering, University of Engineering and Technology, Lahore, Pakistan

Received: 18.12.2017 • **Accepted/Published Online:** 15.11.2018 • **Final Version:** 15.05.2019

Abstract: The Internet offers a large number of applications and services that we use on a daily basis. These widely used applications are hosted on large-scale, high-performance computing systems called data centers. The performance of TCP is inefficient in many-to-one communication, which is a common traffic pattern in data center networks. This many-to-one communication causes significant packet losses followed by timeouts, which consequently results in throughput collapse in data center networks; this problem is known as TCP Incast. In this paper, we present a queue length feedback-based solution to mitigate TCP Incast. The scheme has two parts: i) a novel queue length-based congestion parameter, which accurately measures congestion along the path from source to destination, and ii) a congestion control scheme that effectively uses the new congestion parameter to prevent throughput collapse due to Incast traffic patterns. Results are compared with TCP and DCTCP, the two most common transport protocols deployed in data center networks. The results show that the proposed scheme minimizes packet drops and achieves high utilization and burst tolerance.

Key words: Data center networks, TCP, Incast, many-to-one communication, queue length, explicit congestion notification

1. Introduction

The number of Internet users has grown to hundreds of millions, all around the world, and is increasing. The Internet offers a large number of services; a few popular services are e-mail, Web searching, audio/video hosting and streaming, text/audio/video chatting, providing storage and computation resources, and social networking. These services are hosted on large-scale, high-performance computing systems called data centers. A data center, in simple terms, is a large collection of servers, interconnected through switches and routers, providing computation, storage, and distribution services for huge volumes of data. Data center networks (DCNs) provide the communication infrastructure for today's high-performance parallel computing and current Internet applications [1–5].

All the popular Web applications and services that are used daily by millions of people around the world are hosted on large-scale data center networks. The performance of these Web applications depends on the performance of the underlying DCN. These services may lose revenue of millions of dollars due to even a small lapse in performance. A delay of 100 ms costs Amazon 1% in sales. An extra half of a second in generating responses to Google searches dropped traffic by 20% [5–8]. Therefore, optimized performance in terms of latency and throughput is the most important requirement for these services.

Many applications in DCNs use a many-to-one communication pattern, in which many servers send data to one client in units of data blocks and no server can transmit the next block of data until all the servers have

[*]Correspondence: tohasnain@yahoo.com

completely transmitted the current data block. With the increase in the number of concurrent senders, the data transfer workload can exceed the buffer capacity at the bottleneck switch, resulting in packet losses and subsequent timeouts and retransmissions. This problem is known as TCP Incast and it results in significant throughput degradation [9–11]. TCP Incast is also known as barrier synchronized transmission. Figure 1 illustrates a TCP Incast scenario. Here servers are interconnected through a leaf-spine topology. L_1 to L_4 are leaf switches and S_1 to S_3 are spine switches, and each server (or client) is directly connected to a leaf switch. There are two many-to-one traffic flows: i) servers S_{11}, S_{12}, ..., S_{15} are sending data to client C_1, and ii) servers S_{21}, S_{22}, ..., S_{27} are sending data to client C_2. The width (weight) of the arrow depicts the number of flows (i.e. a thin arrow represents a single flow and a thicker arrow represents a higher number of flows). As shown in the figure, synchronized traffic from servers S_{11}, S_{12}, ..., S_{15} competes for the bottleneck link L_1-C_1 and consequently buffer overflow occurs at the output port associated with the link. Similarly, traffic from servers S_{11}, S_{12}, S_{13}, S_{21}, S_{22}, ..., S_{27} competes for the bottleneck link S_2-L_1 and as a result buffer overflow occurs at the output port associated with the link. The packet losses result in RTOs and retransmissions, which in turn cause drastic throughput degradation.

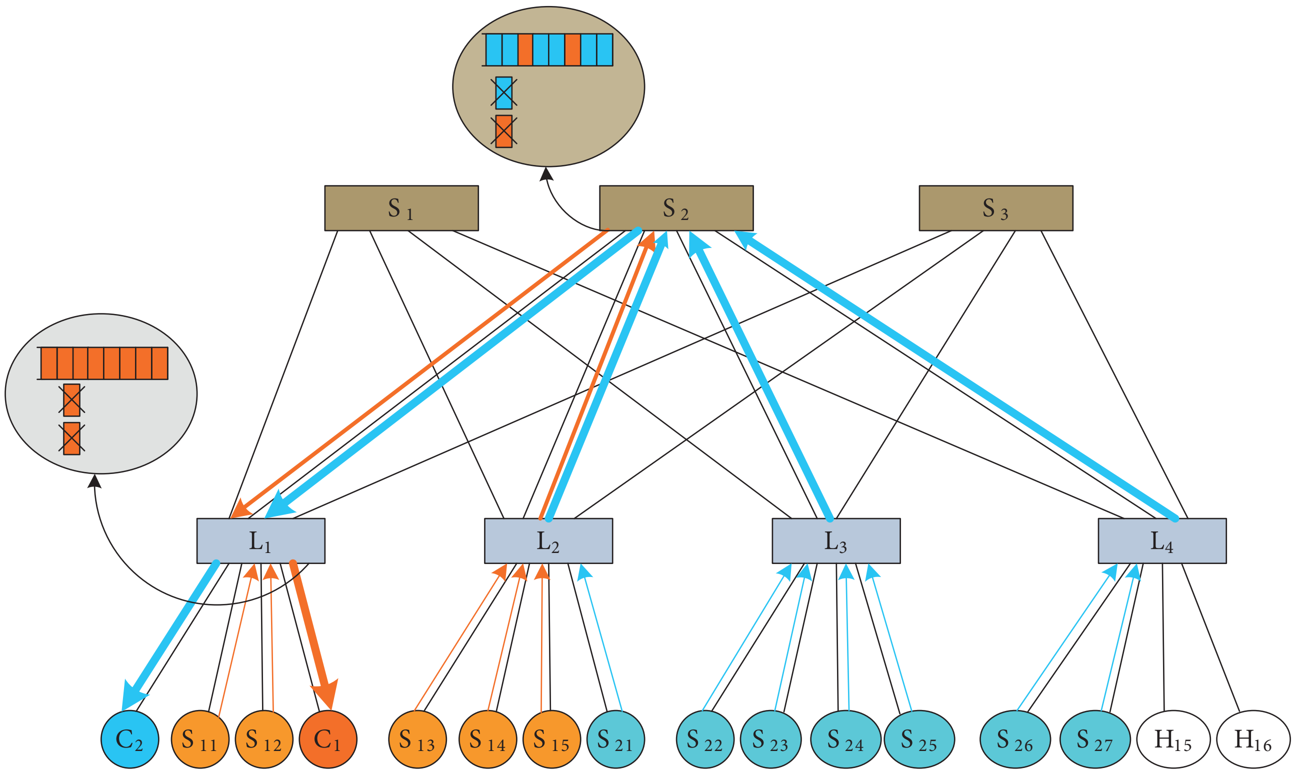

Figure 1. TCP Incast scenario in data center networks.

2. Related work

Many solutions have been proposed by researchers to solve TCP Incast. These solutions can be grouped on the basis of the underlying approach used. Some [12,13] used the ECN marking mechanism for congestion notification. The ECN mechanism only informs about whether the queue length is above a certain level or not; it does not tell about the 'extent' of congestion or rate of increase in queue length and thus ECN-based packet drop prevention schemes are not very effective. Other works [14,15] proposed centralized schemes. Centralized schemes are quite effective in small-scale networks, but in large-scale cloud/cloud service data center networks, where thousands of requests are processed simultaneously, centralized schemes can cause bottlenecks and single points of failure. Some authors [16,17] proposed solutions that require certain information from the application like beforehand information of flow size and number of concurrent senders/receivers. This requires changes in existing applications. Others [18–20] described switching hardware-based solutions. Switch fabric-based schemes can be most accurate and effective, since switching hardware can track the number of current flows and traffic load, and this information can be used to precisely adjust the sending rates. However, these schemes require changes in the switch fabric for their operation, which is quite an expensive option. Another study [21] suggested an SDN-based Incast congestion control framework. SDN's centralized platform enables access to network traffic information at flow-level granularity. This information can be used to carry out precise traffic engineering [22], but this scheme requires the data center to be SDN-enabled, which limits the domain of the solution. One work [23] proposed a TCP pacing-based solution to mitigate TCP Incast. The scheme uses RTT for TCP pacing, but RTTs in data center networks are on the scale of a few hundred microseconds and some studies [11,24] suggested that the current operating systems do not support fine-grained (i.e. microsecond level) timers. Thus, RTT-based schemes are not very applicable in current data center networks.

In this paper, a queue length feedback (QLF)-based solution is proposed to alleviate the TCP Incast problem that is caused due to many-to-one communication patterns. The solution has two components: i) a novel queue length-based congestion parameter, and ii) an end-host-based congestion control scheme that uses the new congestion parameter. The proposed queue length-based congestion parameter gives a precise measure of congestion. The congestion control scheme effectively utilizes the congestion parameter to minimize packet drops and achieve high utilization and burst tolerance. Results are compared with TCP and DCTCP, the two most widely deployed DCN transport protocols. The results show that the proposed scheme effectively prevents Incast throughput collapse.

3. Materials and methods

In this section, we first give the design rationale and then discuss the proposed scheme in detail.

3.1. Design rationale

The objective here is to formulate a new parameter for measuring congestion along the path, which should be more informative and effective than ECN marking, and then use that parameter in a congestion control scheme at the end-host. The congestion control scheme is aimed to: i) minimize the chances of packet drops, ii) achieve high utilization, and iii) ensure a certain degree of burst tolerance.

3.1.1. Queue length feedback (QLF)

The ECN (explicit congestion notification) marking is the basic congestion notification approach used in many transport solutions, most notably DCTCP. The ECN marking scheme checks whether the queue length is above a certain predefined level or not; if it is above the level, the ECN bit is marked in the packet and the packet is forwarded [25]. The scheme only tells whether the queue length is above a certain level or not but does not give the exact measure of queue length. Having explicit information of the exact value of queue length at the bottleneck link can give precise information about congestion along the path and enable much better control over minimizing packet drops. In such a scheme, the sender's congestion window can be increased or decreased proportionally based on the value of the queue length at the bottleneck link. Large-scale DCNs mostly opt for commodity switches for interconnecting the servers. These commodity switches have buffer sizes in the range 64 KB to 512 KB associated with each output port. In addition, the typical maximum transmission unit in DCNs is 1500 bytes. Therefore, the lower order 10 or 11 bits of queue length can be ignored as they do not give much information about queue length in terms of packets. Thus, it is the higher order bits (i.e. all the bits higher than the lower order 10 or 11 bits) that give us queue length on the scale of packets. This queue length can be written in a transport header field of the data packet and propagated to the sender in the ACK packet.

3.1.2. Queue length-based congestion control

QLF, received at the sender, can be very effective in measuring congestion along the path from source to destination; this information can be used to minimize the probability of packet drops and at the same time achieve high utilization. The idea here is to control the traffic rate so as to keep the queue length at a certain level (desired queue length). The value of desired queue length is chosen in such a way that it should simultaneously ensure a certain degree of burst tolerance without packet drops and high utilization. The technique here is to increase or decrease traffic rate so as to bring the current queue length to the desired queue length.

Inflight data have two parts: i) data (packets) in queues, and ii) data travelling along links (inflight packets). Each flow has one bottleneck link (the link having highest value of queue length along the path).

Let
$C_b =$ buffer capacity of output port at bottleneck link
$L_{cr} =$ current queue length at output port of bottleneck link
$L_A =$ allowed (desired) queue length at output port of bottleneck link
$N =$ number of flows having the same bottleneck link
$BDP_i =$ bandwidth-delay product of path of flow i
$D_i =$ current value of amount of inflight data of flow i
Objective I: High utilization
For high utilization: $L_A \geq$ MAX $(BDP_1, BDP_2, ..., BDP_N)$
Objective II: To bring L_{cr} to L_A

Case 1 $L_{cr} < L_A$

If current queue length is less than allowed queue length then the flows having the same bottleneck link may increase $(L_A - L_{cr})$ *the amount of inflight data, or each flow may increase* $D_i \times \frac{L_A - L_{cr}}{L_{cr}}$ *amount of data*

Proof Queue length with D_i amount of data $= L_{cr}$

Queue length with $D_i\left(1+\frac{L_A-L_{cr}}{L_{cr}}\right)$ amount of data $= \frac{L_{cr}}{D_i} \times D_i\left(1+\frac{L_A-L_{cr}}{L_{cr}}\right) = L_A$ □

Case 2 $L_{cr} > L_A$

If current queue length is greater than allowed queue length then the flows having the same bottleneck link should decrease $(L_{cr} - L_A)$ *the amount of inflight data, or each flow may decrease* $D_i \times \frac{L_{cr} - L_A}{L_{cr}}$ *amount of data*

Proof Queue length with D_i amount of data $= L_{cr}$

Queue length with $D_i\left(1-\frac{L_{cr}-L_A}{L_{cr}}\right)$ amount of data $=\frac{L_{cr}}{D_i}\times D_i\left(1-\frac{L_{cr}-L_A}{L_{cr}}\right)=L_A$

Degree of burst tolerance $=C_b-L_A$ □

3.2. Queue length feedback-based scheme

We give a QLF-based solution to the Incast throughput collapse problem in DCNs. The objectives of this scheme are: i) to avoid packet drops (and consequently RTOs), ii) to maintain high utilization, and iii) to provide a certain degree of burst tolerance. The first objective is achieved by controlling and adjusting the congestion window according to the degree of congestion along the path, i.e. decreasing the sender's congestion window if there is high congestion along the path to avoid packet drops. The second objective is achieved by maintaining the allowed queue length greater than or equal to the delay bandwidth product of links along the path. The third objective is achieved by keeping current queue length to approach allowed queue length; this gives a degree of burst tolerance equal to buffer capacity (max. queue size) minus allowed queue length.

The scheme has two parts: collecting the feedback of queue lengths of output ports at switches along the path from source to destination, and adjusting the sender's congestion window (C_w) based on the received feedback of queue lengths '*R_qlf*'. The scheme makes a few changes in the TCP protocol: i) an eight-bit QLF field is added in the TCP header of the data packet, ii) the QLF field in the transport header is updated at switches along the path from source to destination, iii) the receiver receives the value in the QLF field and sends it back to the sender in a similar field in the ACK, iv) the sender receives the value in the QLF field sent in the ACK and adjusts its congestion window '*C_w*' based on the value of '*R_qlf*'. The proposed scheme allows a sender to continue sending data even when its congestion window goes below 1 MSS (maximum segment size); this can happen when there are too many concurrent flows competing for the same bottleneck link. The two parts of the solution are discussed in detail subsequently.

3.2.1. Collecting queue length feedback (QLF)

The QLF field is updated along the path from source edge switch to destination edge as follows: the switches along the path compare the QLF field in the packet with the higher order bits (i.e. bits higher than the lower order 10 bits) of queue length of the output port, overwrite the former with the latter if the latter is greater, and forward the packet. Figure 2 illustrates the process of QLF collection. In the figure, L_1 to L_4 are leaf switches, S_1 to S_3 are spine switches, and H_1 to H_{16} are servers connected to leaf switches. Traffic is flowing from H_4 to H_{16} and H_6 to H_{13}. Queue lengths of output ports are shown along their respective uplinks. The queue length of the output port of the S_3-L_4 link at spine switch S_3 is depicted as higher than others due to two flows competing for the link. The mechanism of QLF collection is illustrated from source H_4 to destination H_{16}: initially, the QLF field in the transport header (TH) of the data packet contains 0 as shown, leaf switch L_1 updates the field before placing the packet in the queue of the output port, spine switch S_3 updates the QLF field, and finally leaf switch L_4 compares the QLF field in the data packet with the queue length of the output port and does not overwrite the QLF field since the latter is not greater than the value in the QLF field of the packet. H_{16} simply sends back to H_4 the value in QLF field in the ACK packet.

3.2.2. Sender's congestion window '*C_w*' calculation

The sender's congestion window '*C_w*' is calculated using the received QLF '*R_qlf*'. In addition, two constants are maintained: i) '*AL*' (allowed limit for *R_qlf*) and ii) '*ML*' (maximum limit for *R_qlf*). '*AL*' is the reference

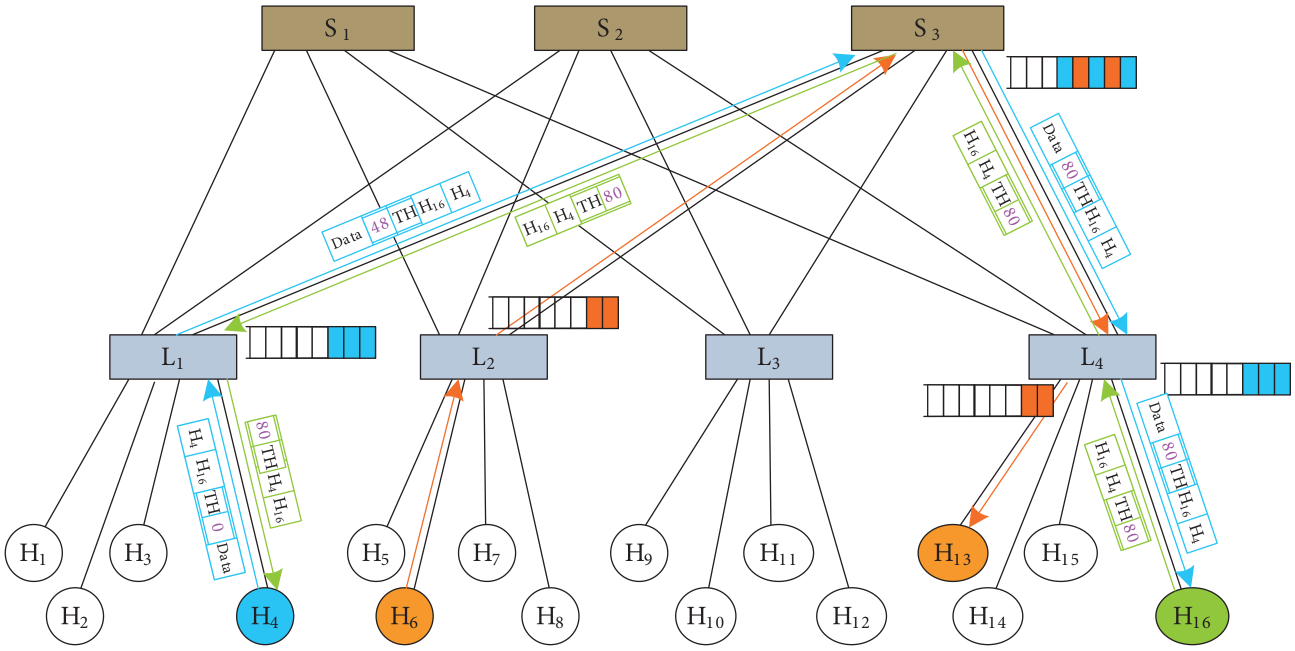

Figure 2. Process of collecting the queue length feedback (QLF).

value to increase or decrease '*C_w*' when '*C_w*' is greater than or equal to 1 MSS and '*ML*' is the reference value to increase or decrease '*C_w*' when '*C_w*' is below 1 MSS (i.e. too many concurrent flows competing for the same bottleneck link). Let '*BUFFER_SIZE*' be the buffer size at an output port of switch. '*BUFFER*_SIZE' minus '*AL*' tells about the degree of burst tolerance. '*C_w*' is calculated as shown in Algorithm 1 and the flowchart of Figure 3. In brief terms, the principle is to bring the current queue length at bottleneck link '*R_qlf*' to the desired level (i.e. '*AL*' if '*C_w*' is greater than or equal to 1 MSS and '*ML*' if '*C_w*' is less than 1 MSS).

3.2.3. Overhead of QLF-based scheme

The proposed scheme adds one byte to the 20-byte TCP header (i.e. both TCP data packet header and acknowledgment header). The overhead in a network of typical 1500-byte packet size is as follow:

Overhead in data packet = 1 / 1459 = 0.0685%

Overhead in acknowledgement = 1 / 40 = 2.5%

4. Results and discussion

The proposed scheme is implemented in network simulator ns-2 and compared with DCTCP and ECN-enabled TCP. The proposed scheme is called QLF-TCP in the results.

4.1. Topology and workload

The details of the topology and other parameters are as follows: i) a leaf-spine topology is created having two spine switches, each connected with three leaf switches using 40-Gbps duplex links, and 40 servers are directly connected with each leaf switch using 10-Gbps duplex links; ii) the buffer size at the output port of the switch

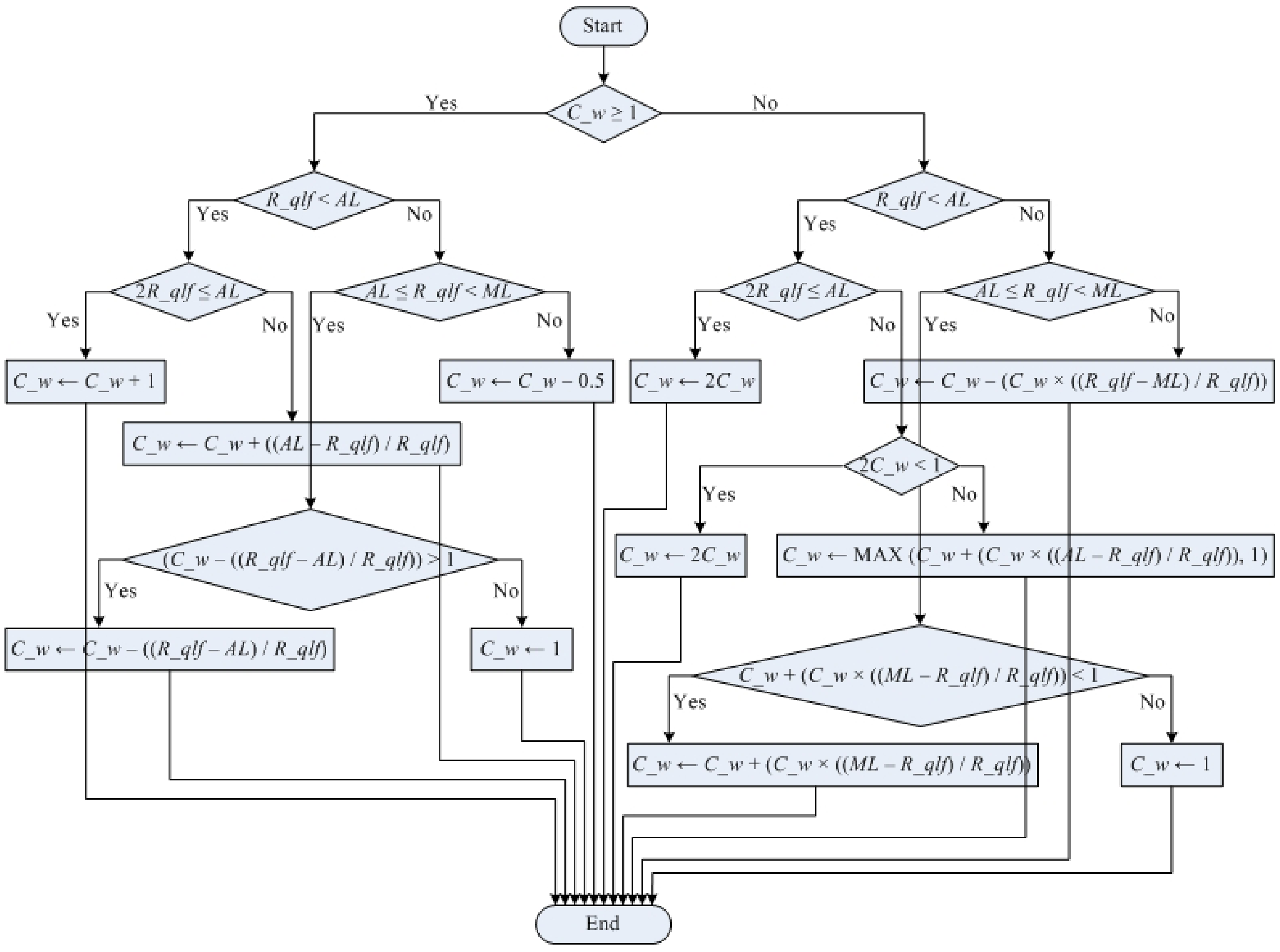
Start
C_w ≥ 1
Yes
No
R_qlf < AL
2R_qlf ≤ AL
AL ≤ R_qlf < ML
C_w ← C_w + 1
C_w ← C_w + ((AL – R_qlf) / R_qlf)
C_w ← C_w – 0.5
(C_w – ((R_qlf – AL) / R_qlf)) > 1
C_w ← C_w – ((R_qlf – AL) / R_qlf)
C_w ← 1
C_w ← 2C_w
C_w ← C_w – (C_w × ((R_qlf – ML) / R_qlf))
2C_w < 1
C_w ← MAX (C_w + (C_w × ((AL – R_qlf) / R_qlf)), 1)
C_w + (C_w × ((ML – R_qlf) / R_qlf)) < 1
C_w ← C_w + (C_w × ((ML – R_qlf) / R_qlf))
End

Figure 3. Flowchart of congestion window calculation.

is taken to be 128 KB, which makes a queue limit of ~87 packets (packet size = 1500 bytes); iii) '*AL*' and '*ML*' values for the proposed scheme are set to 30 and 60, respectively; iv) similarly, '*K*' and '*thresh*' values for DCTCP and ECN-enabled TCP respectively are taken as 30; v) '*minRTO*' for all the schemes is set to 1 ms.

Four different workloads are simulated on each of the three schemes and their results are compared. The four workloads are: a) 80 short flows each of 10 KB, b) 80 short flows each of 50 KB, c) two long flows (each 1 MB) and 50 short flows each of 10 KB, d) two long flows (each 1 MB) and 50 short flows each of 50 KB. Figures 4 and 5 show goodput graphs of the flows for the first two workloads, respectively. Figures 6 and 7 show goodput graphs of the short flows and long flows, respectively, of the third workload. Similarly, Figures 8 and 9 show goodput graphs of the short flows and long flows, respectively, of the fourth workload.

4.2. Observations

1. The QLF-based scheme closely tracks queue length at the bottleneck link and promptly adjusts sending rates in the event of high values of queue length, thus minimizing the chances of packet drops. When the queue length is below the desired level, the scheme quickly and proportionally increases sending rates, thus maintaining high utilization. Since congestion windows are adjusted on the basis of received QLF, the

Algorithm 1 CONGESTION_WINDOW_CALCULATION

```
Input:  R_qlf        ▷ Received value of QLF
        C_w          ▷ Congestion window
        AL ← 30      ▷ Allowed limit for R_qlf
        ML ← 60      ▷ Maximum limit for R_qlf
Output: C_w
1:  IF $R\_qlf < AL$
2:      IF $(R\_qlf \times 2) \leq AL$
3:          IF $C\_w \geq 1$
4:              $C\_w \leftarrow C\_w + 1$
5:          ELSE
6:              $C\_w \leftarrow C\_w \times 2$
7:      ELSE
8:          IF $C\_w < 1$
9:              IF $(C\_w \times 2) \leq 1$
10:                 $C\_w \leftarrow C\_w \times 2$
11:             ELSE
12:                 $C\_w \leftarrow$ MAX $(C\_w + (C\_w \times ((AL - R\_qlf) / R\_qlf)), 1)$
13:         ELSE
14:             $C\_w \leftarrow C\_w + ((AL - R\_qlf) / R\_qlf)$
15: ELSE
16:     IF $(R\_qlf < ML)$ AND $(R\_qlf \geq AL)$
17:         IF $C\_w > 1$
18:             IF $(C\_w - ((R\_qlf - AL) / R\_qlf)) > 1$
19:                 $C\_w \leftarrow C\_w - ((R\_qlf - AL) / R\_qlf)$
20:             ELSE
21:                 $C\_w \leftarrow 1$
22:         ELSE
23:             IF $C\_w < 1$
24:                 IF $(C\_w + (C\_w \times ((ML - R\_qlf) / R\_qlf))) < 1$
25:                     $C\_w \leftarrow C\_w + (C\_w \times ((ML - R\_qlf) / R\_qlf))$
26:                 ELSE
27:                     $C\_w \leftarrow 1$
28:     ELSE
29:         IF $R\_qlf \geq ML$
30:             IF $C\_w > 1$
31:                 $C\_w \leftarrow C\_w - 0.5$
32:             ELSE
33:                 $C\_w \leftarrow C\_w - (C\_w \times ((R\_qlf - ML) / R\_qlf))$
```

competing flows receive a fair share of available bandwidth; hence, all flows achieve almost the same/equal throughput. In addition, the QLF-based congestion control scheme anticipates a burst of new flows; hence, when such a burst of new flows arrives, the scheme accommodates it without causing packet drops by promptly adjusting the sending rates of all the flows having the same bottleneck link. DCTCP and ECN-enabled TCP are ECN-based schemes and, as discussed in previous sections, the ECN marking scheme has no clue of the extent of congestion. In addition, TCP and DCTCP do not anticipate for a burst of new flows, and when such a burst arrives, both schemes experience packet drops and RTOs. Since congestion

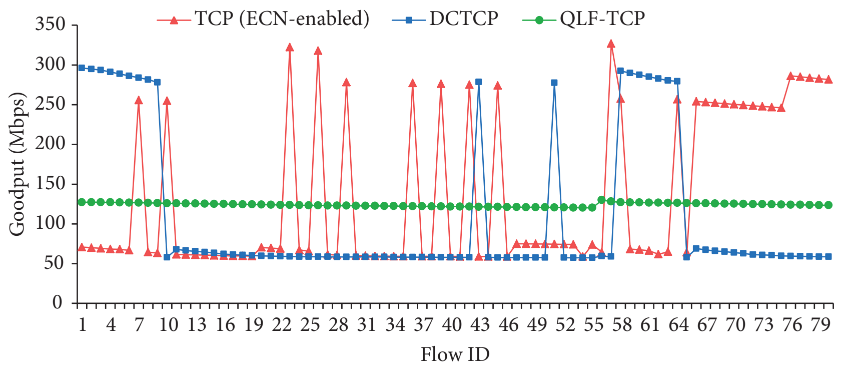

Figure 4. Eighty short flows, each 10 KB.

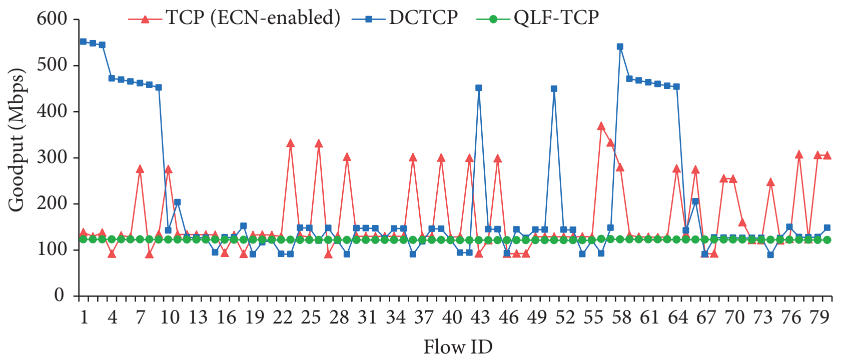

Figure 5. Eighty short flows, each 50 KB.

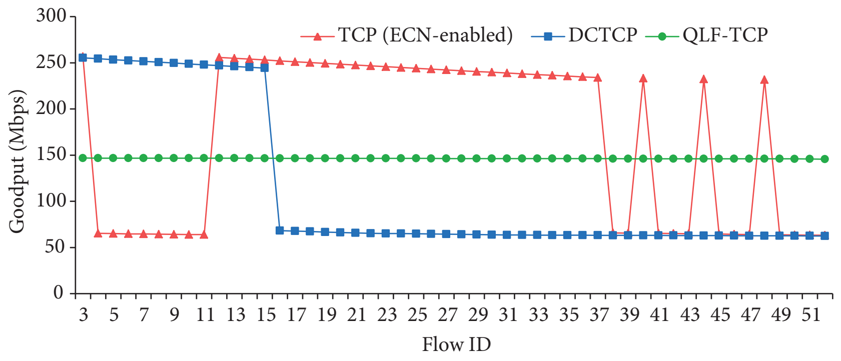

Figure 6. Fifty short flows, each 10 KB.

windows of flows in all four workloads are not big enough to trigger a fast retransmit mechanism, the number of RTOs is equal to the number of packet drops. Table 1 shows the number of packet drops for each of the four workloads. Thus, the throughput of flows that experience RTOs significantly degrades as shown in the goodput graphs of all four workloads.

2. In DCTCP and ECN-enabled TCP, when packet drops occurred simultaneously in many flows, these flows were blocked for the duration of the timeout period, which allowed the rest of the flows to increase their throughput during that period. This is why some DCTCP and ECN-enabled TCP flows achieved very high throughput in all the four workloads as shown in Figures 4, 5, 6, and 8.

3. Surprisingly, ECN-enabled TCP performs slightly better than DCTCP. This is because TCP halves the congestion window if the ECN bit of a packet sent in the previous RTT has been marked, thus resulting

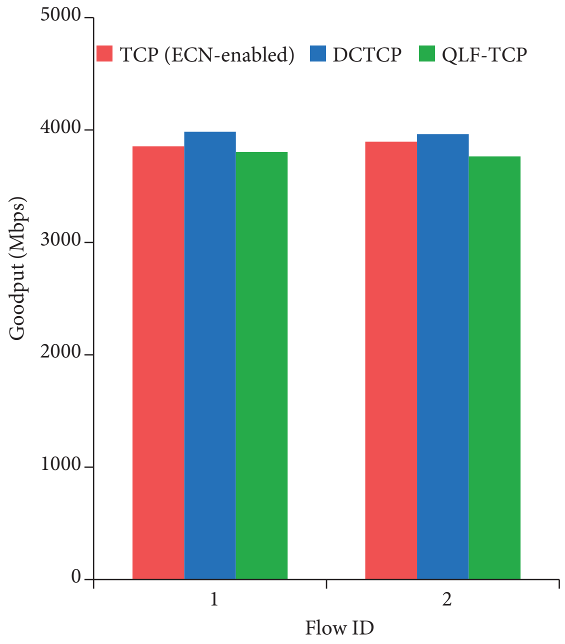

Figure 7. Two long flows, each 1 MB.

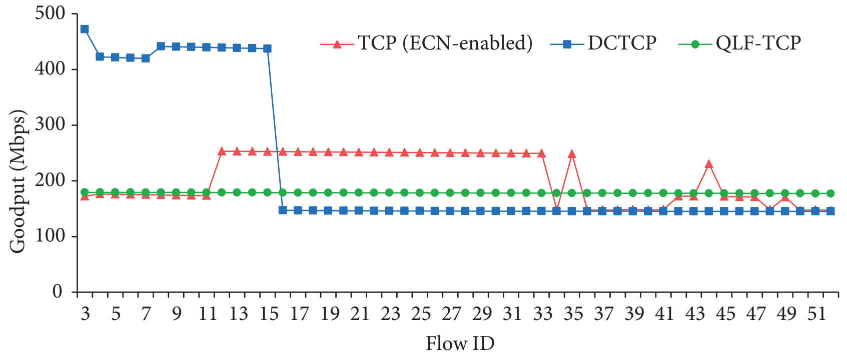

Figure 8. Fifty short flows, each 50 KB.

Table 1. Number of packet drops.

	Workload			
	1	2	3	4
TCP	52	77	20	28
DCTCP	62	95	37	37
QLF-TCP	0	0	0	0

in fewer packet drops and consequently fewer retransmission timeouts than in the case of DCTCP, which reduces the congestion window proportional to the fraction of packets marked.

4. There is not much difference in throughput of long flows in the three schemes. In the 3rd and 4th workloads, the two long flows achieve almost equal goodput for all the three schemes as shown in Figures 7 and 9.

5. In many-to-one barrier synchronized communication, data block transfer time (DBTT) is the time from the start of transmission of the first server to the time of completion of transmission of the last finishing server. DBTT depends on the completion time of the flow that finished last among all the flows of the many-to-one barrier synchronized task. The flow that achieves the lowest throughput for a scheme is the main determinant of DBTT for that scheme. For example, in the goodput graph of Figure 4, flow number 44 achieves the lowest goodput for ECN-enabled TCP, flow number 41 gets the lowest goodput for DCTCP, and flow number 55 achieves the lowest goodput for the proposed QLF-based scheme. Figure 10 shows the DBTT for each of the four workloads. In the 3rd and 4th workloads, since long flows are not part of the many-to-one barrier synchronized task, they are not included in the calculation of DBTT. It is shown in the graph of Figure 10 that, for all four workloads, the QLF-based scheme significantly improves DBTT compared to the other two schemes. Table 2 shows the percentage of improvement in DBTT for all four workloads. Overall, the QLF-based scheme improves DBTT by 37.92% and 38.89% over ECN-enabled TCP and DCTCP, respectively.

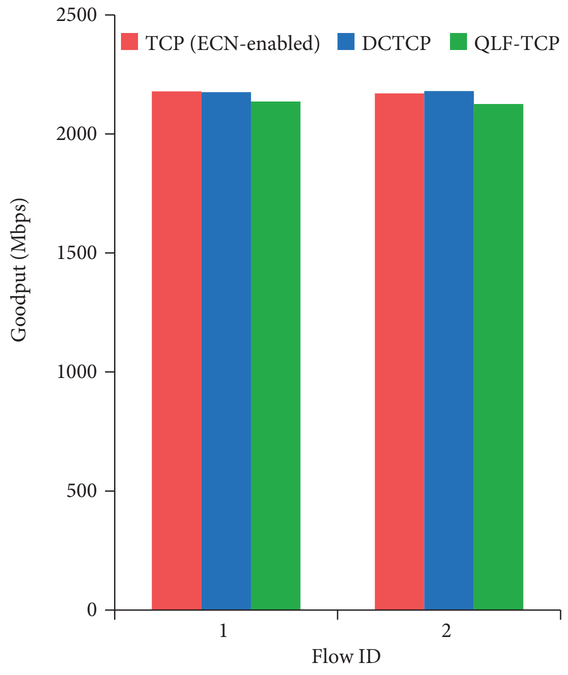

Figure 9. Two long flows, each 1 MB.

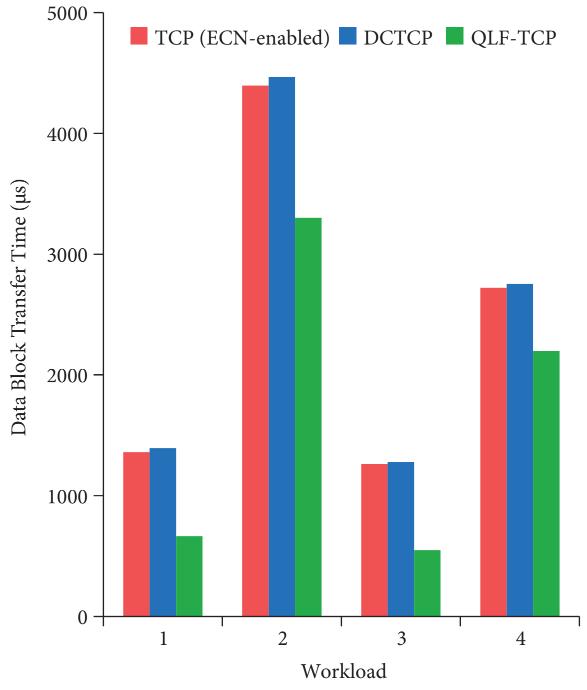

Figure 10. Data block transfer time for each workload.

Table 2. Percentage of improvement in data block transfer time.

	Workload			
	1	2	3	4
QLF-TCP improvementover ECN-enabled TCP	51.05%	24.88%	56.54%	19.19%
QLF-TCP improvement over DCTCP	52.28%	26.07%	57.07%	20.13%

4.3. Conclusion

In this paper, we presented a QLF-based solution to the TCP Incast throughput collapse problem. We presented a new queue length-based congestion parameter, which accurately measures the congestion along the path from source to destination, and a congestion control scheme that uses the new congestion parameter to prevent throughput collapse due to many-to-one communication by minimizing packet drops. The scheme also ensures high resource utilization and burst tolerance. The results show that our scheme significantly improves DBTT compared to the most widely used transport protocols in DCNs. As future work, we aim to modify the scheme to cater for flow priorities.

References

[1] Al-Fares M, Loukissas A, Vahdat A. A scalable, commodity data center network architecture. ACM Sigcomm Comp Com 2008; 38: 63-74.

[2] Greenberg A, Hamilton J, Maltz DA, Patel P. The cost of a cloud: research problems in data center networks. ACM Sigcomm Comp Com 2008; 39: 68-73.

[3] Guo C, Wu H, Tan K, Shi L, Zhang Y. Dcell: a scalable and fault-tolerant network structure for data centers. ACM Sigcomm Comp Com 2008; 38: 75-86.

[4] Abts D, Felderman B. A guided tour of data-center networking. Commun ACM 2012; 55: 44-51.

[5] Zhang Y, Ansari N. On architecture design congestion notification TCP Incast and power consumption in data centers. IEEE Commun Surv Tut 2013; 15: 39-64.

[6] Zhang J, Ren F, Lin C. Survey on transport control in data center networks. IEEE Network 2013; 27: 22-26.

[7] Ousterhout J, Agrawal P, Erickson D, Kozyrakis C, Leverich J, Mazières D, Mitra S, Narayanan A, Ongaro D, Parulkar G et al. The case for RAMCloud. Commun ACM 2011; 54: 121-130.

[8] Bilal K, Khan SU, Zhang L, Li H, Hayat K, Madani SA, Min-Allah N, Wang L, Chen D, Iqbal M et al. Quantitative comparisons of the state-of-the-art data center architectures. Concurr Comp-Pract E 2013; 25: 1771-1783.

[9] Zhang J, Ren F, Lin C. Modeling and understanding TCP Incast in data center networks. In: Proceedings of IEEE INFOCOM; 10–15 April 2011; Shanghai, China. New York, NY, USA: IEEE. pp. 1377-1385.

[10] Ren Y, Zhao Y, Liu P, Dou K, Li J. A survey on TCP Incast in data center networks. Int J Commun Syst 2014; 27: 1160-1172.

[11] Chen Y, Griffith R, Liu J, Katz RH, Joseph AD. Understanding TCP Incast throughput collapse in datacenter networks. In: Proceedings of the 1st ACM Workshop on Research on Enterprise Networking; 16–21 August 2009; Barcelona, Spain. New York, NY, USA: ACM. pp. 73-82.

[12] Alizadeh M, Greenberg A, Maltz DA, Padhye J, Patel P, Prabhakar B, Sengupta S. Data center TCP (DCTCP). ACM Sigcomm Comp Com 2010; 40: 63-74.

[13] Sreekumari P, Jung JI, Lee M. A simple and efficient approach for reducing TCP timeouts due to lack of duplicate acknowledgments in data center networks. Cluster Comput 2016; 19: 633-645.

[14] Xu L, Xu K, Jiang Y, Ren F, Wang H. Throughput optimization of TCP Incast congestion control in large-scale datacenter networks. Comput Netw 2017; 124: 46-60.

[15] Huang J, He T, Huang Y, Wang J. ARS: Cross-layer adaptive request scheduling to mitigate TCP Incast in data center networks. In: 35th Annual IEEE International Conference on Computer Communications; 10–14 April 2016; San Francisco, CA, USA. New York, NY, USA: IEEE. pp. 1-9.

[16] Li S, Li D, Du Z. Adaptive rate control for TCP Incast based on selective ECN-marking. In: 7th IEEE International Conference on Software Engineering and Service Science; 26–28 August 2016; Beijing, China. New York, NY, USA: IEEE. pp. 353-356.

[17] Tseng HW, Chang WC, Peng I, Chen PS. A cross-layer flow schedule with dynamical grouping for avoiding TCP Incast problem in data center networks. In: Proceedings of the International Conference on Research in Adaptive and Convergent Systems; 2016; Odense, Denmark. New York, NY, USA: ACM. pp. 91-96.

[18] Zhang J, Ren F, Yue X, Shu R, Lin C. Sharing bandwidth by allocating switch buffer in data center networks. IEEE J Sel Area Comm 2014; 32: 39-51.

[19] Zou S, Huang J, Zhou Y, Wang J, He T. Flow-aware adaptive pacing to mitigate TCP Incast in data center networks. In: 37th IEEE International Conference on Distributed Computing Systems; 5–8 June 2017; Atlanta, GA, USA. New York, NY, USA: IEEE. pp. 2119-2124.

[20] Adesanmi A, Mhamdi L. M21TCP: Overcoming TCP Incast congestion in data centres. In: IEEE 4th International Conference on Cloud Networking; 5–7 October 2015; Niagara Falls, Canada. New York, NY, USA: IEEE. pp. 20-25.

[21] Abdelmoniem AM, Bensaou B, Abu AJ. Mitigating Incast-TCP congestion in data centers with SDN. Ann Telecommun 2018; 73: 263-277.

[22] Hafeez T, Ahmed N, Ahmed B, Malik AW. Detection and mitigation of congestion in SDN enabled data center networks: a survey. IEEE Access 2018; 6: 1730-1740.

[23] Huang J, Huang Y, Wang J, He T. Adjusting packet size to mitigate TCP Incast in data center networks with COTS switches. IEEE T Cloud Comput 2018; 1: 1-1.

[24] Shukla S, Chan S, Tam ASW, Gupta A, Xu Y, Chao HJ. TCP PLATO: Packet labelling to alleviate time-out. IEEE J Sel Area Comm 2014; 32: 65-76.

[25] Floyd S. TCP and explicit congestion notification. ACM Sigcomm Comp Com 1994; 24: 8-23.