Class-based Flow Scheduling Framework in SDN-based Data Center Networks

Maiass Zaher
Dept. of TMIT
BME
Budapest, Hungary
zaher@tmit.bme.hu

Aymen Hasan Alawadi
Dept. of TMIT
BME
Budapest, Hungary
aymen@tmit.bme.hu

Sándor Molnár
Dept. of TMIT
BME
Budapest, Hungary
molnar@tmit.bme.hu

***Abstract*—The emerging technologies leveraging Data Center Networks (DCN) and their consequent traffic patterns impose more necessity for improving Quality of Service (QoS). In this paper, we propose Sieve, a new distributed SDN framework that efficiently schedules flows based on the available bandwidth to improve Flow Completion Time (FCT) of mice flows. In addition, we propose a lightweight sampling mechanism to sample a portion of flows. In particular, Sieve schedules the sampled flows, and it reschedules only elephant flows upon threshold hits. Furthermore, our framework allocates a portion of the flows to ECMP, so that the associated overhead can be mitigated in the control plane and ECMP-related packet collisions are fewer as well. Mininet has been used to evaluate the proposed solution, and Sieve provides better FCT up to 50% in comparison to the existing solutions like ECMP and Hedera.**

***Index Terms*—Mice flow, Elephant flow, SDN, Data center network, flow scheduling**

I. Introduction

Typically, applications in DCN generate two classes of flows which are mice and elephant flows. Mice flow is the smallest and shortest-lived flows, and it is more conservative to the communication delay. On the other hand, the largest and longest-lived flows (i.e., elephant flows) are more affected by the available bandwidth. The elephant flows are fewer than mice flows, but they carry most (e.g., 80%) of the transferred data [1] [2] in DCN. Therefore, these classes compete for network resources. In this context, SDN paradigm provides opportunities to improve network management and control. The separated control plane provides an effective resource handling in comparison with the traditional networks. Besides, OpenFlow protocol [3] is one of the primary protocols to provide communication between the SDN controller in the control plane and the switches in the data plane. In today's DCN, SDN plays a vital role in network resource allocation and traffic monitoring. Hence, the paradigm has been significantly applied by the research community for flow scheduling and traffic load balancing [4], [5]. The process of managing massive data transmissions in real-time requires an efficient resource and traffic management [6]. The typical design of DCN represents a multi-rooted tree that has many different paths between each pair of hosts. Therefore, the challenge is to identify the suitable path for flows and avoid the potential traffic congestions as well. In particular, such kind of congestions would profoundly degrade QoS of mice flows. Thus, deploying cloud-based applications in DCNs, which leverage the static hashing based scheduling such as ECMP, is not an efficient mechanism due to the probable packets collisions. Furthermore, employing network devices (e.g., hosts or switches) for flow scheduling is still challenging since it requires some modifications in the kernel or sometimes in the hardware. On the other hand, the fully central flow scheduling model yields overhead on the control plane.

In this paper, we present Sieve framework which provides a distributed SDN-based scheduling solution. The key contribution of this paper is proposing a flow scheduling framework that achieves the following objectives:

1) We designed and implemented a distributed SDN-based framework for scheduling flows in DCN.
2) Sieve reschedules only elephant flows based on the ports occupations.
3) Sieve implements a lightweight sampling mechanism to classify flows.
4) Our framework improves FCT of mice flows in DCN without dramatically degrading the throughput of elephant flows.
5) Our framework does not require any modification in end-host kernel nor switch hardware.

This paper is structured as follows: Section 2 surveys the related works. We present the Sieve framework in Section 3, and evaluate its performance in Section 4. We conclude our work in Section 5.

II. Related work

In this section, we present the literature about SDN-based solutions for flow scheduling. Most of the current flow scheduling solutions are either fully central like Hedera [5] or distributed like CONGA [7]. Furthermore, some works are intended to involve network devices (e.g., hosts or switches) in flow scheduling decision. However, these works are still challenging to be deployed in today's DCN. For example, Mahout [4] requires modifications in the kernel of servers to detect elephant flows, whereas some other works employ end-host like in [8] to calculate the transmission delay. The works

T978-1-7281-6330-7/20/$31.00 ©2020 IEEE

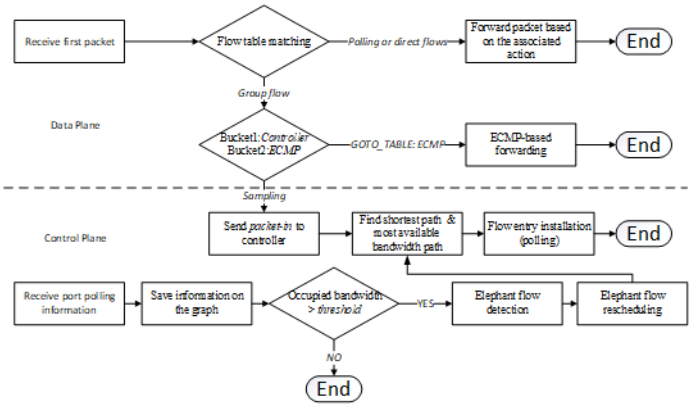

Fig. 1. Flow chart of the proposed framework

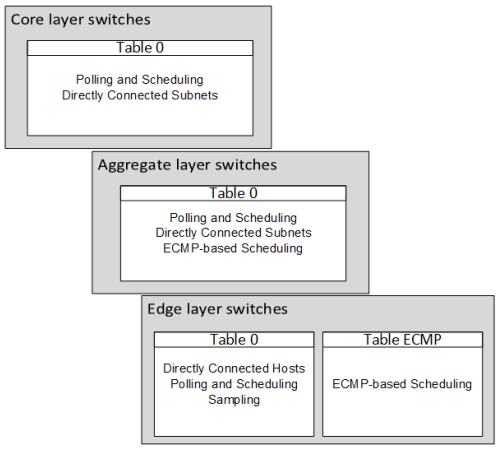

Fig. 2. Flow entry tables in different layers switches

in [7] and [9] add extra functions to the switches for sampling and delay calculations.

Hedera [5] reschedules a flow when its consumption exceeds 10% of the link capacity. Besides, Hedera schedules mice flows by hashing-based ECMP only. Mahout [4] employs end-hosts in DCN to detect the elephant flows. However, deploying such a method yields overhead due to the contacting with the end-hosts. Yazidi et al. [10] propose hot and cold link detection mechanisms to reschedule elephant flows. This solution requires to monitor demands of all flows. Tang et al. [11] propose a flow classification and a scheduling model for mice and elephant flows. The solution identifies the flows by tracking their inter-arrival times. The work in [12] presents a proactive method for flow scheduling by estimating an alternative path based on the port utilization, but monitoring every flow results in overhead in the control plane. The work in [13] proposes a flow scheduling algorithm in SDN-based DCN where it schedules aggregated elephant flows whose consumption is more than 10% of the link capacity. However, this study provides no investigation in terms of FCT of mice flows. Fu et al. [14] proposes a balancing flow table scheme to mitigate FCT of mice flows by computing the best path for each new mice flow, which incurs overhead since the number of mice flows is big in DCN. Therefore, Sieve is a new distributed solution, but it does not involve end-hosts contacting. In addition, Sieve relieves the overhead by applying its functions on a portion of the flows. Furthermore, no modifications in kernel nor hardware are required to apply it.

III. The Proposed Framework

In this section, we introduce the architecture and functionality of our framework.

A. Framework architecture

The framework architecture is depicted in Fig. 1. Our framework functionalities are distributed in the data and control planes that different flow tables are used in the data plane for specific purposes. Besides, the control plane contains modules for scheduling the sampled packets, polling network statistics, detecting and rescheduling elephant flows.

Due to the fact that ECMP yields flow collisions as a result of static hashing, we aim to sample a portion of the network flows so that our framework will schedule the sampled flows. ECMP schedules the remaining part of the network flows. In addition, Sieve improves FCT of mice flows by detecting and rescheduling only elephant flows upon threshold hits; hence, it mitigates the ECMP-related flow collisions.

Fig. 2 depicts the flow tables and the types of flow entries in switches of the three layers. Switches in the edge layer contain two tables. Table 0 contains proactive flow entries, which are for forwarding the packets destined to the directly connected hosts and for sampling packets, as presented in Table I. In addition, Table 0 contains flow entries for polling and scheduling sampled flows, which are re-actively installed by the framework in Table 0 of all switches along the chosen path. On the other hand, Table ECMP is the second flow table, in the edge switches, pipelined from Table 0 in case of selecting the second bucket of *Sampling* group entry of Table 0. As well as, switches in core and aggregate layers contain proactive flow entries for the directly connected subnets. Besides, ECMP-related scheduling is applied to the flows forwarded to an upper layer as shown in Table ECMP and Table 0 of edge and aggregate layers, respectively. Table II presents the structure of ECMP-related scheduling, which is a proactive group entry contains equally weighted buckets to apply ECMP logic. Furthermore, each flow entry type has a specific priority value matches the preference presented in Fig. 2.

Sieve employs an equally weighted bucket group of OpenFlow protocol to sample a portion of flows. In particular, the first bucket of the sampling group entry, presented in Table I, is for sending packet samples to the controller and the second one for applying ECMP. The first packet of a flow, arrived at an edge layer switch from a directly connected end-host, will match sampling group entry, so it is either forwarded based on

TABLE I
SAMPLING GROUP ENTRY

group_id	group_type	bucket_action
group_id=1	select	bucket=weight:50,actions=CONTROLLER, bucket=weight:50,actions=GOTO_TABLE:ECMP

TABLE II
GROUP ENTRY OF ECMP-BASED FORWARDING

group_id	group_type	matching_criteria	bucket_action
group_id=1	select	*dst_subnet_ip*	bucket=weight:50,actions=OUTPORT:1 bucket=weight:50,actions=OUTPORT:2

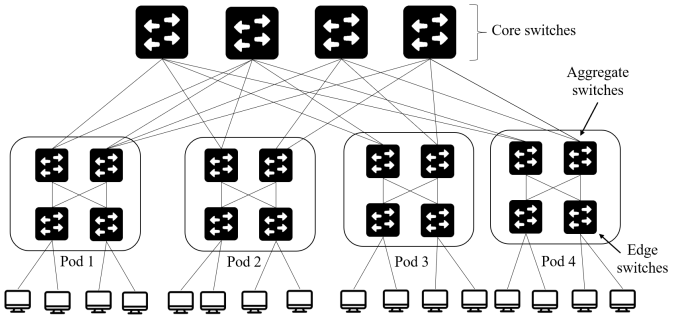

Fig. 3. *K-4* Fat-tree DCN topology

ECMP or sent to the controller by sending *packet-in* message. As a result, ECMP schedules a portion of flows, and Sieve will schedule the remaining part of the flows.

Then, Sieve starts to compute the shortest four paths between the source and the destination then adopts the path whose bottleneck link is the maximum. For that sake, Sieve inspects the network graph representing the network topology and the available bandwidth. Finally, Sieve installs a new polling flow entry for the best path into switches along it for the subsequent packets. In particular, the installed polling flow entries contain source IP, destination IP, source transport port, and destination transport port fields. Sieve assigns incremental priority for the installed polling flow entries provided the order presented in Fig. 2 is maintained. Specifically, our framework schedules the sampled flows similarly regardless they are elephant or mice flows.

Besides, Sieve periodically probes port statistics from all switches, and it maps the information on the network graph. Whenever, the occupied bandwidth on an edge switch port hits the threshold, which is 25% of the link capacity, Sieve starts to reschedule some or all elephant flows on the edge switch. Hence, it starts to detect the elephant flows forwarding out of this port. Sieve detects elephant flows based on the bytes count of the polling flow entries installed into the edge switch. Specifically, Sieve fetches all polling flow entries whose bytes count is more than 50KB. In this context, we consider the flow size as the classification principle in conformity with the study in [1]. Then, our framework finds the bottleneck of the possible new paths, and it examines if the alternative paths have greater available bandwidth than the original one. In addition, Sieve tries to find a new path that the original edge switch port is not a part of it. The framework reschedules a number of elephant flows proportional to the available bandwidth on both the original port and the bottleneck of the new path. However, in case there is only one elephant flow on an edge switch port upon threshold hits, Sieve will try to reschedule it to a better path. As a result, Sieve considers the potential ECMP-related collisions by tracking port statistics. Upon threshold hits, it provides more bandwidth for mice flows by rescheduling elephant flows.

B. Flows distribution

We implement *Select* type of the Openflow group feature with two equal weighted buckets. In particular, in case a packet matched the sampling group entry on an edge switch, the hashing value will be computed based on the packet header and the bucket weight value. Then, the corresponding action of the chosen bucket will be executed. To illustrate the hashing based packet allocation, we apply SUHA (Simple Uniform Hashing Assumption) on the proposed sampling mechanism. This claim can be achieved under two conditions:

1) The probability of placing any new item to the available buckets is equal.
2) The process of bucket-based hashing is independent.

These conditions can be implemented on our sampling technique. Assuming the incoming flows x and y will have different hashing values $f(x)$ and $f(y)$ depending on the packet header values and values of the bucket weights. However, the probability of allocating the flow to the first or the second bucket is equal. As a result, a flow can not be allocated to both buckets at the same time. Eq. 1 presents the probability of the hashing function.

$$Pr(h(x) = i) = \frac{1}{n} \tag{1}$$

$h(x)$ is the hashing function, i is the index value ($i \in [0, n-1]$), and n is the hashing locations (i.e., in our case n = 2). Therefore, the upper bound of the flows allocated to ECMP bucket can be calculated as in Eq. 2 using Boole's inequality.

$$Pr(ECMP \quad receives \geq k \quad flows) \leq \binom{n}{k} \frac{1}{n^k} \tag{2}$$

The binomial combination $\binom{n}{k}$ represents the subset of the hashed flows where every flow from k will be sent to the ECMP bucket with the probability of $\frac{1}{n^k}$, as proven in [18] for the Balls-and-Bins model. Therefore, the flows will be evenly distributed.

C. Framework implementation

We integrated the framework modules with Ryu SDN controller [16]. In addition, we leveraged OpenFlow 1.3.1 and Mininet 2.2.2d as the testbed environment to evaluate our framework in *4-ary* Fat-tree DCN topology, as shown in Fig. 3 [17]. We used Intel Core i5-8400 3.20 GHz, 16 GB RAM, Ubuntu 16.04.

IV. Experimental results

We compared the performance of our framework to that of Hedera and ECMP since Hedera is the mainstream scheduling and detection framework for DCN, and ECMP acts as a commonly used scheduler in academic and business sectors. For that sake, we employed *k-4* Fat-Tree topology in which core layer connects to aggregation layer with 100 Mbps bandwidth and 250 µs one-way propagation delay links, 20 Mbps and 1 ms one-way propagation delay for links connect aggregation and edge layers, and 10 Mbps and 2 ms one-way propagation delay for links connect edge layer and end-hosts.

We conducted two scenario groups. Firstly, we evaluated the performance of Sieve under three different 300 seconds scenarios. In the first scenario, called as *Concentrated Traffic (CT)*, elephant and mice flows follow many-to-one patterns, in which twelve servers send data to three servers on different pods than the sources. The second one is the uniform pattern, called as *Uniform Traffic (UT)*, where all servers are employed to generate the traffic, and each source has a different destination. Finally, the third scenario called as *Multi Destinations (MD)*, we generated traffic from two servers, connected to the same edge switch, to ten different servers on the other pods (i.e., five destinations for each source).

We employed *iperf* for generating elephant flows, and Apache server [18] for generating mice flows by repeatedly requesting a web page file of 10 KB in size at the tenth second of the simulation lifespan. As a result, for this group of scenarios, our framework will be examined in a situation where mice flows are synchronized to generate burstiness, and elephant flows exist to evaluate the framework under high load. We repeated the experiment 10 rounds for each different scenario. In addition, we employed high elephant flows share of 1:1 (i.e., mice to elephant ratio) to investigate the impact of the framework under a high volume of elephant traffic. Besides, we simulated a mice to elephant ratio of 3:1 as the ratio reported in [1]. Basically, we followed the evaluation model in [19] to evaluate the Sieve performance in terms of FCT of mice flows and goodput of elephant flows. Besides, we compared our framework performance under the traffic pattern employed in [5] in terms of loss rate and delay.

A. FCT of Mice flows and goodput of Elephant flows

We compared FCT of each algorithm since FCT is the most essential performance metric for mice flows. Basically, we present the average value of FCT and goodput as well as 99% percentile values of them to present the major value ranges. As shown in Fig. 4 and Fig. 6, Sieve outperforms Hedera and ECMP, specifically under CT traffic pattern. Our framework provides less delay for mice flows by rescheduling elephant flows. On the other hand, ECMP provides no distinction between elephant and mice flows, and it may yield collisions due to static hashing. Similarly, Hedera reschedules elephant flows based on their consumption only. Therefore, some congestions may occur without any reaction as long as elephant flows consumption under 10% of the link capacity. Similarly, FCT of mice flows under our framework is less than that under

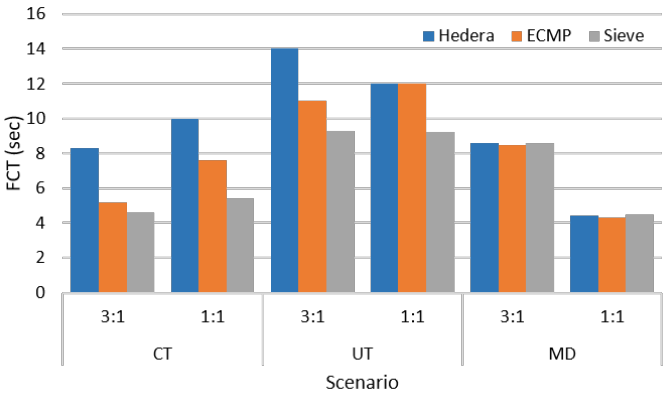

Fig. 4. 99% FCT of mice flows

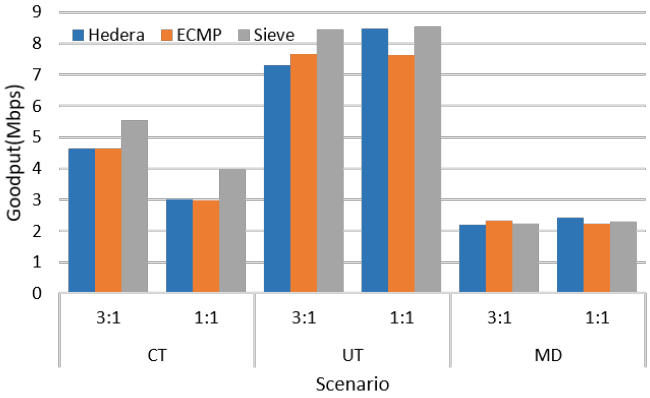

Fig. 5. 99% Goodput of elephant flows

Hedera and ECMP in case of UT scenario, as shown in Fig. 4 and Fig. 6. Since the number of the sources and destinations are same, Sieve finds more opportunity to reschedule elephant flows to better paths upon threshold hits. Finally, in the case of MD scenario, all algorithms provide the same performance in most cases, since there are only two sources connected to the same edge switch. Therefore, the possibility of rescheduling elephant flows to better paths is limited.

Furthermore, we compared the goodput of elephant flows in all scenarios, as shown in Fig. 5 and Fig. 7. Sieve provides goodput for elephant flows close to that of Hedera and ECMP under the most scenarios. In particular, under the ratio of 3:1, which is close to the real ratio in DCN. Table III & IV present the 95% confidence intervals of mice flows FCT and elephant flows goodput. From Table III, we observe that Sieve always had the best FCT among all algorithms in different scenarios. In addition, Sieve provides so close goodput values in comparison to other algorithms, as shown in Table IV.

B. Packet Loss and Network Delay

We conducted a second scenario group to investigate our framework performance in terms of loss rate and delay in the same topology shown in Fig. 3. We generated *ping echo*

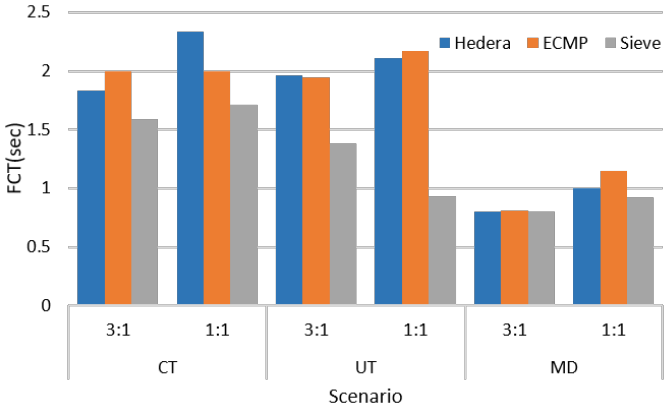

Fig. 6. Mean FCT of mice flows

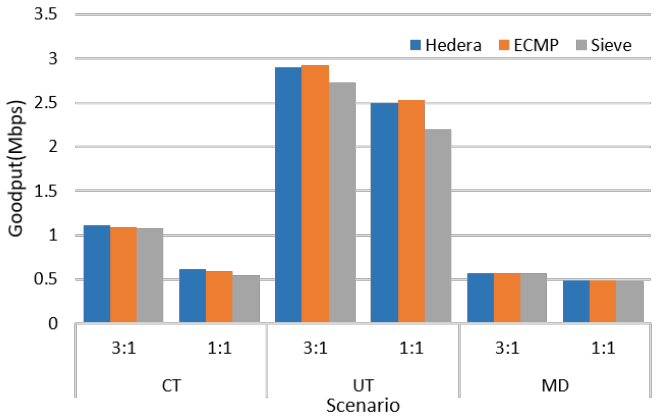

Fig. 7. Mean Goodput of elephant flows

TABLE III
GOODPUT WITH 95% CONFIDENCE INTERVAL

Scenario	Hedera	ECMP	Sieve
CT 3-1	1.11 ± 0.11	1.09 ± 0.12	1.08 ± 0.12
UT 3-1	2.9 ± 0.2	2.92 ± 0.21	2.73 ± 0.26
MD 3-1	0.57 ± 0.04	0.57 ± 0.04	0.57 ± 0.04
CT 1-1	0.61 ± 0.05	0.59 ± 0.04	0.54 ± 0.07
UT 1-1	2.5 ± 0.18	2.53 ± 0.17	2.2 ± 0.23
MD 1-1	0.49 ± 0.04	0.49 ± 0.04	0.48 ± 0.04

TABLE IV
FCT WITH 95% CONFIDENCE INTERVAL

Scenario	Hedera	ECMP	Sieve
CT 3-1	1.83 ± 0.03	2 ± 0.03	1.59 ± 0.05
UT 3-1	1.96 ± 0.24	1.94 ± 0.23	1.38 ± 0.19
MD 3-1	0.8 ± 0.14	0.81 ± 0.15	0.8 ± 0.14
CT 1-1	2.33 ± 0.24	2 ± 0.23	1.71 ± 0.2
UT 1-1	2.11 ± 0.35	2.17 ± 0.35	0.93 ± 0.23
MD 1-1	1 ± 0.2	1.15 ± 0.21	0.92 ± 0.19

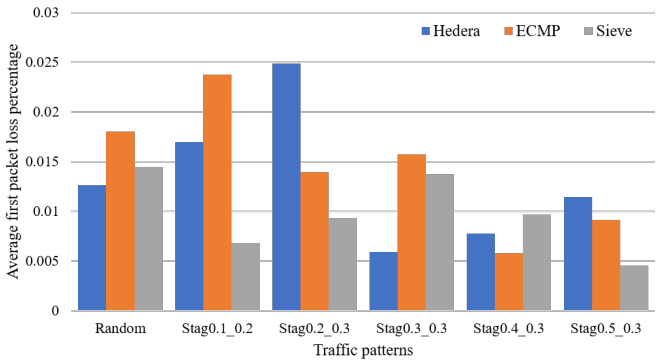

Fig. 8. Average first packet loss

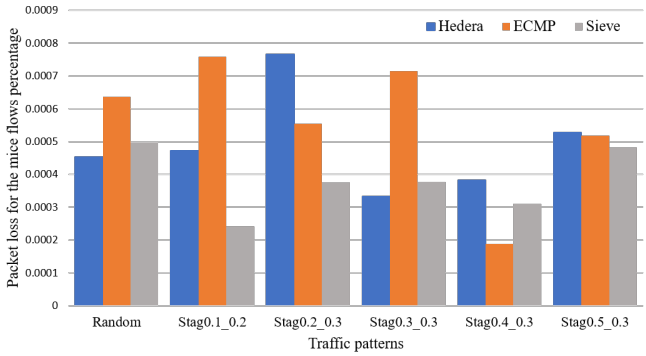

Fig. 9. Average successive packets loss

messages between the end-hosts according to the following traffic patterns:

1) Random: each end-host sends traffic to other hosts in the network according to uniform distribution.
2) Staggered probability ($Edge\_p, Pod\_p$): each end-host sends traffic to another one connected to the same edge switch with probability ($Edge\_p$), to the same pod with probability (Pod_p), and to other pods in the network with probability ($1 - Edge\_p - Pod\_p$).

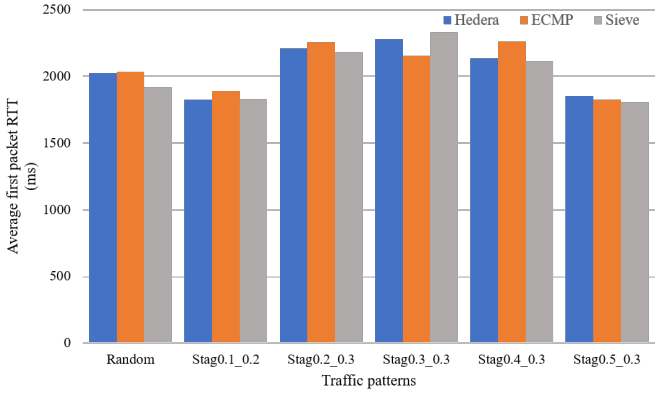

Fig. 10. Average round trip delay for the first packet

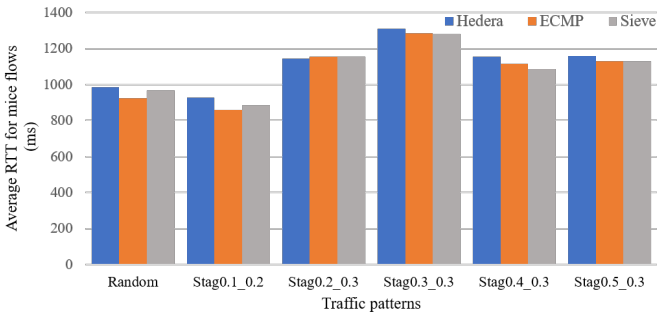

Fig. 11. Average round trip delay for the successive packets

Basically, the traffic pattern presented in [5] has been adopted. We aimed to overwhelm all links by involving all end-hosts in traffic generation by initiating one TCP elephant flow generated by *iperf* for 60 seconds from each end-host. Besides, each end-host sent 600 *ping* echo messages to another end-host according to the traffic pattern. We repeated the experiment 20 rounds for each solution, and we averaged the results. We compared the results of the average first packet loss and the average successive packets loss reported by *ping* as shown in Fig. 8 and Fig. 9, respectively. Sieve outperforms Hedera and ECMP in case of connections spanned all DCN topology layers (e.g., Stag0.1_0.2, Stag0.2_0.3) as shown in Fig. 8 and Fig. 9. In case high $(1 - Edge\_p - Pod\_p)$ value, Sieve can utilize DCN topology to reschedule elephant flows to better paths and minimize the loss rate as a result. On contrary, in case of connections on a same pod, Sieve can not find many alternative paths between the source and the destination. In case of connections between end-hosts connected to the same edge switch, Sieve has no effect as directly connected flow entries will be matched.

Furthermore, we illustrate RTT values reported by *ping* in Fig. 10 and Fig. 11 in case of the average first packet delay and the average successive packets delay respectively. These figures reveal that Sieve provides connections with delay values so similar to that of Hedera and ECMP. In this scenario group, the network is overwhelmed with elephant flows so that it represents the worst case scenario.

V. Conclusion

In this paper, we presented a flow scheduling framework is called Sieve in DCN to improve FCT of mice flows. Our framework has distributed functions among the data and control planes. In addition, it has partial flow visibility to mitigate the overhead in the control plane and to avoid the ECMP-related collisions. Sieve employs group feature of OpenFlow protocol to provide flow sampling, and it probes installed flow entries to identify elephant flows. We compared FCT of mice flows under Sieve to the existing solutions, Hedera and ECMP. We showed that Sieve provides better FCT for mice flows up to 50% in case of uniform traffic distribution without impairing the throughput of elephant flows. We also conducted a second scenario group with a different traffic pattern for further analysis of its impacts in an overwhelmed network in terms of packet loss and delay. We found that Sieve provides better loss rate in case of the connections spanned the three layers of DCN, and it has similar delay to other solutions in term of delay. Moreover, asymmetric DCN is common in today data center, so our future work will cover evaluating Sieve in such an environment.

References

[1] T. Benson, A. Akella, and D. A. Maltz, “Network traffic characteristics of data centers in the wild,” Proceedings of the 10th annual conference on Internet measurement - IMC 10, 2010.

[2] T. Mori, R. Kawahara, S. Naito, and S. Goto, “On the characteristics of Internet traffic variability: spikes and elephants,” CCECE 2003 - Canadian Conference on Electrical and Computer Engineering. Toward a Caring and Humane Technology (Cat. No.03CH37436).

[3] N. Mckeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow,” ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, p. 69, 2008.

[4] A. R. Curtis, W. Kim, and P. Yalagandula, “Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection,” 2011 Proceedings IEEE INFOCOM, 2011.

[5] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, ”Hedera: dynamic flow scheduling for data center networks”, Proceedings of the 7th USENIX conference on Networked systems design and implementation. USENIX Association, pp. 19–19, 2010.

[6] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, “Inside the Social Networks (Datacenter) Network,” ACM SIGCOMM Computer Communication Review, vol. 45, no. 5, pp. 123–137, 2015.

[7] M. Alizadeh, N. Yadav, G. Varghese, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, and R. Pan, “Conga,” Proceedings of the 2014 ACM conference on SIGCOMM - SIGCOMM 14, 2014.

[8] L. Liu, Y. Jiang, G. Shen, Q. Li, D. Lin, L. Li, and Y. Wang, “An SDN-based Hybrid Strategy for Load Balancing in Data Center Networks,” 2019 IEEE Symposium on Computers and Communications (ISCC), 2019.

[9] S. Wang, J. Zhang, T. Huang, T. Pan, J. Liu, and Y. Liu, “Flow distribution-aware load balancing for the datacenter,” Computer Communications, vol. 106, pp. 136–146, 2017.

[10] A. Yazidi, H. Abdi, and B. Feng, “Data center traffic scheduling with hot-cold link detection capabilities,” Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems - RACS 18, 2018.

[11] F. Tang, H. Zhang, L. T. Yang, and L. Chen, “Elephant Flow Detection and Differentiated Scheduling with Efficient Sampling and Classification,” IEEE Transactions on Cloud Computing, pp. 1–1, 2019.

[12] C. N. Sminesh, E. G. M. Kanaga, and K. Ranjitha, “Flow monitoring scheme for reducing congestion and packet loss in software defined networks,” 2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS), 2017.

[13] X. Jingwen, W. Muqing and H. Xiaolan, ”A Traffic Scheduling Scheme for Data Center Networks Based on SDN,” In Proceeding of 2019 IEEE 5th International Conference on Computer and Communications (ICCC), pp. 1417-1422, 2019.

[14] Q. Fu, E. Sun, Q. Wang, Z. Wang, and Y. Zhang, ”A Joint Balancing Flow Table and Reducing Delay Scheme for Mice-Flows in Data Center Networks,” In Proceeding of 2019 IEEE Global Communications Conference (GLOBECOM), pp. 1-6, 2019.

[15] M. Mitzenmacher and E. Upfal, ”Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis”. Cambridge university press, 2017.

[16] Ryu: Ryu SDN Framework. http://osrg.github.io/ryu/ Accessed 12 Mar. 2019

[17] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” Proceedings of the ACM SIGCOMM 2008 conference on Data communication - SIGCOMM 08, 2008.

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

[19] Y.-C. Wang and S.-Y. You, “An Efficient Route Management Framework for Load Balance and Overhead Reduction in SDN-Based Data Center Networks,” IEEE Transactions on Network and Service Management, vol. 15, no. 4, pp. 1422–1434, 2018.