International Journal of Electrical and Computer Engineering (IJECE)
Vol. 9, No. 4, August 2019, pp. 3203~3211
ISSN: 2088-8708, DOI: 10.11591/ijece.v9i4.pp3203-3211

Traffic management with elephant flow detection in software defined networks (SDN)

Hnin Thiri Zaw[1], Aung Htein Maw[2]
[1]University of Computer Studies, Myanmar
[2]University of Information Technology, Myanmar

Article Info

Article history:

Received Jul 23, 2018
Revised Mar 23, 2019
Accepted Apr 3, 2019

Keywords:

Bandwidth utilization
Elephant flow
End-to-end delay
Multipath
SDN
sFlow

ABSTRACT

Multipath routing is to distribute the incoming traffic load among available paths between source and destination hosts. Instead of using the single best path, multipath scheme can avoid the congested path. Equal Cost Multi-Path (ECMP) performs the static traffic splitting based on some tuples of the packet headers. The limitation of ECMP does not consider the network parameters such as bandwidth and delay. Unlike the traditional networks, Software-Defined Network (SDN) has many advantages to support dynamic multipath forwarding due to its special characteristics, such as separation of control and data planes, global centralized control, and programmability of network behavior. In this paper, we propose a new architecture design for dynamic multipath-based traffic management approach in the SDN, which comprises of five components: detecting long (elephant) flow, computing shortest paths, estimating end-to-end delay and bandwidth utilization, calculating least cost path and rerouting traffic flow from the ongoing path to the best path. The simulation environment is created through the usage of Mininet emulator and ONOS controller. The evaluation outcomes show that the proposed traffic management method outperforms the ECMP and reactive forwarding method for both TCP and UDP traffic.

Corresponding Author:

Hnin Thiri Zaw,
University of Computer Studies,
No. (4) Main Street 4, Yangon, Myanmar.
Email: h.thirizawucsy@ucsy.edu.mm

1. INTRODUCTION

New technologies have changed because the nature of networking infrastructure has been more and more complicated. For example, the cloud computing and massive data centers demands have made effective networking much more complex. To adapt to these requests, network administrators need their systems to be smarter and they need to have the capacity to better control and manage them. Accordingly, software-defined networks (SDN) become the new emerging infrastructure to address these issues.

The software-defined network (SDN) architecture, which separates the data forwarding layer and control layer, permits network administrators to manage the network services though abstraction of functionality with an external entity (controller), which can alter the forwarding behavior of the network components specifically. Since the SDN is a network infrastructure with high adaptability, network operators can manage greatly the SDN-enabled switches by the programmability. Due to the combination of virtualization and solidification, network operation costs can be eliminated, by optimizing resource usage and decoupling between control and data planes through centralization.

OpenFlow [1] exchanges control data of network traffic between the data forwarding devices (such as switches and routers) and network operators. In an OpenFlow network, the OpenFlow controller manages routing decisions instead of the forwarding devices, as in traditional network. Therefore, the utilization of

forwarding devices CPU usage can then be saved for faster packet processing and other functions. Administrators can drive with the SDN controllers to more implicitly run the networks.

Most of conventional routing protocols are enhanced to choose a default single shortest path that can cause significant resource underutilization in multi-rooted tree topologies. To address the limitation of single shortest path problem, Equal Cost Multi-Path (ECMP) [2] approach has been proposed. ECMP is utilized as multipath algorithm in data center networks, where a new incoming flow between a pair of hosts is routed over one candidate shortest path among available paths, which is chosen by computing the hashing of some attributes of header fields. With ECMP, two long flows can collide being routed over the same output port of the switch initiating hot-spots in the network. As a result, the network throughput degradation and the latency of path travel across the congested link increases, increasing the flow completion time (FCT).

The SDN controller gathers the network statistics from the switches periodically and determines whether congestion happens. Depending on the congestion point, current and new flows are redirected to least load path [3-6]. As described in [3-6], when a particular link utilization exceeds a certain threshold, the existing flow is rerouted to least load link. In this study, controller collects switch ports statistics and poll at regular intervals of time by using OpenFlow statistics request and reply messages. The controller uses the shortest path algorithm. When congestion occurs, it can reroute some flows to lessen loaded paths. In [3–6], the proposed schemes are not differentiated long (elephant) flow and short (mice) flow. In general, there are two terms in flow size as elephant flow and mice flow. Elephant flow means long flow, which carries a large amount data and consumes many network resources, such as VM migration. Mice flow means small flow, which carries a small amount of data such as web surfing. To become efficient traffic management, SDN controller only needs to consider on the elephant flows that have a great incurs for the network performance. A large number of mice flows come and go too fast to wait the flow entries installation according to controller's policy. Therefore, differentiating elephant and mice flow is critical to make the optimized routing policy for various types of traffic flows [7]. In addition, [4, 6] can be applied only for UDP traffic flows.

Authors in [8] proposed a local re-routing approach that considers the idea of flow classification whether these are elephants or mice. The elephant flow is marked when the flow size is at least 100kB. In the event of congestion, only the elephant flows are re-routed while the mice flows are allowed to proceed. In addition, the redirecting is applied locally at the congested hop or before one instead of redirecting the elephant flows beginning from the sender. To calculate the least load path between any pair of end hosts, port statistics are gathered by the controller from all the connected OpenFlow switches. When the link load exceeds 75% of link capacity, the link is identified as the congested path and elephant flow is rerouted to the least load path. Hedera [9] is a flow scheduling scheme based on fat-tree networks to solve ECMP flow collision problem. The per-flow statistics are collected periodically at the edge switches to mark elephant flows. To place the elephant flow from congested path to less active flows path, it compares two algorithms: Global First Fit and Simulated Annealing. Although Hedera outperforms than ECMP, it has poor scalability due to collect per flow statistics periodically from SDN controller. The proposed schemes in [5, 6] incurs poor scalability and overhead in terms of messages for collecting per table, per flow and per port statics in every pair of end-hosts periodically. In [10], our previous study only focuses on end-to-end delay and elephant flow is rerouted to the least delay path. This study can handle only TCP traffic flow and simulation environment is based on simple topology, not fat-tree. In this paper, our proposed method can adapt in also fat-tree topology and handles both TCP and UDP traffic at the same time. Most of the studies on flow re-routing have focused on bandwidth utilization (current load of the network); however, they do not consider network latency (end-to-end delay) problem. When the network has high latency, data transmission time will take a long time. Long data transmission time causes bottlenecks in the network nodes. Therefore, this paper mainly focuses on not only bandwidth utilization but also network latency to tackle the congestion problem.

In this paper, we propose a new traffic management method that redirects the elephant flows from the ongoing path to the least cost path by measuring end-to-end delay and bandwidth utilization. There are two main tasks in our proposed method. The first task is to differentiate elephant flow and mice flow. Second, when the new flow becomes elephant flow, it is rerouted to the least cost path. Otherwise the route decision for new flow is made by reactive forwarding method [11] which is a default application in ONOS [12] controller. The goals of our study are as follows:

- To design an effective traffic re-scheduling scheme in SDN by estimating end-to-end delays and bandwidth utilization in order to avoid link congestion.
- To improve network throughput and to reduce flow completion time (FCT) by rerouting elephant flows, which have an impact on network performance over a period of time, from ongoing path to least cost path.

Moreover, we test and evaluate the functionality of flow management scheme in k=4, three-layer fat-tree topology which is commonly used in data center network. The experimental result trend proves that the proposed method can give advantages in the several scenarios as compared with ECMP and the reactive forwarding method.

2. BACKGROUND THEORY

In this section, we describe the core concept of SDN and traffic rerouting to tackle the network congestion problem.

2.1. Software-defined networking (SDN)

The rapid adoption of OpenFlow [1] and Software Defined Networking (SDN) has introduced significant changes in today's enterprise datacenter network architectures and revenue models. In the OpenFlow network, each network device maintains a flow table. OpenFlow match and handling the packets of traffic flow by controller defined rules or pre-defined rules. A basic rule of OpenFlow includes match fields (some tuples of header), counters and actions. Every first packet of new arrival flow has to match with matching fields of flow table, and the matching domain fields consists of, Ethernet layer to transport layer, eg; the source/destination MAC address, protocol type and source/destination IP address, or source/destination TCP/UDP port number, etc. When packets successfully match a rule, it will refresh the corresponding statistical data fields (counters) of flow rule firstly, and then take corresponding actions of the rule. Only OpenFlow version 1.0 has a single flow table and above has pipeline of multiple flow tables. When a packet is in the pipeline of the final flow table, the packet can be forwarded to an output port, modified a particular field the packet, dropped the packets, etc.

Compared with the traditional network, SDN network basically has the following advantages: (1) SDN controller can gather the state statistics of the whole network, and make network traffic admission control and dynamic routing in real time with a global network view, (2) SDN makes network administration with software programmability, and greatly simplifies the network innovation with better adaptability [13].

In general, data forwarding in SDN can be accomplished in a reactive or proactive way. In reactive way, the forwarding decision is made whenever a new flow arrives at each switch [11] along the path. The main process is that when the new flow arrives at the switch, the switch sends a copy of the first packet header from new flow to the controller and then the controller installs forwarding rule to the switch. Proactive forwarding, installs flow rules to switches along the path proactively, can reduce the communication time between controllers and switches.

2.2. Flow re-routing (or) per-flow multipath routing

Fat-tree topology is a well-known data center network topology, which contains various paths between end hosts, so it can give higher accessible data transmission than a single path tree with a similar number of nodes. It is normally a 3-layer various leveled tree that comprises of switches on the edge, aggregate, and core layers. Among multiple links, congestion problem can still occur when one or more elephant flows which same have output ports across the same link as shown in Figure 1. Therefore, maximizing network throughput and minimizing transferring latency are two critical targets. In order to achieve them, multiple flows through a bottleneck switch towards a common destination can be diverted through the multi-paths. The main concept of flow re-routing, as shown in Figure 2, is that the existing flow or new flow is re-rerouted from congested path to uncongested path in order to improve network performance.

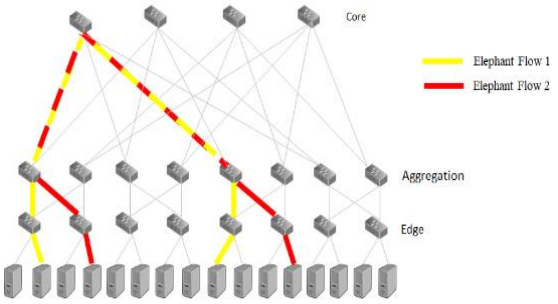

Figure 1. Flow collision problem of ECMP

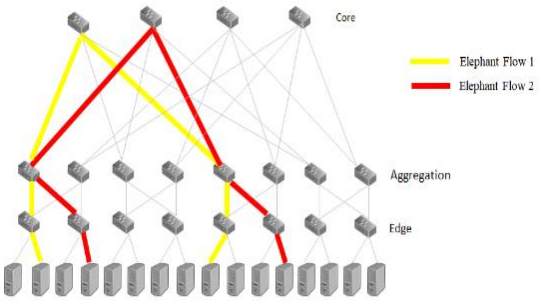

Figure 2. Example of flow re-routing

3. SYSTEM DESIGN

This section describes the overall design of the proposed flow management method as shown in Figure 3. There are five main components in proposed rerouting method.

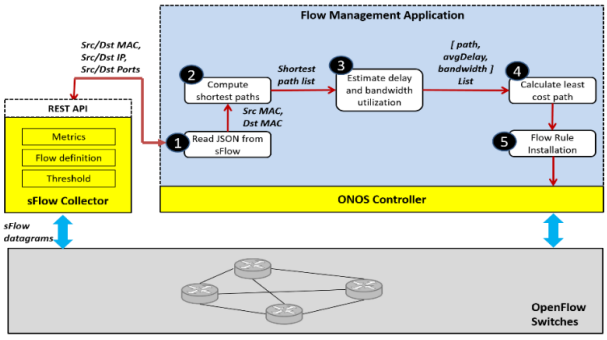

Figure 3. Architecture design

3.1. Detecting elephant flow

Elephant detection can be done in several ways: maintaining and polling per-flow statistics, packet sampling, and end-host based monitoring. Per-flow statistics as used in Hedera [9] has high accuracy at the cost of poor scalability in commodity switches. End-host based monitoring such as Mahout [14] overcomes the scalability issue, though it has not been widely adopted. In our current design, we use packet sampling, since it is widely used in practice with mature switch support such as sFlow [15]. Since the architecture of sFlow is based on collector and agents, sflow agents are needed to embed in network devices (eg. Open vSwitch) to be monitored. Two threshold values and two flow definitions are implemented in sFlow collector as shown in Figure 4. Every sFlow agents send the continuous streams of sFlow datagrams to collector. The sFlow datagram contains encapsulation and header information of sampled packet from individual flows. The collector computes the flow rate of sFlow datagrams in every second. If the flow rate exceeds the predefined threshold (10% of link capacity), the collector generates the elephant flow event and converts header information into metrics based on flow definition. In proposed architecture, there are two flow definitions, as we need to detect TCP and UDP elephant flows. The output metrics are represented by JSON format which consisting of attribute-value pairs. According to the flow definition as shown in Figure 4, the output information of elephant flow includes source and destination MAC addresses, IP addresses, TCP/UDP port numbers and the names associated with the ports of a link.

In order to access the elephant flow information from elephant flow rerouting application, the sFlow REST API: /events/json which is used to filter the threshold exceed events, is called periodically. In the proposed method, the REST API calling interval is set 1 second. The new elephant flow event can be defined in rerouting application by comparing the time stamp values of elephant flow events since sFlow REST API provides flow information with timestamp values. The output metrics are represented by JSON format which consisting of attribute-value pairs. According to the flow definition as shown in Figure 4, the output information of elephant flow includes source and destination MAC addresses, IP addresses, TCP/UDP port numbers and the names associated with the ports of a link and the value means the flow rate will be calculated in bytes.

```
SetFlow('tcpflow',
        {keys :'macsource,macdestination,
               ipsource,ipdestination,
               tcpsourceport,tcpdestinationport,
               link:inputifindex,link:outputinfindex',
         value:'bytes'});
```

```
SetFlow('udpflow',
        {keys :'macsource,macdestination,
               ipsource,ipdestination,
               udpsourceport,udpdestinationport,
               link:inputifindex,link:outputinfindex',
         value:'bytes'});
```

Figure 4. Flow definitions for TCP and UDP flows

3.2. Computing shortest paths

To find available shortest paths between source/destination pair where elephant flow happens, ONOS [12] controller provides DijkstraGraphSearch as a module. Its primary usage is in TopologyManager and flow management application invokes it associated with source/destination MAC addresses whenever elephant flow is detected.

3.3. Estimating end-to-end delay and bandwidth utilization

After computing available shortest paths between the source and destination hosts, the end-to-end path delay and bandwidth utilization are needed to measure for these paths.

3.3.1. Estimating end-to-end delay

End-to-end delay estimation comprises two main tasks: (1) probe packet creation and (2) delay estimation. In probe packet creation, each probe includes two parts as shown in Figure 6: header and payload. The header field includes faked source/destination (src/dst) MAC addresses and Ethernet type value (0x8888). Here, the faked src/dst MAC addresses for probes are generated by unique identifier (UID) value. Instead of traditional packet encapsulation, the time stamp (probe packet sent time) value is encapsulated in payload field of probe. Figure 5 show the delay measurement scenario.

$$MAC_{Probe} = UID \quad (1)$$

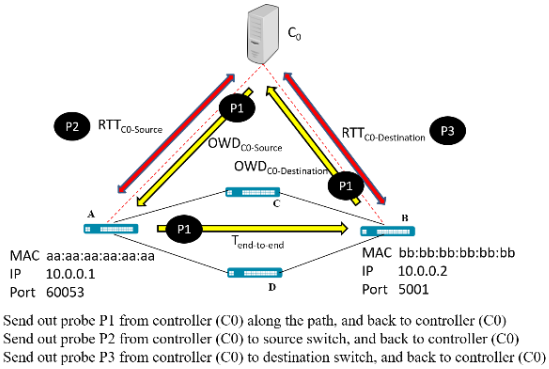

Figure 5. Delay measurement scenario

After probe packet creation, the first probe (P1) with faked MAC address as shown in (1) is sent from controller, through the path and back to the controller. After receiving first probe, the probe sent time is extracted from payload. The total time (T_{total}) can be computed by differentiating probe sent time (T_{sent}) from probe receive time ($T_{receive}$) as shown in (2).

$$T_{total} = T_{receive} - T_{sent} \quad (2)$$

Since T_{total} contains the one-way-delay from controller to source node ($OWD_{C0\text{-}Source}$), time taken from source node to destination ($T_{end\text{-}to\text{-}end}$) node, and destination node to controller ($OWD_{C0\text{-}Destination}$). Therefore $T_{end\text{-}to\text{-}end}$ can be derived as follows:

$$\begin{aligned} T_{total} &= OWD_{C0-Source} + T_{end-to-end} + OWD_{C0-Destination} \\ T_{end-to-end} &= T_{total} - OWD_{C0-Source} - OWD_{C0-Destination} \end{aligned} \quad (3)$$

Where the one-way-delays for $OWD_{C0\text{-}Source}$ and $OWD_{C0\text{-}Destination}$ are given as:

$$OWD_{C0-Source} = \frac{RTT_{C0-Source}}{2}, OWD_{C0-Destination} = \frac{RTT_{C0-Destination}}{2} \quad (4)$$

Where ($RTT_{C0\text{-}Source}$) is the round-trip-time between controller and source switch. The next probe packet (P2) is sent out from controller to source switch. As there is no matched rule in source switch for this probe, the source switch sends it back to controller. The $RTT_{C0\text{-}Source}$ can be retrieved from this probe P2. The other probe (P3) is also sent out to destination switch in similar way to measure the round-trip-time between controller and destination switch $RTT_{C0\text{-}Destination}$.

6 bytes	6 bytes	2 bytes	8 bytes
Source MAC	Destination MAC	Type	Payload

Figure 6. Probe packet frame

3.3.2. Estimating bandwidth utilization

To estimate the bandwidth utilization, we compute traffic load statistics of each links for specific path. Total sent bytes (sent bytes) and total receive bytes (receive bytes) of specific port which is associated with the device and port for link represent the total transmission bytes (or) the current bandwidth usage. The calculation of current total bytes (L) for source port and destination port of each link can be derived as follows:

$$L = src\_port\_statistics.bytesReceived + src\_port\ statistics.bytesSent + Dst\_port\_statistics.bytesReceived + dst\_port\_statistics.bytesSent \quad (5)$$

By applying (5) repeatedly until there is no next link in path, the total bytes of specific path can be calculated. Then the total bandwidth utilization b_k can be calculated as follows:

$$b_k = \sum_i^N \frac{L_i}{\mu_i} \quad (6)$$

Where L_i is total bytes of each link in path p_k, $L_i \in p_k$ and $p_k \in P$, and μ_i is the link capacity.

3.4. Calculating least cost path

In order to compute the best path, the proposed flow management method uses end-to-end path delay and bandwidth utilization. In this paper, we apply two ways to calculate least cost path based on flow type. If the elephant flow is TCP flow type, we choose the best path with the least minimum delay among available shortest paths. If the elephant flow is UDP type, the cost of each path $p_k \in P$ can be computed as follows:

$$p_k = d_k + b_k \quad (7)$$

Where d_k is average delay of path p_k, b_k is the total bandwidth utilization cost of $L_i \in p_k$ as shown in (6). The proposed flow management method selects the least cost path p_k for UDP elephant flow in order to avoid congested path.

3.5. Flow rule installation

After choosing the best path, the new flow entries are injected to respective devices through this path by using FlowRuleService which is provided from ONOS controller. The traffic selection fields of each flow entry are source MAC address, destination MAC address, protocol type and TCP/UDP ports. When the traffic flow rate does not exceed the threshold, the route decision and flow entries are made by using reactive forwarding method. As soon as the sFlow analyzer detects elephant flow, the route decision and new flow entries are made by proposed flow management application. The old entries which are injected from reactive forwarding will be removed automatically after 10 seconds in idle-timeout. In flow rule installation module, two main contributions are added in order to improve the performance of proposed method: (i) first, the flow rules placement in different tables, and (ii) second, avoiding unnecessary flow rule installation. Firstly, the flow rules that are generated from flow management application are mainly categorized into two: flow rules from delay measurement function and flow rules from rerouting function. As mentioned in previous section, delay measuring method is based on probing. The flow rules for probes are needed to install proactively to pass through the path. Figure 7 represents the scenario of flow rule placement (e.g. switch S1). In Figure 7, the red color box highlights the flow rules for probe packets and yellow color box highlights the flow rules for rerouting. If all of these flow rules are placed in one OpenFlow table “Table=0”, it makes the unnecessary flow matching time for rerouting. Therefore, in the proposed approach, the new flow rules for rerouting are defined as first priority rules because the elephant flow rerouting timely is studied as an important fact for network performance. Therefore, the flow rules for rerouting are placed in OpenFlow table “Table=0” and the flow rules for probes are placed in OpenFlow table “Table=1”. Secondly, the proposed approach considers the condition between finding least cost path and flow rule installation functions as shown in Figure 8. Sometimes, the elephant flow may be existed on optimal path. For this event, the new flow rule installation is unnecessary and even makes an increase in packet loss rate due to unnecessary flow rule modification. This condition is to check whether the current flow existing path is equal to the least cost path. If it is equal, the flow rerouting action does not need to take because the flow is already taken on the best path. If not, the elephant flow is needed to reroute to the least cost path.

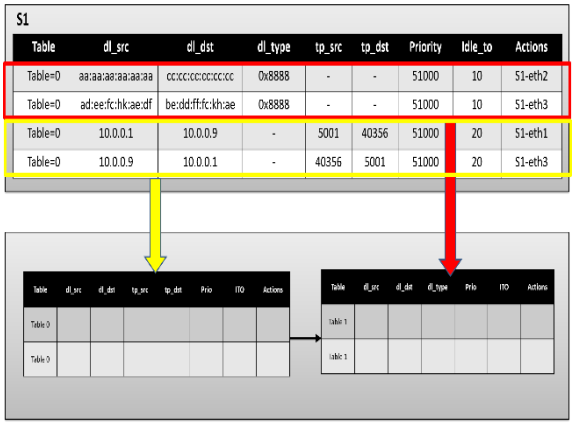

Figure 7. Flow rule placement scenario

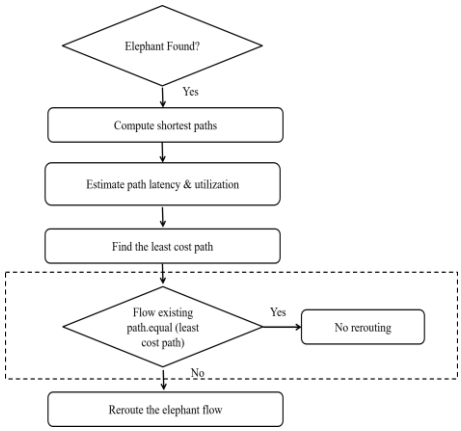

Figure 8. Avoiding unnecessary flow rule installation

4. PERFORMANCE EVALUATION

In this section, evaluation environment measurements and results are described. Tests are conducted by Mininet on Ubuntu host with the ONOS OpenFlow controller. In environment, two laptop PCs are used for evaluating the performance results. The first PC (i.e., Core i5-5200U CPU @ 2.20GHZ with RAM 8GB, Ubuntu 14.04 on Oracle VM VirtualBox) serves as ONOS controller. The second Laptop PC (i.e., Core i5-5200U CPU @ 2.20GHZ with RAM 8GB, Ubuntu 14.04) serves as mininet emulator and sFlow-rt collector. In this study, the simulation experiments were conducted by the Mininet emulator [16]. Mininet is used to model fat tree topology as shown in Figure 9. To evaluate the proposed method, k=4 fat-tree network with 20 switches and 16 hosts is built. The proposed elephant flow management application has been developed using the ONOS controller version 1.8 and OpenFlow version 1.3. Besides, Iperf [17] is used to generate both TCP and UDP traffic for simulations and to measure the network parameters:

- Throughput: successful data transfer rate (in Mbps), and
- Flow completion time (FCT): time difference between the time when the first packet of a flow leaves the sourceand the time when the last packet of the same flow arrives at the destination (in seconds) [18].

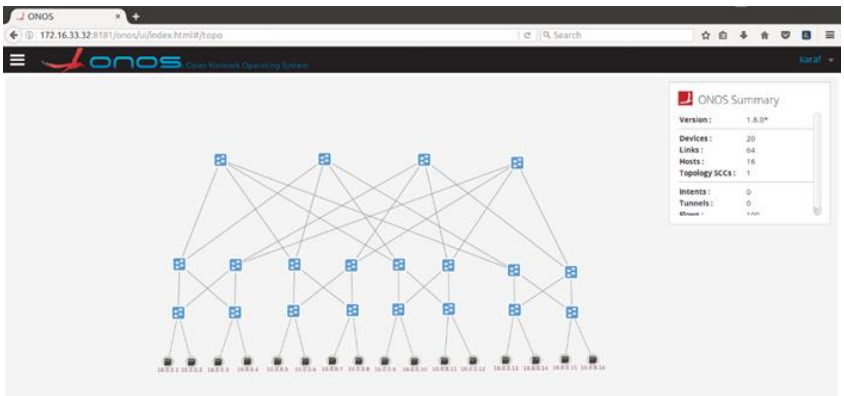

Figure 9. k=4, fat-tree topology

4.1. Simulation environment and measurement

In this measurement, we use traffic pattern random type, which is a host sends the amount of traffic to any other host in the network with uniform probability. For TCP traffic, every elephant flow size is 1 GB. For UDP traffic, we set the target file transfer rate 100 Mbps and run time duration 100 seconds. In topology setting as shown in Table 1, the edge link speed is set 300 Mbps, the aggregation link speed is set 100 Mbps and the core link speed is set 200 Mbps. The delay range is 20~95.7 ms and delay difference between paths is ranging from 15 ms to 80 ms. In sFlow setting as shown in Table 2, the elephant detection threshold is >=30 Mbps (10% of edge links). There are two sampling in sFlow, packet sampling and counter sampling. Packet

sampling consists of statistical data gathered from individual flows and counter sampling is polling of counters to gather interface data. In this paper, we use packet sampling rate: 1-in-300 packets and counter polling interval: 10 seconds.

Table 1. Topology setting

Parameters	Values
Link speed	200 Mbps :100 Mbps :300 Mbps
Link Delay	(20~95.7) ms
Delay Difference	(15~30) ms

Table 2. sFlow setting

Parameters	Values
Elephant Flow Detection Threshold	>= 30 Mbps
Sampling Rate	1-in-300 packets
Polling Inverval	5seconds

5. RESULT AND ANALYSIS

The results of the proposed scheme are compared with ECMP and reactive forwarding. In Figure 10, the elephant flow management method has been tested the average throughput improvement 19.18%~43.03% than ECMP and 33.62%~53.13% than reactive forwarding method. This is because the proposed elephant flow rerouting method reroutes the elephant flows to the least cost path based on types of traffic while ECMP chooses the route based on hashing of header values and the reactive forwarding method only uses the shortest paths for all traffic flows. When the number of elephant flow is 1, the proposed flow management method can schedule elephant flow in a way that provides the maximum throughput. Then the throughput of all algorithms decreases while the number of elephant flow increases. However, the proposed algorithm keeps the higher average throughput under more than one elephant flow. Figure 11 shows the average FCT per elephant flow in random traffic. In general, the FCT goes higher with the random number of elephant's flows in network increases. The proposed method has FCT reduction 16.83%~44.72% rather than ECMP and 28.84%~45.54% rather than the rerouting forwarding method. When the number of elephant flow reaches to 12, the average FCT of the proposed method is 132.52 seconds and the average FCT of ECMP and reactive forwarding is 165.35 and 249.93 seconds respectively. Figure 12 shows the average throughput per UDP elephant flow in random traffic. The number of elephant flows is generated from 1 to 12. The proposed method has throughput improvement 22.85%~45.7% rather than ECMP and 34.24%~52.50% rather than reactive forwarding method. According to the above results and verification, our proposed method can improve the network performance in terms of throughput and FCT for both TCP and UDP flow types.

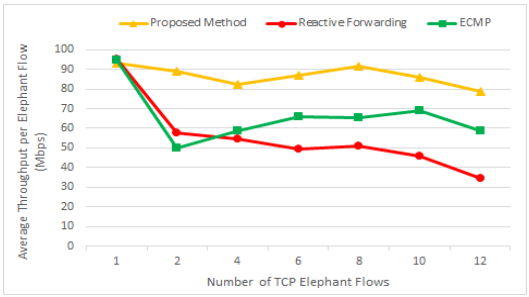

Figure 10. Average throughput per TCP elephant flow

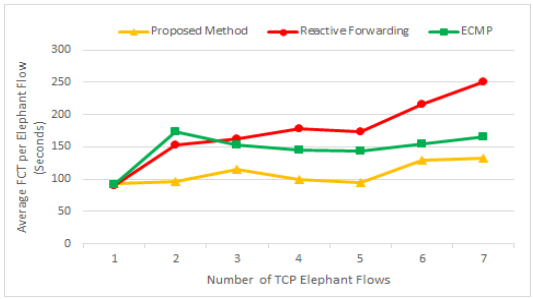

Figure 11. Average FCT per TCP elephant flow

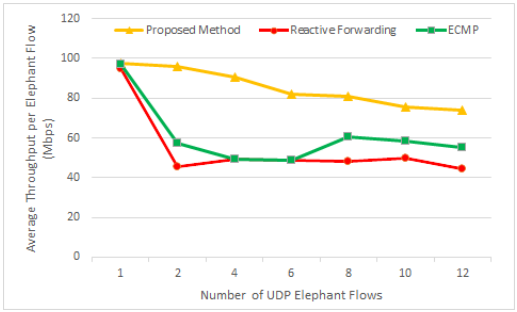

Figure 12. Average throughput per UDP elephant flow

6. CONCLUSION

In this paper, a new dynamic flow management scheme for fat-tree network is presented which differentiates elephant flows and re-scheduling to least cost path for both TCP and UDP traffic. Making use of SDN infrastructure and sFlow engine, our proposed approach can detect and re-reschedule TCP/UDP elephant flows using end-to-end path delay and bandwidth utilization, while mice flows are transmitted via reactive forwarding method. As the verification results, our proposed method improves average throughput and FCT for elephant flows in comparison with traditional ECMP and reactive forwarding. However, due to the requirement of Mininet emulator for modeling SDN fat-tree infrastructure, we encourage to research and evaluate on the realistic SDN testbed.

REFERENCES

[1] N. McKeown, T. Anderson, et al, "OpenFlow: Enabling innovation in campus networks," *SIGCOMM Computer Communication*, vol. 38, no. 2, pp. 69-74, 2008.
[2] E. Hopps, "Analysis of an equal-cost multi-path algorithm," 2000.
[3] M. Gholami, B. Akbari, "Congestion control in software defined data center networks through flow rerouting," *23rd Iranian Conference on Electrical Engineering (ICEE)*, Tehran, Iran. pp. 654-657, 2015.
[4] Y. Li, D. Pan, "Openflow based load balancing for fat-tree networks with multipath support," *12th IEEE International Conference on Communications (ICC13)*, Budapest, Hungary, pp. 1-5, 2013.
[5] K. Truong, S. Kukliski, W. Kujawa, M. Ulaski, "MSDN-TE: Multipath based traffic engineering for SDN," *Asian Conference on Intelligent Information and Database Systems*, Berlin, Heidelberg. pp. 630-639, 2016.
[6] J. Eric, D. Pan, J. Liu, L. Butler, "A simulation and emulation study of SDN-based multipath routing for fat-tree data center networks," *Proceedings of the Winter Simulation Conference (WSC)*, pp. 3072-3083, 2014.
[7] B. Conghui, X. Luo, T. Ye, Y. Jin, "On precision and scalability of elephant flow detection in data center with SDN," *Globecom Workshops (GC Wkshps)*, pp. 1227-1232, 2013.
[8] R. Kanagevlu, K. M. M. Aung, "SDN controlled local re-routing to reduce congestion in cloud data center," *2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*, pp. 80-88, 2015.
[9] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *NSDI*, vol. 10, no. 8, pp. 89-92, 2010.
[10] Z. H. Thiri, A. H. Maw, "Elephant flow detection and delay-aware flow rerouting in software-defined network," *9th International Conference on Information Technology and Electrical Engineering (ICITEE)*, Thailand, pp.1-6, 2017.
[11] A. Bianco, P. Giaccone, R. Mashayekhi, M. Ullio, V. Vercellone, "Scalability of ONOS reactive forwarding applications in ISP networks," *Computer Communications*, vol. 102, pp. 130-138, 2017.
[12] "Open Network Foundation (ONF)," [Online]. Available: https://onosproject.org, [accessed at: Aug 18, 2017].
[13] L. Cong, W. Yong-Hao, "Strategy of Data Manage Center Network Traffic Scheduling Based on SDN," *2015 International Conference on Intelligent Transportation, Big Data and Smart City (ICITBS)*, pp. 29-34, 2016.
[14] A. R. Curtis, W. Kim, P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-hostbased elephant detection, *Proceedings of IEEE INFOCOM*, pp. 1629-1637, 2011.
[15] "InMon Corp," [Online], Available: https://inmon.com/, [accessed at: Mar 30, 2016].
[16] "Mininet," [Online], Available: http://mininet.org, [accessed at: Dec 21, 2012].
[17] "Iperf," [Online], Available: https://iperf.fr/, [accessed at: Jul 7, 2014].
[18] Carpi, A. Engelmann, A. Jukan, "DiffFlow: Differentiating short and long flows for load balancing in data center networks, " *Global Communications Conference (GLOBECOM)*, pp. 1-6, 2016.

BIOGRAPHIES OF AUTHORS

Hnin Thiri Zaw She received master degree in computer technology from University of Computer Studies, Yangon (UCSY), in 2011. She is currently pursuing her PhD. from UCSY. Her research work is on traffic engineering and software-defined network.

Aung Htein Maw received the Master of Information Science (M.I.Sc.) degree from University of Computer Studies, Yangon (UCSY), in 2001, the master degree in Engineering Physics (Electronics) from Yangon Technological University (YTU), Myanmar, in 2002, and the Ph. D degree in Information Technology from UCSY, in 2009. He is one of the professors of Faculty of Computer Systems and Technologies, University of Information Technology. His research interests include Data Science and Advanced Network Systems. He has published technical papers in these areas, in the conference proceedings and jounals like IEEE and ACM Computing Survey. He has been cooperated at Research Collaborator in AssiaConnect Project and Subject Matter Expert in Asean Cyber University (ACU) project.