Enhancing of Micro Flow Transfer in SDN-based Data Center Networks

Maiass Zaher
Department of Telecommunication and Media Informatics
Budapest University of Technology and Economics
Budapest, Hungary
zaher@tmit.bme.hu

Sándor Molnár
Department of Telecommunication and Media Informatics
Budapest University of Technology and Economics
Budapest, Hungary
molnar@tmit.bme.hu

***Abstract*—As micro traffic is dominant in data center networks, many conditions must be satisfied to guarantee a very short delay. In this context, the flexibility provided by software defined networks (SDN) represents an opportunity for proposing new quality of service (QoS) based solutions. In this paper, we propose a new SDN-based framework to utilize QoS mechanisms in providing better conditions for data center networks to deliver micro flows efficiently. We present a Micro-flow framework which deals with queue length as a sign to avoid congestion events so that the flow completion time of micro flows can be improved by adapting the data rate of background flows without any intervention of source end hosts for rapid reaction. By creating queues match the network topology, our framework probes the length of queues dedicated for micro flows such that the data rate of background flows on upstream switches can be updated proportionally to the length of the micro queue on the downstream switch. We evaluated our solution by Mininet in different scenarios. The experiments demonstrate that Cubic and New Reno with our framework give better completion time for micro flows compared to that when DropTail is used by Cubic and New Reno, mitigate the probability of congestion events and do not severely throttle the data rate of background flows.**

***Index Terms*—Micro flow, Software Defined Networks, Quality of Service, SDN controller, Data Center Networks, OpenFlow, Network congestion**

I. Introduction

Since networks have limited resources, specifically buffer size, network congestion might decrease the performance of the network since it has a strong influence on the quality of service. In this context, there are many solutions for the congestion which can be classified into two categories: i) in-network solutions, ii) end-host solutions [1]. The first category encompasses solutions rely on scheduling [2], rerouting [3] or change TCP parameters [4] [5] which mainly intend to mitigate or avoid the congestion, where network nodes like switches and routers hold mechanisms to deal with the congestion. In contrast, the solutions belong to the second category define a direct reaction at end-hosts based on the network situation [6] [7]. However, all end-hosts solutions are performed by TCP protocol which provides congestion control algorithms for adapting the data rate according to network conditions by tuning the size of the congestion window based on the network feedback. Many-to-one data transmission pattern is a reason of the network congestion problem, especially in data center networks [8] and cluster-based storage systems [9] where it is a result of a simultaneous data sending from many senders to one receiver. As a result, TCP re-transmissions after timeouts can degrade the overall throughput [10].

End-to-end delay is the summation of processing, transmission, propagation, and queueing delays where processing delay can be neglected, and propagation delay can be computed based on network characteristics, so the summation of transmission and queueing delays over a route makes up the upper bound total end-to-end delay [11]. However, queueing delay depends on the scheduling mechanism at egress ports along the route. As a result, each link in the network can provide a unique level of QoS based on the output queue situation. In this context, the introduction of SDN enables us to implement the principles of quality of service such that we can avoid and mitigate the congestion by allocating network resources centrally [12]. In this regard, SDN provides valuable capabilities to deal with congestion events such as central monitoring and controlling so that proposing new solutions will be more flexible. Therefore, the categories of congestion solutions can be implemented by SDN since southbound APIs like OpenFlow can be employed by SDN controllers to inspect the network situation where SDN provides a fine-granular way to handle flows based on many packet fields and parameters collected from the data plane.

The main contribution of this paper is proposing a framework to improve the flow completion time of micro flows in data center networks by leveraging of SDN paradigm without dramatically degrading the throughput of background flows. Furthermore, our framework can provide better service for micro flows by eliminating the delay resulted from contacting the end-hosts upon congestion events. Besides, our framework does not require any modification in TCP stack nor switch hardware.

This paper is structured as follows. Section 2 presents related works. We discuss our framework in Section 3 and evaluate its performance under congestion scenarios in Section 4. We conclude our work in Section 5.

II. Related Works

In this section, we present the literature about SDN based solutions for the congestion problem. The researches related to this problem can be classified into three main categories. The

first category is QoS based rerouting solutions. The second category contains solutions which modify TCP fields in IP packets. The last category contains QoS queue based solutions. Regarding the last category, the solution presented in [13] can avoid the congestion in private campus networks by creating many queues on switch ports and finding paths based on SDN paradigm such that satisfying the bandwidth requirements without exceeding the capacity of the links. Authors in [14] define QoS requirements for all applications in data center network to create prioritized queues on switch ports such that packets are dequeued out of ports based on their priority without exceeding the bandwidth of the ports to avoid the congestion. A dynamic routing and rate limit system has been proposed in [15] to find non-congested routes and dynamically reroute flows upon detection the congestion on any switch port based on SDN paradigm as well as to define prioritized rate limits. Other SDN based solutions fulfill QoS requirements by setting data rates [16] [17]. However, no distinction between micro and background flows has been proposed in the previous solutions.

Our framework stems from the fact that end-to-end solutions that rely on window management can be far from effective, requiring many round trip times (RTTs) to react properly to congestion by end hosts. Also, the solutions rely on rerouting do not provide the satisfied guarantee for micro flows. Finally, some solutions in the last category require modifications to TCP stack and need time for a proper response more than the lifespan of micro flows in data center networks.

III. Micro-flow Framework

In this section, we introduce the architecture and functionality of our framework. The terminologies and variables used by Micro-flow framework are presented and described in Table I and Table II respectively.

A. Framework Architecture

Micro-flow framework aims to guarantee the required bandwidth for micro flows by reducing the data rate of background flows under high load epochs so that micro flows will be served with less delay. To control the data rate of background flows, queues have been created on each switch port, dedicated for background flows, as many as egress ports a downstream switch has as well as one more queue has been created on each port dedicated for micro flows. Background flows in an upstream switch will be mapped to one of background queues based on their destination and all micro flows will be mapped to the micro queue. Fig. 1 shows flows forwarded out of an egress port on the upstream switch to a port on the downstream switch. As a result, the data rate of the flows will have a direct impact on the queue length of egress ports on the downstream switch.

Micro-flow framework checks the length of micro queues so that the data rate of background flows forwarded through the corresponded mapped queues on upstream switches will be adapted accordingly. The architecture of Micro-flow framework is shown in Fig. 2. We can see that Micro-flow

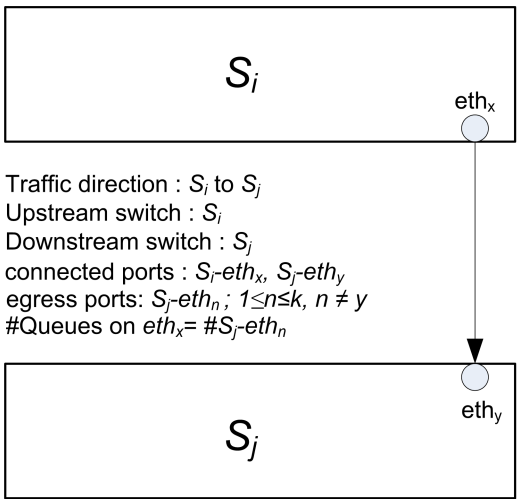


Fig. 1 Connected switches

framework employs an SDN controller, where the control plane of SDN paradigm comprises the queue monitoring module, background flow selection module, and data rate control module. By using the SDN paradigm, the previous modules can control and monitor the data plane. Micro-Flow framework considers the main idea which is that the delay of micro flows in data center networks is yielded by the existence of background flows. Therefore, our framework proposes a new approach to deal with this problem by applying QoS based solution on upstream switches instead of notifying data sources due to the fact that such kind of notifications takes time longer than the lifespan of micro flows. For the sake of immediate reaction to any probable degradation in flow completion time (FCT) of micro flows, our framework updates the data rate of the corresponding background queues on upstream switches commensurate with the length of micro queues on downstream switches.

B. Queue monitoring

Centrally in the control plane of SDN paradigm, Queue monitor module periodically probes the length of queues over

TABLE I
TERMINOLOGIES

Terminology	Description
Upstream Switch	a switch from which a traffic flow is forwarded to its downstream switch
Downstream Switch	a switch receives a traffic flow from its upstream switch
Egress Port	any switch port out of which the arrived traffic on another port will be forwarded
micro queue	a queue dedicated for micro flows
background queue	a queue dedicated for background flows forwarded to a specific egress port on a downstream switch

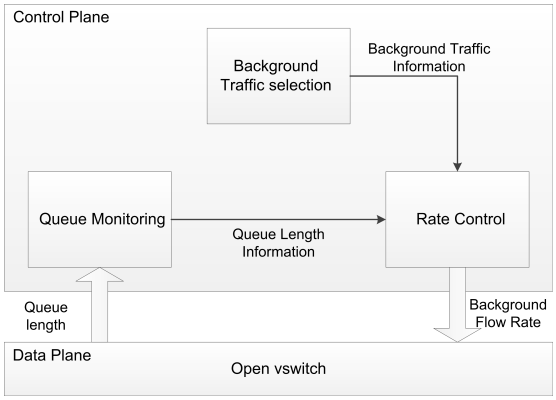


Fig. 2 Architecture of Micro-flow framework

a varied interval. Queue monitor module is based on the observation that more than 80% of flows in data center networks are micro flows, and their size less than 10KB [18]. According to this fact, we have defined the congestion threshold as a function of the number of egress ports. To mitigate the overhead, queue monitor module probes the length of the micro queue on switch ports over intervals proportionally vary to the length of micro queue such that the probing interval will be longer as the length of micro queue gets shorter. Algorithm 1 shows the steps of the queue monitor module.

Algorithm 1 Framework Applications
initial: *ql* = 0, *p* = *k*, *pkt* = 1500B, *qc* = BDP, *interval* = 0, *bgt_size* = 10KB, timeout = 100ms

Function *Queue_Monitor*( )
repeat per *interval*:
TH = ($qc_{s_j-eth_n}$ - (*p* - 1) * *pkt*)
  if ($ql_{s_j-eth_n}(t) \geqslant TH$)
    Datarate_Update (s_i- eth_x, $ql_{s_j-eth_n}$)
  endif
  $interval = \frac{TH}{ql_{s_j-eth_n}} * RTT$
Function *Find_BGT*( )
  if (*flow_size* > *bgt_size*):
    bgt_flows = requests.get(*s-eth*)
    for *m* in *bgt_flows*
      bgt_list[*m*] = (*ip_src, ip_dst, port_src, port_dst*)
      map *bgt_list[m]* to a queue
  for *m* in *bgt_flows*
    if *bgt_list*[*m*] **NOT IN** *bgt_flows*:
      del (*bgt_list*[*m*])
Function *Datarate_Update*(s_i-eth_x, $ql_{s_j-eth_n}$)
  $\alpha = \frac{ql_{s_j-eth_n}(t) - TH}{qc_{s_j-eth_n} - TH}$
  if ($ql_{s_i-eth_x} < TH$):
    $\beta = \frac{qc_{s_i-eth_x} - ql_{s_i-eth_x}}{qc_{s_i-eth_x}}$
  else:
    exit
  $TR = BW_{s_j-eth_n} \times (1 - \frac{\alpha \times \beta}{2})$
  change *TR* on s_i-eth_x for *timeout*

C. Background flow selection

Our background flow selection module leverages sFlow [19] to identify background flows. In addition, this module maintains an active background flows table that includes all active background flows forwarded out a specific port. Since our framework needs to maintain the required information to define background flows uniquely, each table entry comprises of *src_ip, dst_ip, src_port, dst_port, port*. Background flow selection module adds a new entry to the table when the flow volume is larger than 10 KB. When the length of the micro queue is longer than the threshold, this module will retrieve all entries for background flows forwarded out a specific port. Whenever the length of the micro queue is less than the threshold, all entries associated with that port will be deleted. Furthermore, to maintain active background flows only in the table, all entries of idle flows will be deleted whenever their connections end. Algorithm 1 presents background flow entry insertion and deletion operations.

D. Data rate control

After adding entries into the active background flow table, data rate control module employs QoS capabilities of the SDN controller to regulates the data rate of background queues on the upstream switches in accordance with the length of background queues and that of the micro queue on an egress port in a downstream switch. More specifically, a switch port is overloaded if the received traffic rate is more than its link bandwidth. As a result, its queue will grow proportionally to the difference between the received traffic rate and link bandwidth. Therefore, Micro-flow framework mitigates the received traffic rate on a port by reducing the data rate of background queues on upstream switches to guarantee fast transmission for micro flows. Therefore, background flows will be forwarded out of an egress port on an upstream switch by a rate directly proportional to the length of its background queue and inversely proportional to the length of the micro queue on the egress port of the downstream switch. The data rate of background queue on upstream switch port is computed as presented in Algorithm 1 where α and β are control parameters proportional to the length of the micro queue on the downstream switch and that of the background queue on the upstream switch port, respectively. Consequently, the data rate of background queues on ports of upstream switches will be mitigated up to 50% when the length of the micro queue on egress port of the downstream switch is about to have congestion event as well as no backlog in the background queue on upstream switches. In contrast, it cannot be mitigated when the length of a background queue is larger than or equal to the threshold. Fig. 3 illustrates the work-flow of the framework. Micro-flow framework was implemented in python to run along with any SDN controller can provide Rest API. We integrate our framework with Ryu controller [20].

IV. Experimental results

We evaluated the performance of Micro-flow framework via Mininet 2.3.0. In this section, we present the results of

TABLE II
VARIABLES USED BY MICRO-FLOW FRAMEWORK

Variable	Description
ql	Queue occupation
p	Number of switch ports
k	The scale of Fat-tree topology
qc	Queue capacity
BDP	Bandwidth delay product
interval	Probing queue length frequency
bgt_size	The threshold to identify background traffic
timeout	The maximum time of applying updated data rate
$s_a - eth_b$	The Ethernet port number b on switch a
s-eth	Any Ethernet port on any switch
pkt	The packet size
TH	The threshold of micro queue occupation
flow_size	The size of a flow
bgt_flows	The set of all identified background flows
bgt_list	The active background flows table
α	the control parameter of micro queue length on downstream switch
β	the control parameter of background mapped queue length on upstream switch
TR	The data rate of background mapped queue on upstream switch
BW	Bandwidth of egress port on a downstream switch

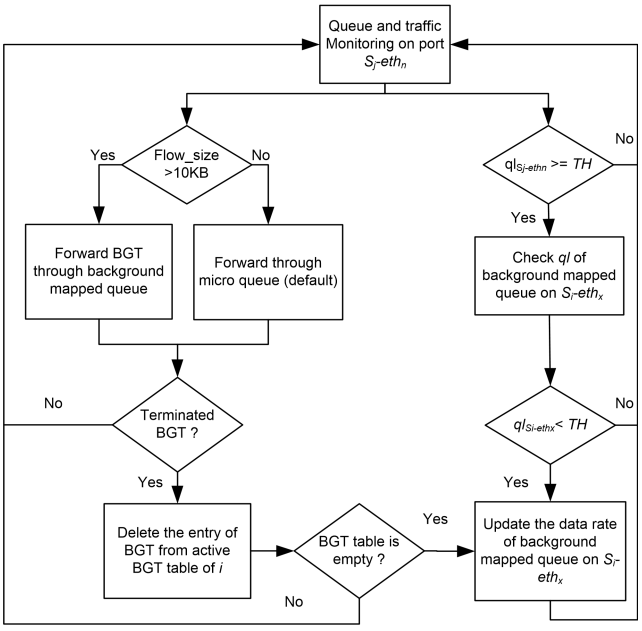
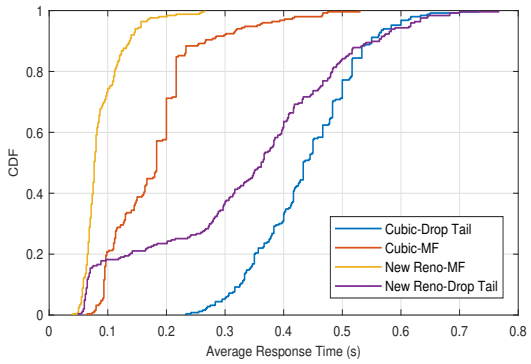


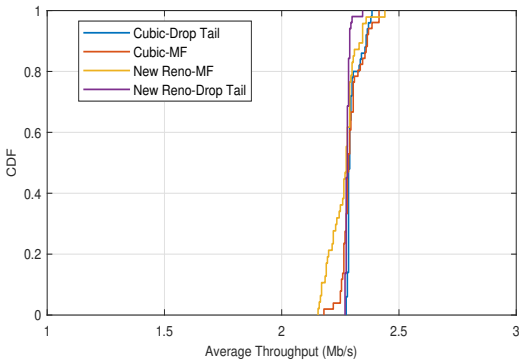
Fig. 3 Micro-flow Framework workflow

the experimental evaluation of Micro-framework where we compared the performance of TCP new Reno and TCP Cubic with drop tail Active Queue Management (AQM) enabled to that with Micro-flow framework. However, TCP Cubic and TCP new Reno use packet loss to detect network congestion, but TCP Cubic is less aggressive than new Reno and it is used by default in Linux kernels. For proper configuration, we enabled Selective Acknowledgment (SACK) for all scenarios, we set minimum Retransmission Timeout (RTO) to the default value of Linux which is 200ms, IP packet size to 1500 Bytes, and the buffer size to 36 packets (54 KB). We used fat-tree topology whose scale k=4 similar to the one proposed in [21]. Each core switch connects to four aggregation switches with 100 Mbps bandwidth and 250 μs one-way propagation delay links, 10 Mbps and 2 ms one-way propagation delay for links connect aggregation and edge layers, and 5 Mbps and 5 ms one-way propagation delay for links connect edge switches and end-hosts where each edge switch has two servers. Therefore, the over-subscription ratio is 1:1 at edge level. We evaluate the performance of the framework by run 10 seconds simulations for three different scenarios contain a mix of 128 micro and background flows. In all scenarios, background and micro flows follow two patterns, the first for generating connections span all topology layers, and the second for connections span the edge and aggregation layers where all servers in all racks have been employed to generate the traffic volume for each scenario. We employed iperf for generating unlimited background traffic during the whole simulation period and Apache server [22] for generating micro flows by repeatedly requesting "index.html" webpage of size 10 KB as reported in [18] at the beginning of each two seconds during the simulation period. Therefore, Micro-flow framework will be examined in a situation where micro flows are synchronized to generate burstiness, and background flows exist to evaluate the framework under high load. We repeated the experiment 50 rounds for three different scenarios and the average throughput of background flows has been computed for each round while the average flow completion time of micro flows has been computed for each two seconds epoch. The details of three scenarios are as follows, in the first scenario, we simulated a background-to-micro ratio of 1:3,i.e., 32 background and 96 micro flows as the ratio reported in [18] [10]. In the second scenario, we employed higher background flows share of 3:1 to investigate the impact of Micro-flow framework under a high volume of background traffic. In the third scenario, we run a ratio of 1:1. For micro flows, we present Cumulative Distribution Function (CDF) of average FCT. These results are shown in Fig. 4 for the 1:3 scenario, Fig. 5 for the 3:1 scenario, and Fig. 6 for the 1:1 scenario. For background flows, we essentially present CDF of the average throughput.

In 1:3 scenario, Fig. 4a shows that Micro-flow framework can improve FCT of micro flows under high load of micro flows compared to both Cubic Drop Tail and New Reno Drop Tail. This mainly due to the fact that a high load of micro flows

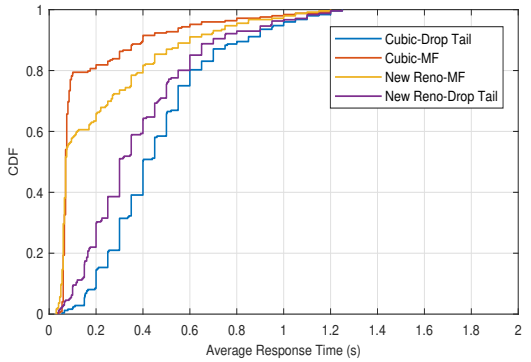


(a) Average FCT for micro flows

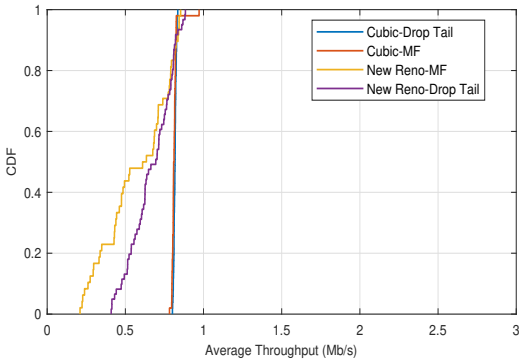


(b) Average throughput for background flows

Fig. 4: Performance of Micro-flow framework for TCP Cubic and TCP New Reno in 1:3 ratio scenario

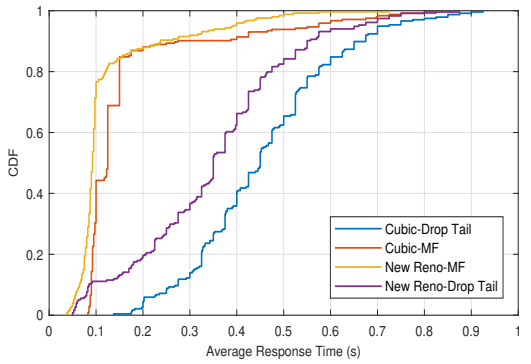


(a) Average FCT for micro flows

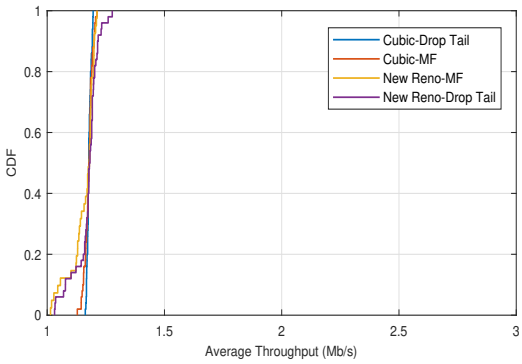


(b) Average throughput for background flows

Fig. 5: Performance of Micro-flow framework for TCP Cubic and TCP New Reno in 3:1 ratio scenario



(a) Average FCT for micro flows



(b) Average throughput for background flows

Fig. 6: Performance of Micro-flow framework for TCP Cubic and TCP New Reno in 1:1 ratio scenario

bloats the buffers which worsens their completion time. Fig. 4b shows that Micro-flow framework nearly does not aggressively impact on the throughput of background flows under the high ratio of micro flows due to the fact that when threshold hits, there is not a high load of background flows; therefore, the data rate of background flows will be proportionally reduced. In 3:1 scenario, Fig. 5a shows similar results to the previous scenario under high ratio of background flows compared to both Cubic Drop Tail and New Reno Drop Tail because of buffers bloating which worsens the completion time of micro flows. Also, Fig. 5b shows that Micro-flow framework does not degrade the throughput of background flows due to that there are not so many threshold hits during this scenario. Finally, similarly Fig. 6a shows that Micro-flow framework improved FCT of micro flows under the balanced ratio. Furthermore, Fig. 6b shows the same results as before. As a result, Micro-flow framework maintains a low impact on the throughput of background flows because it temporarily mitigates the data rate of background flows which will be restored after a very short period equals at most to the transmission time of micro flow.

V. Conclusion

In this paper, we presented an SDN-based QoS congestion control framework to improve the flow completion time of micro flows in data center networks. Our framework relies on monitoring of queue length by employing the capabilities of SDN paradigm to reduce the probability of service degradation of micro flows that might take place due to the coexistence of high volume of micro and background flows. Our framework creates queues on each switch port employing one for micro flows and the others for background flows based on their destinations. Micro-flow framework mitigates the data rate of background flows when the micro queue occupation on a downstream switch hits the predefined threshold. Our framework showed mitigation in flow completion time of micro flows up to 400 ms without impairing the throughput of background flows by comparing its performance to the performance of Cubic and New Reno with DropTail AQM. It is worth to mention that Micro-flow framework can provide instant reaction to probable congestion events without contacting the data sources to mitigate the delay of reaction especially in an environment like data center network where the majority of the traffic is micro flows which need more rapid reactions than that in the literature. However, further performance evaluation of Micro-flows framework in real production data center networks with realistic conditions should be conducted.

References

[1] T. Hafeez, N. Ahmed, B. Ahmed, and A. Malik, "Detection and mitigation of congestion in SDN enabled data center networks: A survey," *IEEE Access*, vol. 6, pp. 1730-1740, 2018.

[2] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. NSDI*, Vol. 10, no. 8, pp. 89-92, Apr. 2010.

[3] Y. Hu, T. Peng, and L. Zhang, "Software defined congestion control algorithm for IP networks," *SCI. PROGRAMMING-NETH*, 2017.

[4] Y. Lu, X. Fan, and L. Qian, "EQF: An Explicit Queue-Length Feedback for TCP Congestion Control in Datacenter Networks," in *Proc of Fifth International Conference on Advanced Cloud and Big Data (CBD)*, Aug. 2017, pp. 69-74.

[5] X. Jiang, and G. Jin, "CLTCP: An Adaptive TCP Congestion Control Algorithm Based on Congestion Level," *IEEE Commun. Lett.*, vol. 19, no. 8, PP. 1307-1310, Aug. 2015.

[6] S. Jouet, C. Perkins, and D. Pezaros, "OTCP: SDN managed congestion control for data center networks," in *Network Operations and Management Symposium (NOMS)*, Apr. 2016, pp. 171-179.

[7] M. Wang, J. Wang, and S. Han, "Adaptive congestion control framework and a simple implementation on high bandwidth-delay product networks," *Comput. Networks*, vol. 64, PP. 308-321, May 2014.

[8] G. Jereczek, G. L. Miotto, and D. Malone, "Analogues between tuning TCP for data acquisition and datacenter networks," in *Proceeding of ICCC*, Jun. 2015, pp. 6062–6067.

[9] A. Phanishayee, et al. "Measurement and analysis of TCP throughput collapse in cluster-based storage systems," in *Proceeding of FAST*, vol. 8, May 2008, pp. 1–14.

[10] Alizadeh, M., et al. "Data center TCP (DCTCP)," *ACM SIGCOMM computer communication review* vol. 38, no. 2, Mar. 2010, pp. 69-74.

[11] J. W. Guck, A. Van Bemten, and W. Kellerer, "DetServ: Network Models for Real-Time QoS Provisioning in SDN-Based Industrial Environments," *IEEE Trans. on Netw. Serv. Manage.*, vol. 14, no. 4, pp. 1003-1017, Dec. 2017.

[12] S. N. Hertiana, and A. Kurniawan, "A joint approach to multipath routing and rate adaptation for congestion control in openflow software defined network," in *Proceeding of 1st International Conference on Wireless and Telematics (ICWT), IEEE*, Nov. 2015, pp. 1-6.

[13] J. Xiao, S. Chen, and M. Sui, "The strategy of path determination and traffic scheduling in private campus networks based on SDN," *Peer Peer Netw. Appl.*, pp. 1-10, Dec 2017.

[14] F. Li, J. Cao, X. Wang, and Y. Sun, "A QoS guaranteed technique for cloud applications based on software defined networking," *IEEE Access,* vol. 5, pp. 21229-21241, 2017.

[15] N. F. Huang, I. J. Liao, H. W. Liu, S. J. Wu, and Chou, C. S. (2015). "A dynamic QoS management system with flow classification platform for software-defined networks," in *proceeding of 8th International Conference onUbi-Media Computing (UMEDIA)*, Aug. 2015, pp. 72-77.

[16] S. Sharma, et al. "Implementing quality of service for the software defined networking enabled future internet," in *Proceeding of Third European Workshop on Software Defined Networks (EWSDN)*, Sep. 2014, pp. 49-54.

[17] S. Tomovic, N. Prasad, and I. Radusinovic, "SDN control framework for QoS provisioning," in *22nd Telecommunications Forum Telfor (TELFOR)*, Nov. 2014, pp. 111-114.

[18] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010, pp. 267-280.

[19] sflow.org: sflow. https://sflow.org/. Accessed 2 August 2017.

[20] Ryu: Ryu SDN Framework. http://osrg.github.io/ryu/ Accessed 12 Sep. 2017.

[21] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Computer Communication Review*, Vol. 38, No. 4, pp. 63-74, 2008.

[22] Apache.org: Apache HTTP server benchmarking tool. http://httpd.apache.org/docs/current/install.html