Computers, Materials & Continua CMC, vol.62, no.1, pp.385-398, 2020

An OpenFlow-Based Load Balancing Strategy in SDN

Xiaojun Shi[1], Yangyang Li[2, *], Haiyong Xie[2, 3], Tengfei Yang[2], Linchao Zhang[2], Panyu Liu[4], Heng Zhang[4] and Zhiyao Liang[5]

Abstract: In today's datacenter network, the quantity growth and complexity increment of traffic is unprecedented, which brings not only the booming of network development, but also the problem of network performance degradation, such as more chance of network congestion and serious load imbalance. Due to the dynamically changing traffic patterns, the state-of the-art approaches that do this all require forklift changes to data center networking gear. The root of problem is lack of distinct strategies for elephant and mice flows. Under this condition, it is essential to enforce accurate elephant flow detection and come up with a novel load balancing solution to alleviate the network congestion and achieve high bandwidth utilization. This paper proposed an OpenFlow-based load balancing strategy for datacenter networks that accurately detect elephant flows and enforce distinct routing schemes with different flow types so as to achieve high usage of network capacity. The prototype implemented in Mininet testbed with POX controller and verify the feasibility of our load-balancing strategy when dealing with flow confliction and network degradation. The results show the proposed strategy can adequately generate flow rules and significantly enhance the performance of the bandwidth usage compared against other solutions from the literature in terms of load balancing.

Keywords: Load balancing, OpenFlow, data center network, elephant flow, multi-path routing.

1 Introduction

The current networks are facing tremendous traffic growth with so many new types of network applications emerge rapidly, which turned it to a mega-datacenter era. However, the big data situation is degrading application performance due to the dynamically changing traffic patterns. This calls for more efficient and effective network management technologies, especially in traffic monitoring aspect [Al-Fares, Radhakrishnan, Raghavan et al. (2010)]. As SDN (Software-defined networking), a novel network structure, decouples the control plane and data plane and offers a flexible and programmable idea

[1] Department of Science and Technology, China Electronics Technology Group Corporation, Beijing, China.

[2] National Engineering Laboratory for Public Safety Risk Perception and Control by Big Data, China Academy of Electronics and Information Technology, Beijing, China.

[3] University of Science and Technology of China, Hefei, China.

[4] National University of Defense Technology, Changshan, China.

[5] Macau University of Science and Technology, Avenida WaiLong, Taipa, Macau.

[*] Corresponding Author: Yangyang Li. Email: liyangyang@cetc.com.cn.

for network operators, it enables dynamic, fine-granularity and network-wide traffic scheduling. The new emerging network applications (data center, traffic engineering and etc.) take advantages of it, which significantly increases the importance of SDN-based traffic monitoring.

According to some authoritative DCN traffic researches, flows in DCN are classified as elephant flows and mice flows. The former is long-lived and the latter is short-lived. To be specific, the elephant flow could be video data such as YouTube streams and the mice flows could represent query traffic like Google search. Another report published by Cisco shows that video-based traffic will reach 80%-90% of the overall datacenter traffic in 2019 but only account for 10% of the whole flow population [Basat, Einziger, Friedman et al. (2017)]. The mice flows tend to be more latency-intensive and hard to manage due to the dramatic increment in quantity. Moreover, in order to process tremendous data aggregated from thousands of machines in DCN, multi-rooted trees with equal-cost path is needed based on port densities limitation. This might cause big problems with elephant flows and mice flows are treated equal. So, elephant flow detection and load balancing technologies are very heated topics in academic field [Liu, Liu, Liu et al. (2018)]. And it is important to distinguish elephant flows from mice flow and treat them with different manner.

The mainstream way dedicated on handling elephant flows properly and left mice flows for load-unaware network management methods. Existing flow detection methods can be concluded into four types, which are host-based detection, proactive query detection, passive report detection and sampling. Mahout, a host-based detection method, breaks the limitation of round procedure between controller and switches and create a shim layer on each end-host while monitoring flows, but it needs additional modification on hardware [Wu, Chen, Durairajan et al. (2013)]. Proactive query methods detect elephant flows by periodically querying statistics from switches. This could guarantee correctness, but to cause heavy monitoring costs. Passive report detection upload flow information when reaching the threshold. Considering the complexity and dynamic of network, it can cause severe detection error due to the static threshold [Akyildiz, Lee, Wang et al. (2016)]. Sampling has a long history in traffic monitoring, but the intrinsic defect of inaccuracy and useless samples always exists. Equally, ignoring mice flows through load-unaware network management methods is far from effective and efficient. Mice flow is abrupt and hard to detect in the first place, once encounters busy network, load-unaware method may induce huge flow detection error. To conclude, the aforementioned monitoring methods have the following problems: high bandwidth occupation, large memory and computation overhead, low detecting efficiency, unbearable error confine and etc.

According to these problems, we are motivated to propose an intelligent traffic monitoring method with OpenFlow-based load balancing strategy, aiming at minimizing the bandwidth cost, memory overhead and error confinement. To achieve the goal, we decided to split it into two-stage. First stage, we start the initiation actions by pre-defining a small amount of high bandwidth routing paths under the confinement of minimal hops. This is for elephant flow monitoring when a flow is tagged elephant flow, which we call elephant zone. The rest will be called mice zone. We calculated weight for every available path and arrange the minimal workload path for flows. The weighted route path is based on current workload of switches which can be represented by numbers of

installed flow entries. The higher number of flow entries a switch contained, the lower weight it will get. The second stage is load balancing procedure. We design a persistent classification algorithm to distinguish elephant flows from mice flows in a proactive way. Once a flow is tagged on elephant flow, it will be removed from mice zone to elephant zone and redirected to pre-defined path for further monitoring actions. The mice flow will be routed under the consideration of shedding workload evenly to every switch in the network. In this way, we can maximize bandwidth usage, avoid network congestion and improve traffic monitoring efficiency and accuracy at the same time. An experimental prototype is constructed to verify the use of OpenFlow-based load balancing strategy and evaluate the benefits under the comparison of correctness, effectiveness, bandwidth consumption with other solutions [Liu, Cai, Xu et al. (2015)]. The main contributions of this paper are summarized as follows:

(1) We confine elephant flows with several fixed route paths under the requirement of minimal switch occupations due to its long-lived and less changeable characteristics [Curtis, Andrew, Kim et al. (2011)]. In this way, we expect to avoid network congestions and promote further elephant flow monitoring.

(2) We design weighted route path calculation algorithm for mice flows. This is capable of shedding monitoring workload evenly to all switches, maximizing the use of bandwidth and alleviating the network congestion [Vijay, Vishnoi and Bidkar (2013)]. To be specific, the heavy loaded paths will be allocated a relatively low weight so that fewer traffic will go through these loaded paths.

(3) We fulfill flow detection function by proposing a persistent classification method. Considering the importance of flow detection correctness, we actively poll switches for flow statistics to detect its features and set tags on it with high detection accuracy and efficiency [Marco, Kindler and Schapira (2017)].

The rest of the paper is organized as follows: Section 2 discusses the background and related works and Section 3 formally states the problems and introduce implementation details. Section 4 covers the experimental evaluation of our load-balancing solution and analyze the results. Finally, we conclude this paper in Section 5.

2 Background and related works

2.1 OpenFlow

OpenFlow now becomes the standard communication protocol in SDN network. The basic idea is to deploy network applications without designing new devices, in other words, hardware independency. It designs for devices with inbuilt TCAM storage so that OpenFlow protocol can enable controller to monitor the network and make rules for devices in data plane. In v1.3, the format has changed into pipeline of flow tables, which contains information about transfer actions and other records such as execution times. The minor element will be flow entries, these are instructions for every flow so that controller could conduct fine-granularity management of network [Tang, Li, Barolli et al. (2017)].

The basic operations between controller and switches are as follows. Once the switch launched, the connection will be established to controller, which is called OpenFlow channel. There are proactive and reactive modes. It depends on who takes the first move.

The controller pre-installed flow entries in switches is called proactive mode. It can happen at the whole functioning time and it doesn't need to send flow entries directly after the OpenFlow channel established. The reactive mode is opposite [Ramon, Katrinis and Muntean (2017)]. Once the OpenFlow switches receive a new packet and no flow entries match it, the switch will encapsulate it in Packet_In message and forwarded it to the controller over OpenFlow channel. The controller responds with a flow table and the original packet, then the switches finish the rest execution as instructed [Wang, Wang, and Yan (2016)]. Also, if a flow action set contains an option of forwarding to controller, the flow packet will also be sent to controller, which we call it specialized reactive action.

2.2 SDN-based load balancing technologies

Datacenter network is facing high workload for most times, which relies on multipath planning between pairs of end hosts to share burden among links and avoid network congestions as much as possible [Lan and Heidemann (2003)]. In this end, load balancing is needed for network traffic management. The most commonly used routing algorithm is OSPF (Open Shortest Path First), which pre-set the shortest path between node pairs and all destined traffic will go through soon afterwards. However, OSPF doesn't provide load balancing strategy over multipath situation, which may cause insufficient bandwidth usage and network congestion [Xu and Li (2014)]. In the meantime, flow monitoring error can be magnified due to poor bandwidth condition.

ECMP (Equal-Cost Multi-Path) is a novel way to compensate the defect of bandwidth waste. It adopts the idea of OSPF to pre-calculate shortest routing path but making it multiple for further traffic distribution [Cai, Wang, Zheng et al. (2013)]. All road is statically decided without a real-time response from the network, which cannot guarantee the effectiveness of load balancing [Nathan, Porter, Sivasankar et al. (2010)]. What's more, ECMP doesn't differentiate between elephant flows and mice flows, and fail to fully utilize bandwidth. TinyFlow fixed the problem by breaking elephant flows into a large number of mice flows, flexibly arrange these flows across the network. TinyFlow presents a per-mice routing approach and automatically change the traffic characteristics, which alleviates head-of-line block caused by egress buffer full of elephant flow packets [Brownlee and Claffy (2002)]. But breaking and aggregating elephant flows may create a large number of mice flows and causes unexpected error based on disorder and other situations happened in the network, and it is not a good way to have a good visual of the network.

2.3 Flow detection solutions

Many literatures have addressed the importance of flow detection, especially on network congestion occasion. It may cause much more trouble for flow monitoring and the elephant flows are the chief culprit. As mentioned before, there are many solutions for elephant flow detection. In this section, we are giving a brief introduction for up-to-date flow detection technologies and their features.

Hedera is an OpenFlow-based flow management solution which detects elephant flows by per-flow statistics query. The idea is simple. In first step, all flows are recognized as mice flows and forwarded through equal-cost paths [Lin, Chen, Chang et al. (2014)]. And the switches are gathering the information during the process. When elephant flow

demand is detected, controller will compute good path and instruct switches to redirect elephant flows. Based on periodically polling switches during the detect time, too much statistics of flows may cause OpenFlow channel congestions. ESHSP (Elephant Sensitive Hierarchical Statistics Pulling) detects elephant flows by combining aggregation and individual statistical messages based on OpenFlow protocol. ESHSP sent query messages to switches with aggregate statistical request, when the flow size reaches the threshold, it then divided the aggregated flows into smaller ones by changing the IP range [Dias, Esteves, Granville et al. (2017)]. The iteration stops when elephant flows are detected. The core part is elephant store and range split, which avoid counting known elephant flows and enhance new elephant flow detection. ESHSP reduces bandwidth consumption and process time, but it fails to offer further actions. The ignorance of mice flows in also an inevitable defect.

Mahout breaks the limitation of elephant flow detection by shifting the detection burden to end hosts. Mahout adds a shim layer to every end host for flow monitoring. When elephant flow detected, the flow packet will be marked, and switches will forward the marked packet to controller for best transfer paths, while mice flows will be managed in a load-unaware way. Though this method can reduce overhead for controllers and enhance feasibility of elephant flows, it needs modifications for every end hosts which kills mahout for further promotion. DevoFlow is a flow management solution that differentiates between mice flows and elephant flows and report only elephant flows to the DevoFlow controller. In this way, the controller can only have the visibility of elephant flows [Cui, Zhang, Cai et al. (2018)]. The detail of the detection is to monitor elephant flows at the edge switch by setting a threshold for transfer bytes. It keeps flow in data plane at their best effort and concentrate more on elephant flows while leaving mice flows for randomly selected microflow path. This static threshold way is more subjective not precise when making detection decision.

NetFlow and sFlow are very typical sampling flow detection scheme, which devoted to achieving fine-grained flow measurement by conducting packet sampling. The basic idea is to mandate switch capture packet at a certain rate. Once the packet is captured, the header will be forwarded to controller with an extra metadata containing necessary information such as capture timestamp, sampling frequency, port number etc. This information may help deduce the probabilistic result of flow information. Estan and Varghese focus on elephant flows by using random samplings. It offers two algorithms by conducting multistage filter, sample and hold. In the final step, it keeps a certain memory in switches to contain flow statistics. However, it is not suitable for network with high volume traffic, because it can't afford the memory usage with SRAM. OpenSample is a low-latency, sampling-based traffic measurement platform, devoted to accelerating control loops for software-defined network. It relies on TCP sequence numbers of sampled packets and divide the difference by time to get the result. It is near real-time for both workload and individual flows. After a flow is classified as elephant, it is rerouted to another path using a global first fit algorithm [Tan, Liu, Xie et al. (2018)]. Comparing with polling-based solutions, it is far more efficient, however it sacrifices the precision of flow detection and more suitable for micro flows through aggregation.

In summary, current proposed solutions for traffic monitoring in the context of datacenter

networks have their advantages in promoting bandwidth usage [Liu, Guo, Cai et al. (2019)], alleviating network congestion and confining result error rate. However, their limitations are obvious, which guarantee performance of one aspect by sacrificing the other (shown in Tab. 1). Inspired by the above approach and take those pros and cons into consideration, we propose an intelligent traffic monitoring technology with OpenFlow-based load balancing strategy in order to solve the aforementioned problems effectively. We separate different zones for elephant flows and mice flows, treating them with customized strategy, as well as a suitable flow classification method to facilitate the further flow monitoring and guarantee the correctness and effectiveness.

Table 1: Summary of four flow detection types

Methods	Tech Names	Shortcomings	Overhead	Accuracy
Proactive pulling	Hedera, ESHSP	High bandwidth consumption high and statistics overhead	High	High
Reactive detection	FlowSense	Not suitable for time sensitive applications	Low	Low
Host-based detection	Mahout, DevoFlow	Need modification in edge devices	Low	Medium
Sampling	NetFlow, sFlow, OpenSample	Costs in proportion to accuracy	Medium	Medium

3 Problem formulation and implementation

In this section, we define the general running datacenter network model and formulate the inevitable load balancing problem and introduce our solutions.

3.1 Problem formulation

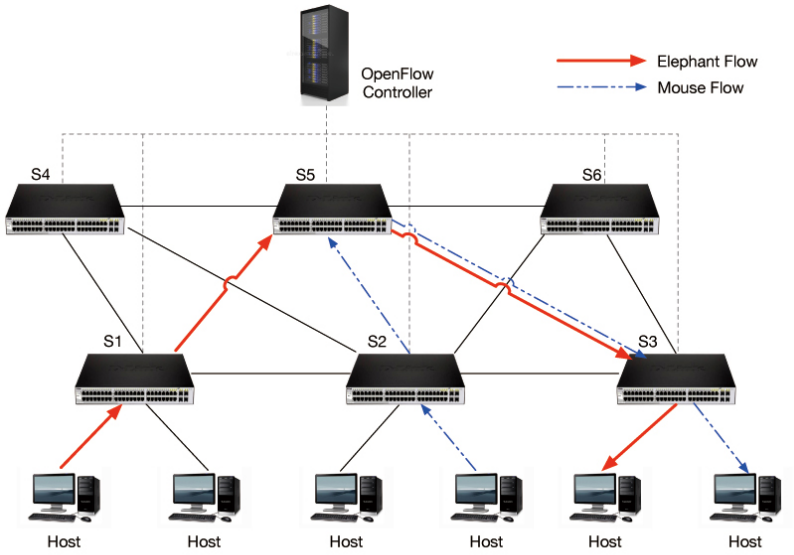
A datacenter network topology is commonly represented by a connected graph *G= (E, V)*, *V* is the set of nodes and *E* is the set of direct links between nodes. Usually, a path is denoted by finite distinct nodes, $p = (v_0, v_{1\ldots}v_n), s.t. \forall i \in [0, n], (v_i,\ v_{i+1}) \in E$. This routing path is decided by OSPF in a traditional way. Moreover, in datacenter network, it's a huge waste for other links in idle state and the mainstream path may suffer from data torrent, which accelerate the improvement of multi-path routing technologies. Based on ECMP, $P_{i,j}$ represents the set of equal cost paths between source node *i* and destination node *j*, and $\forall p \in P_{i,j}, e \in E, \exists k \in (0, K), k = Hash(flowID)\%\theta$, which *K* is the number of *p* in $P_{i,j}$, *k* is the selected number of *p* and is the selection factor for flowIDs, due to the equal weight of every path. However, ECMP only function well when network flow sizes are similar and may cause capacity constraints. The following definitions rigorously illustrate the problem.

Theorem 1. Given a network topology *G= (E, V)* and $\forall p \in P_{i,j}$, the available capacity C_p of path *p* is the minimum c_e of all links in the path. $C_p = Min\ c_e, \forall e \in E$.

Theorem 2. Given a network topology *G= (E, V)* and current link flow bandwidth f_e, the link congestion factor is denoted by f_e/c_e.

Theorem 3. Given a network topology *G= (E, V)*, a flow capacity f_p and the appear time σ of link *e* among all flow paths. Thus, the capacity constraint is $\forall p \in P_{i,j}, e \in E, \sum_1^\sigma f_p \approx C_p < c_e$.

It is strongly believed that hash collision plays a key role for network congestion which some links of the path reaches the maximum bandwidth while others not and result in low bandwidth utilization (shown in Fig. 1). In case (a), S_5 and S_3 run into downstream link congestion and in case (b), S_1 and S_5 run into upstream link congestion.



(a) Remote Downstream Collision

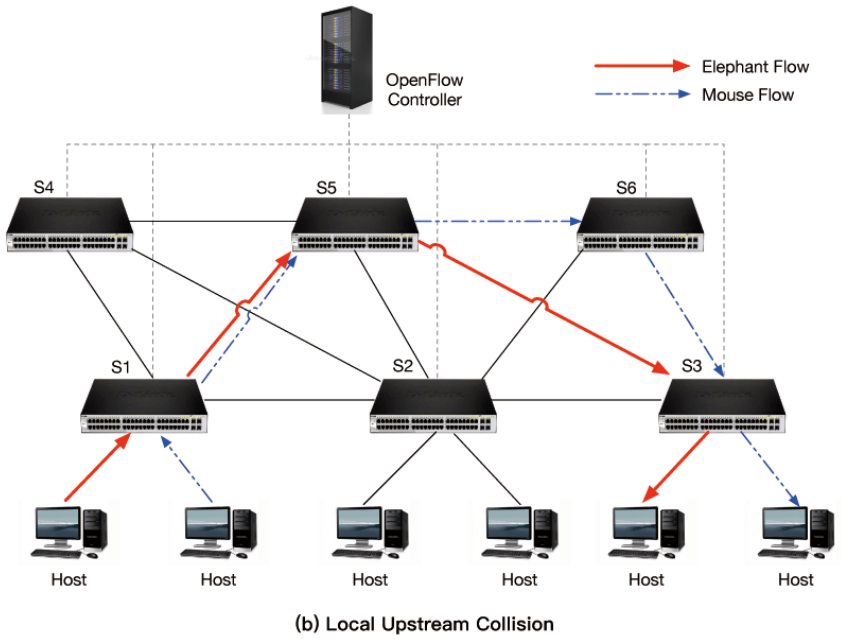


Figure 1: Problem formulation

3.2 Architecture design

In this section, we will first give the ideal result we want to achieve and then analyze the detail implementation of our architecture and solutions. As aforementioned problems illustrated before, we'd like to vividly share our opinion through Fig. 2. The flows are scheduled properly and can effectively avoid network congestion.

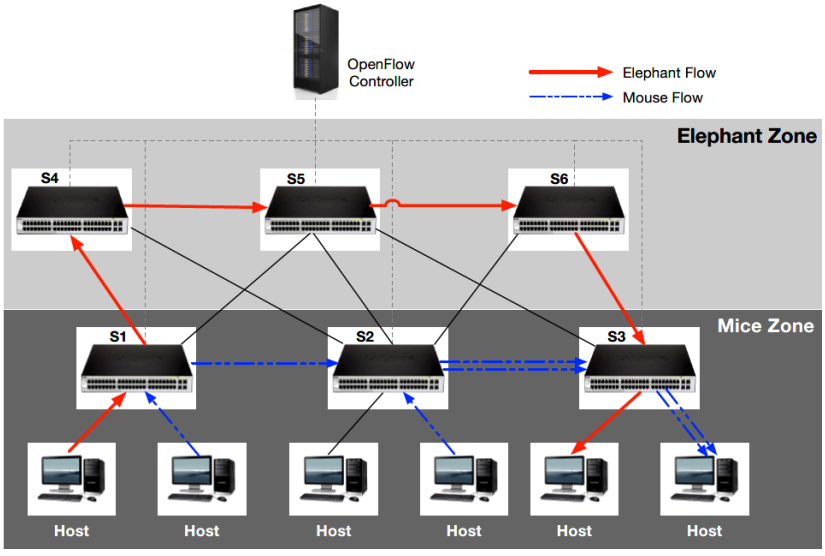


Figure 2: Expected result for load balancing implementation

We notice the defect and propose an OpenFlow-based load balancing strategy to mitigate the problems and enhance traffic monitoring effect. There are two benefits that we can achieve. The first is intellectual flow detection for elephant and mice flows. The second is elephant/mice zone separation and distinct flow scheduling for different flow types. As we mentioned before, the root of problem is flow collision between elephant and mice flows. We need to adjust measures to specific flow conditions so as to avoid bandwidth congestions based on flow classification. The whole procedure can be divided into three stages.

Stage one-elephant/mice one separation. The first stage is conducting the initialization job. The objective is to reserve one or two idle paths in advance for elephant flow placement, with which these paths requires minimum hops and minimum workloads. This work should be done directly after controller and switches finished OpenFlow channel built. At the Controller side, it will compute the set of minimal cost paths, based on the minimum number of hops between source and destination nodes and minimum link congestion factors. Once the idle path is reserved, the following elephant flows will go through smoothly without any retardation when flow rules are inserted to switches.

To be specific, once a flow is tagged, the flow path will be fixed until the flow is expired or paths condition in elephant zone need a rearrangement.

Stage two-elephant flow detection. As we do not know any information of flows initially, we coarsely aggregate the ingress flows and forward them by using ECMP. So, when a new flow starts, the default switch action is to encapsulate it in a Packet In message to forward it based on a hash on the 5 tuples along the minimal-weight path. Based on OpenFlow protocol, flow entries contain several important counters for flow statistics, we focus on Received Packets, Received Bytes, Duration (msec) from Per Flow Counter and Per Queue Counter. Once the flow grows past a threshold rate, switches will actively report it to Control Logic in controller, and the flow will be tagged and redirected to elephant zone, by which the preset flow rules will be installed to corresponding switches in elephant zone (shown in Fig. 3).

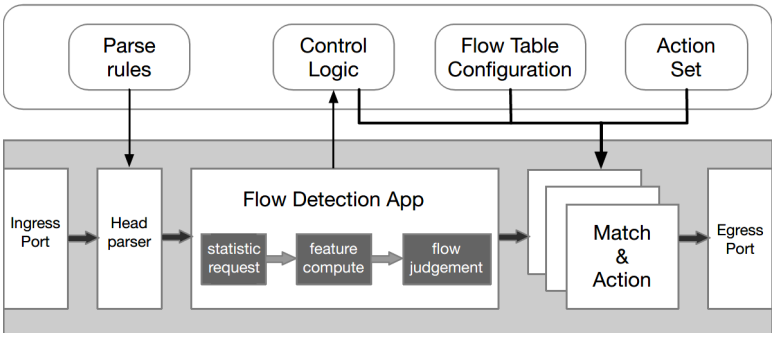


Figure 3: Elephant flow detection

As for the preset threshold, used as a core flow judgment, based on current research about 90% of bytes in datacenters are generated by elephant flows and their duration time is much longer than mice flows, the controller will actively query flow statistics from switches and believe 100 Mbps of a flow throughput within any time interval (10% of each host 1 GbE per link) is appropriate in our implementation.

Stage three-load balancing. After the second stage, all tagged flows will traverse through elephant zone. However, the remaining flows also need a rearrangement schedule, according to the large amount of flow population. At the controller site, the path with lowest bandwidth occupation and minimum hops will be the potential selection. To take full advantages of datacenter network bandwidth utilization and guarantee network performance, the mice zone will generate the set of paths with corresponding weight respectively. Here, we propose a method of path weight W_p (given in Definition 4) calculation for path condition judgement.

Theorem 4. Given a network topology *G= (E, V)* and $\forall p \in P_{i,j}, e \in E$, the weight of a link w_e relates to link congestion factor as described in definition 2. So $w_e = 1 - f_e/c_e$. The weight of a path $W_p = (\sum_1^H w_e)/K$. To be specific, *H* is the number of hops, *K* is the number of p in $P_{i,j}$ and $\sum_1^K W_p = 1$.

Once the weight is computed, the path weight is updated to the Action Set module and stored in a special bucket group. And based on its short-lived feature, all mice flow with scheduled path will persist until the flow entry reaches a hard/soft timeout, which means the flow expired. To this end, flows with different tags will run in separate zones and perfectly avoid flow collision and network congestion (Shown in Fig. 4).

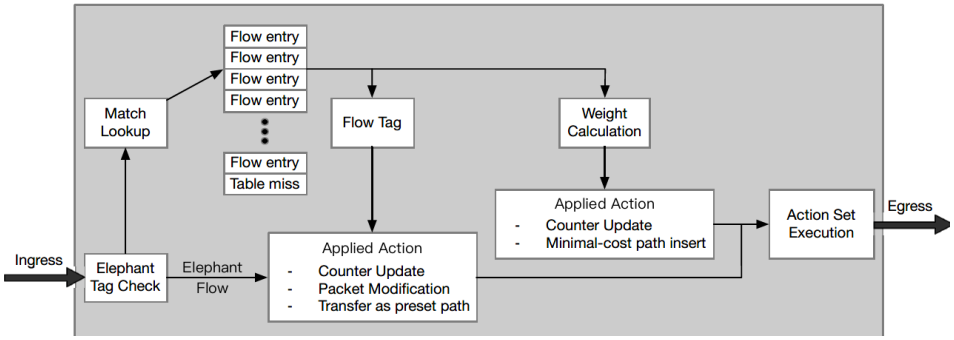


Figure 4: Load balancing strategy

4 Experimental evaluation

This section describes the experimental scenarios, presents the results and performs result analysis. The aim of this section is to validate the benefit of our strategy in terms of routing elephant and mice flows to their own zones according the generated flow rules so as to enable load balancing. We begin by introducing the experimental scenario and the methodology used in the tests. Following that, we present and discuss the main results achieved so far.

4.1 Scenario

We emulate the topology (shown in Fig. 2) by using the Mininet emulator running in a virtual machine (VM), which is hosted in a Core i5-6360 MacBook Pro 13 Laptop with 8 GB RAM through VMfusion. OpenVSwitch (OVS) is used for it perfectly supports every version of OpenFlow implementation. The POX Controller is hosted in the same VM with the controller IP address is set 127.0.0.1 and port number 6633 for communication

convenience. In this scenario, the fabric consists of 6 OpenFlow-based OVS and 6 hosts generating traffic inside each one. Each switch is a 16-port 1 Gbps switch, and the network has full bisection bandwidth.

To be specific, we prefer empirical workloads close to traffic condition in reality. We consider the traffic follows Poisson distribution from a datacenter mostly running web search, which is heavy-tailed with a mix of mice and elephant flows. In this workload, we approximate 80% of the bytes are from 20% of the flows whose throughput is larger than 100 Mbps, each flow is composed of a sequence of packets which share the same 5-tuple (Source Port, Destination Port, Source IP, Destination IP, Protocol Type). The workload increases gradually to evaluate our load-balancing performance in different conditions. We also implemented the ECMP routing algorithm for performance comparison with ours.

4.2 Results and discussion

Our goal is to locate the problem of network congestion and compare the network performance, here we choose latency and throughput metrics used as reflection of network congestion degree. As we increase the workload by adjusting the proportion of elephant flows in network traffic, ECMP can control latency to some extend and achieve relative high saturation point, but when elephant flow proportion goes higher, ECMP will run into serious flow collision and latency rises. Latency still rises in our Load-balancing strategy as elephant flows account for much of the whole network traffic, but it controls the latency well and performs better than ECMP (As shown in Fig. 5).

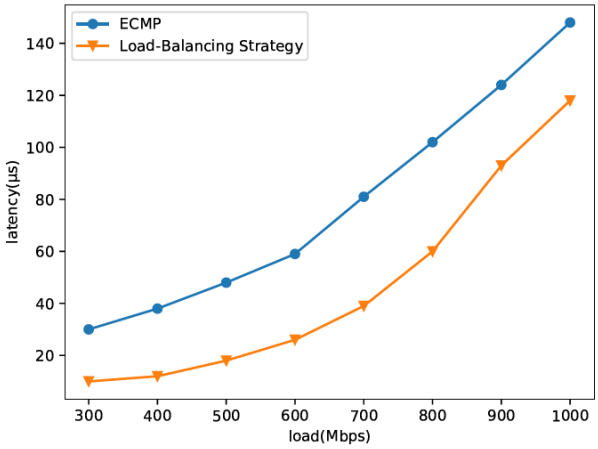


Figure 5: Network latency comparison between ECMP and our load balancing strategy

Also, we still need an intuitive view of bandwidth utilization performance under different traffic conditions (shown in Fig. 6). In first stage, utilization increases near linearly for both routing strategies. As traffic load keeps increasing, ECMP takes the lead of reaching the bottleneck, network performance degrades and bandwidth utilization drops, while our load-balancing strategy meets the bottleneck later than ECMP and the maximum value is higher than ECMP. To be more specific, when workload starts from zero and reaches the maximum, for ECMP the link utilization rate can reach 73% at most and averages to 62%, for our load-balancing strategy, the maximum utilization rate is near 90% and the average

utilization can stabilize in 78%. The result shows that our load-balancing strategy has a better performance than ECMP in dealing with network traffic with diversity.

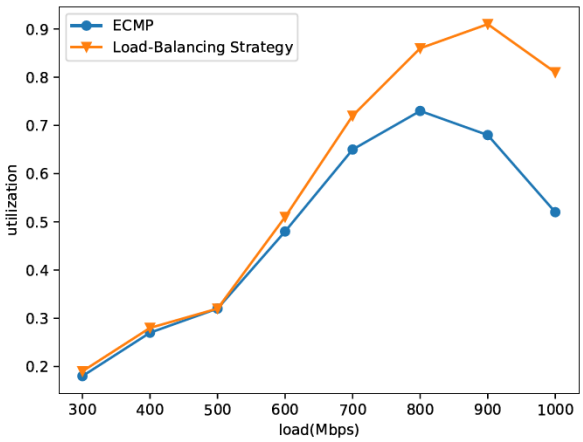


Figure 6: Utilization performance comparison between ECMP and our load balancing strategy

5 Conclusion

In this paper, we proposed the design and evaluation of an OpenFlow-based load-balancing strategy that logically split elephant zone and mice zone for different flow types. This solution naturally shed elephant traffic load on elephant zone with several pre-defined idle paths while the rest is confined in mice zone with minimal weighted path, which is calculated in accordance with current load conditions. We implement the prototype in Mininet testbed with POX controller and verify the feasibility of our load-balancing strategy when dealing with flow confliction and network degradation. The simulation shows that our solution can effectively alleviate network congestion and improve network utilization in comparison with ECMP.

Acknowledgement: This work was supported by the CETC Joint Advanced Research Foundation (Grant Nos. 6141B08010102, 6141B08080101) and the National Science and Technology Major Project for IND (investigational new drug) (Project No. 2018ZX09201014).

References

Al-Fares, M.; Radhakrishnan, S.; Raghavan, B.; Huang, N.; Vahdat, A. (2010): Hedera: dynamic flow scheduling for data center networks. *Proceedings of USENIX Conference on Networked Systems Design and Implementation.* USENIX Association, Berkeley, CA, USA.

Akyildiz, I.; Lee, A.; Wang, P.; Luo, M.; Chou, W. (2016): Research challenges for traffic engineering in software defined networks. *IEEE Network*, vol. 30, no. 3, pp. 52-58.

Basat, R. B.; Einziger, G.; Friedman, R.; Kassner, Y. (2017): Optimal elephant flow detection. *IEEE International Conference on Computer Communications*, *Atlanta, GA,*

USA, pp. 1-9.

Brownlee, N.; Claffy, K. C. (2002): Understanding internet traffic streams: dragonflies and tortoises. *IEEE Communications Magazine*, vol. 40, no. 10, pp. 110-117.

Cai, Z. P.; Wang, Z. J.; Zheng, K.; Cao, J. N. (2013): A distributed TCAM coprocessor architecture for integrated longest prefix matching, policy filtering, and content filtering. *IEEE Transactions on Computers*, vol. 62, no. 3, pp. 417-427.

Cui, J. H.; Zhang, Y. Y.; Cai, Z. P.; Liu, A. F.; Li, Y. Y. (2018): Securing display path for security-sensitive applications on mobile devices. *Computers, Materials & Continua,* vol. 55, no. 1, pp. 17-35.

Curtis, A. R.; Kim, W.; Yalagandula, P. (2011): Mahout: low-overhead datacenter traffic management using end-host-based elephant detection. *International Conference on Computer Communications*, vol. 11, pp. 1629-1637.

Dias, K. L. A.; Esteves, R. P.; Granville, L. Z.; Tarouco, L. M. R. (2017): Mitigating elephant flows in SDN-based IXP networks. *IEEE Symposium on Computers and Communications*.

Farrington, N.; Porter, G.; Radhakrishnan, S.; Bazzaz, H. H.; Subramanya, V. et al. (2011). Helios: a hybrid electrical/optical switch architecture for modular data centers. *ACM International Conference on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, vol. 41, no. 4, pp. 339-350.

Lan, K.; Heidemann, J. (2003): On the correlation of internet flow characteristics. *Technical Report ISI-TR-574, USC/ISI*.

Lin, C. Y.; Chen, C.; Chang, J. W.; Chu, Y. H. (2014): Elephant flow detection in datacenters using OpenFlow-based hierarchical statistics pulling. *IEEE Global Communications Conference*.

Liu, S. H.; Cai, Z. P.; Xu, H.; Xu, M. (2015): Towards security-aware virtual network embedding. *Computer Networks*, vol. 91, no. 4, pp. 151-163.

Liu, F.; Guo, Y. T.; Cai, Z. P.; Xiao, N.; Zhao, Z. M. et al. (2019): Edge-enabled disaster rescue: a case study of searching for missing people. *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 5, pp. 36-49.

Liu, Y. X; Liu, A. F; Liu, X.; Huang, X. D (2018): A statistical approach to participant selection in location-based social networks for offline event marketing. *Information Sciences*, vol. 480, pp. 90-108.

Liu, F.; Tang, G. M.; Li, Y. H. Z.; Cai, Z. P.; Zhang, X. Z. et al. (2019): A survey on edge computing systems and tools. *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1-26.

Marco, C.; Kindler, G.; Schapira, M. (2017): Traffic engineering with equal-cost-multipath: an algorithmic perspective. *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 779-792.

Ramona, T.; Katrinis, K.; Muntean, G. M. (2017): OFLoad: an OpenFlow-based dynamic load balancing strategy for datacenter networks. *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 792-803.

Tan, J. W; Liu, W.; Wang, T.; Xiong, N. N; Song, H. B. et al. (2019): An adaptive

collection acheme based matrix completion for data gathering in energy-harvesting wireless sensor network. *IEEE Access*, vol. 7, pp. 6703-6723.

Tan, J.; Liu, W.; Xie, M.; Song, H.; Liu, A. et al. (2019): A low redundancy data collection scheme to maximize lifetime using matrix completion technique. *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, pp. 5-13.

Tang, F. L.; Li, L.; Barolli, L.; Tang, C. (2017): An efficient sampling and classification approach for flow detection in SDN-based big data centers. *IEEE International Conference on Advanced Information Networking and Applications*.

Teng, H. J; Liu, Y. X; Liu, A. F; Xiong, N. N.; Cai, Z. P. et al. (2018): A novel code data dissemination scheme for internet of things through mobile vehicle of smart cities. *Future Generation Computer Systems*, vol. 94, pp. 351-367.

Vijay, M.; Vishnoi, A.; Bidkar, S. (2013): Living on the edge: monitoring network flows at the edge in cloud data centers. *International Conference on Communication Systems and Networks*.

Wang, H.; Wang, Y.; Yan, Y. J. (2016): A distributed network traffic monitoring platform based on SDN. *Electric Power Information and Communication Technology*, vol. 14, no. 10, pp. 22-27.

Wu, W. F.; Chen, Y. Z.; Durairajan, R.; Kim, D. C.; Anand, A. et al. (2013): Adaptive data transmission in the cloud. *IEEE/ACM International Symposium on Quality of Service*.

Xu, H.; Li, B. C. (2014): TinyFlow: breaking elephants down into mice in data center networks. *IEEE International Workshop on Local & Metropolitan Area Networks*.