International Journal of Trend in Scientific Research and Development (IJTSRD)
Volume 3 Issue 5, August 2019 Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470

Traffic Engineering in Software-Defined Networking (SDN)

Aung Htein Maw

Faculty of Computer Systems and Technologies, University of Information Technology, Yangon, Myanmar

How to cite this paper: Aung Htein Maw "Traffic Engineering in Software-Defined Networking (SDN)" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-3 | Issue-5, August 2019, pp.1320-1323, **https://doi.org/10.31142/ijtsrd26618**

IJTSRD26618

ABSTRACT

The traditional single path routing can cause imbalanced link utilization and is not efficient for all traffic types such as long-lived large flows. Moreover, it can lead to low network throughput and high network latency. Traffic engineering (TE) is a key solution to solve these problems of single path. The main purpose of TE is to optimize the network resource utilization and improve network performance by measuring and controlling network traffic. One of the TE approach for large flows is multipath routing which distribute traffic load among available multiple paths. However, most of multipath solutions do not classify traffic flows (for example elephant or mice) and do not concern the existing delays of routes. Therefore, to be intelligent multipath routing based on traffic types, we proposed three main folds: (1) large flow detection approach by using sFlow analyzer in real time, (2) measuring end-to-end delays of available paths between source node and destination node where large flow occurred and (3) reroute the large flow to minimum round-trip time delay path in order to improve network performance. Through experimental results, our proposed method gains over 30-77% throughput improvement over reactive forwarding application which is implemented in ONOS controller.

KEYWORDS: *multipath; sdn; traffic engineering.*

INTRODUCTION

The existing traffic rerouting models implement different strategies in the multipath forwarding mechanism. The authors in [3] propose the routing algorithm splits the elephant flow into mice and distributes them across multiple paths based on source routing (label based forwarding) with round-robin manner. The limitation of their method is that it requires overhead bytes to implement policy in packet header increases linearly with path length. The difference is that their approach uses round-robin to split traffic load and our method is based on estimated delays of each path. Hedera [4] is a flow scheduling scheme to solve the hash collision problem of Equal Cost Multipathing (ECMP). It reduces large flow completion time (FCT) caused by network congestion and Utilizes the path diversity of data center network topologies. The difference is that Hedera uses per flow statistics for large flow detection, which has poor scalability and our method uses packet sampling. DiffFlow [5] differentiates short flow and long flow by using a packet sampling method. It applies ECMP to short flows and Random Packet Spraying (RPS) method to long flows. Their method causes packet reordering problem while transferring each packet to random egress ports because of different packet delivery time of available paths between source and destination. Our proposed method can avoid reordering problem since it is flow-based rerouting. Another work of traffic rerouting in [6] monitors congested path by collecting port statistics of each switch by using Open Flow protocol. When congestion occurs, it computes the least loaded path and reroutes some traffic flows from the congested path. Tiny Flow [6] presents large flow detection and random rerouting method. Once an elephant is identified, the edge switch adds a new rule to the flow table and collects byte count statistics periodically. When the byte count exceeds a limit, the switch picks an alternate egress port out of the equivalent cost paths randomly for elephant, reinstalls the new flow entry, and resets the byte count. The drawback of Tiny Flow is the elephant flow collision problem at the random egress ports of aggregate switches, resulting in poor bandwidth utilization. In this proposal, the proposed rerouting method is mainly based on large flow identification and end-to-end delay estimation. As soon as large flow is detected, the controller computes delays of parallel multiple paths between source and destination and reroutes the large flow to the path with the least delay path in order to improve throughput and minimize latency.

A. Proposed Method

As the elephant flows causes congestion and makes latency to other mice flows, differentiating elephant flow from mice flows is important process for improving network performance. Our proposed method uses sFlow-RT analyzer [8] which is explained in Section 1.1 to monitor large flow in data plane and extracts large flow information from sFlow in every one second.

The elephant flow information consists of source/destination IP addresses, source/destination MAC addresses, source/destination ports, links where large flow occurred.

As soon as large flow is detected, available shortest paths in terms of hop counts can be calculated between source and destination hosts. Then, end-to-end delay (d) can be measured for each path from set of available path lists and

stored in path-delay table. Path-delay table consists of path and set of delay values. End-to-end delay measurement method is described in Section 1.2. The average delay (Davg) of each path can be calculated from total delays from table and number of probe packets (Nprobe). According to these average delays, Minimum delay path and second minimum delay path is chosen for TCP traffic. Here, we use different paths (minimum and second minimum) for data forwarding and acknowledgement to improve QoS requirements.

Algorithm 1: Proposed Algorithm

Pre-Process: Monitor elephant flow by using sFlow-rt analyzer in every 1 sec;
Main Process:
Input: flow, elephant flow information from sFlow-rt
if elephant flow exists

get available shortest path-list in terms of hop counts between source and destination;

for each path in path-list

measure end-to-end delay (d) by sending probe packet;

calculate average delays of each path $D_{avg} = \frac{\sum d}{N_{probe}}$;

choose minimum delay path and second minimum delay path;

install new rules to calculated paths;

else

use reactive forwarding application for normal flows;

end

B. Experiment

In our test bed, we use ONOS controller (version 1.8) because of its performance, high level abstractions and API. ONOS is distributed system which is designed for scalability and high availability. It serves as network operating system with separation of control and data plane for service provider network. Our topology is created by using Mininet emulator (version 2.2.1) which can create virtual network and provide hundreds and even thousands of virtual hosts, and Open Flow (version 1.0). Two virtual machines is used for our test bed, sFlow analyzer and Mininet topology is running on one VM and ONOS controller is running on another. ONOS controller ran on a laptop powered by Core i5-5200U CPU @ 2.20GHZ with RAM 4GB. Mininet and sFlow ran on Laptop PC powered by Core i5-5200U CPU @ 2.20GHZ with RAM 4GB.

For sFlow analyzer, we configure sFlow agents on all open vswitches in topology according to Figure

1. In this figure, interface name (if name) of switch is used as sflow agent address and target address is sflow collector address. As we set link bandwidth 10Mbps, sampling rate is 1 in 10 packets and polling interval is 20 seconds. Then, we define flows in our large flow detection script as presented in Figure 2 which are used to match packets that share common attributes. In flow definition, a flow called pair that captures MAC addresses, IP addresses, TCP ports, interface indexes, and calculate bytes per second for each flow. After that, we also define threshold which is applied to metrics. We set threshold value 1MB in this script. When the rate value of a flow exceeds the threshold, it will notify as 'elephant' flow. Our ONOS application can access JSON output from sFlow analyzer by calling REST API: /events/json in every one second. This REST API list top active flows, and remove all duplicates for flows reported by sources. In our experiment, we use iperf tool as traffic generator for TCP flow.

```
def configSFlow(spine, leaf, collector, ifname):
    sflow = 'ovs-vsctl -- --id=@sflow create sflow
agent=%s target=%s sampling=10 polling=20 --' %
(ifname, collector)
    for s in range(1,spine+1):
      sflow += ' -- set bridge s%s sflow=@sflow' % s
    for ls in range(1,leaf+1):
      sflow += ' -- set bridge s%s sflow=@sflow' %
(spine+ls)
```

Figure1. sFlow agents configuration

```
setFlow('pair',{'keys':'macsource,macdestination,ipsource,
ipdestination,tcpsourceport,tcpdestinationport,link:inputifindex
,link:outputifindex','value':'bytes'});
setThreshold('elephant',{'metric':'pair','value':1000000/8,'byFlo
w':true,'timeout':1});
```

Figure2. Defining flow and threshold

C. Experimental Testbed and Results

In our testbed, we use ONOS controller (version 1.8) among other kinds of SDN controllers (eg.NOX, POX, Ryu, FloodLight) because of its performance, high levelabstractions and API. ONOS is distributed system which is designed for scalability and high availability. It serves as network operating system with separation of control and data plane for service provider network. Our topology is created by using Mininet emulator (version 2.2.1) which can create virtual network and provide hundreds and even thousands of virtual hosts, and OpenFlow (version 1.0). Two virtual machines is used for our testbed, sFlow analyzer and Mininet topology is running on one VM and ONOS controller is running on another.

Leaf-spine topology is based on two-tier topology which is mostly used in data center infrastructure. The leaf layer includes switches to provide connectivity of end devices. The spine layer provides connectivity of leaf switches. In our leaf-spine testbed topology in Figure 3, we use 8 switches and 8 hosts. Bandwidths of all links in this network topology are set 10 Mbps. ONOS controller ran on a laptop powered by Core i5-5200U CPU @ 2.20GHZ with RAM 4GB. Mininet and sFlow ran on Laptop PC powered by Core i5-5200U CPU @ 2.20GHZ with RAM 4GB.

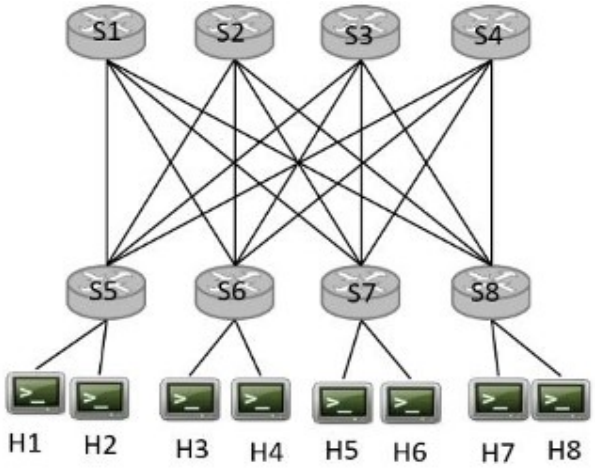

Figure3. Leaf-spine topology

TABLE I. Parameter Setttings

Parameter	Value
Leaf-Spine Topology	Leaf = 4, Spine = 4
Link Speed	10 Mbps
Large Flow Detection Threshold	>= 1Mbit/s
Sampling Rate	1-in-10 packets
Large Flow Metrics	Src Mac, Dst Mac Src IP, Dst IP, Src Port, Dst Port

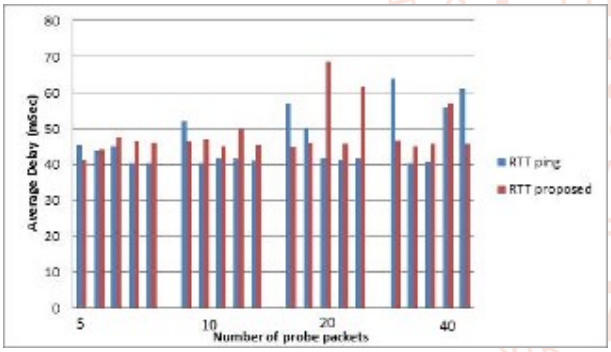

Figure4. Average delays in round-trip time based on number of probe packets

According to above Figure 4, we can study that ten and less than ten probe packets result the similar average values with ping results. In twenty and above, some significant difference points can be found. This is because of processing rate for delay calculation is directly proportional to the number of probe packets. According to Figure 5, 6, 7 and 8, our proposed method improve throughput 30%-77% based on delay difference values of each path.

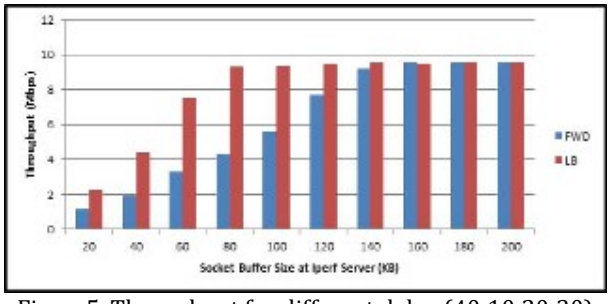

Figure5. Throughput for different delay (40:10:20:30)

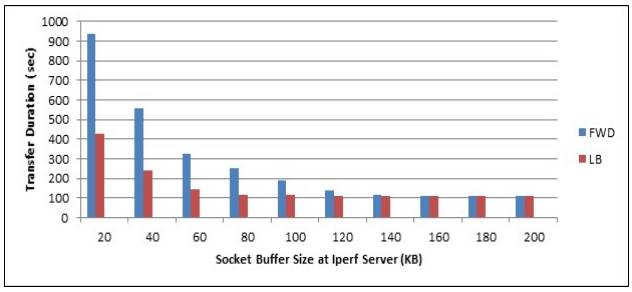

Figure6. Transfer duration for different Delay (40:10:20:30)

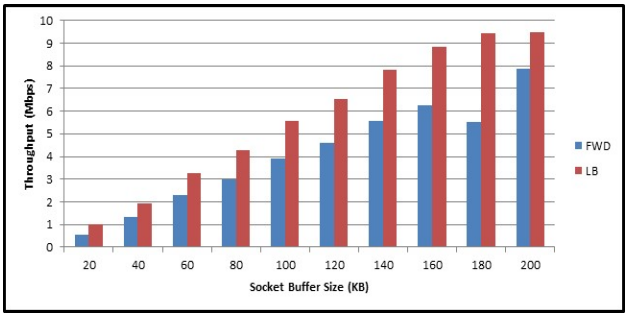

Figure7. Throughput for different delay (40:50:80:110)

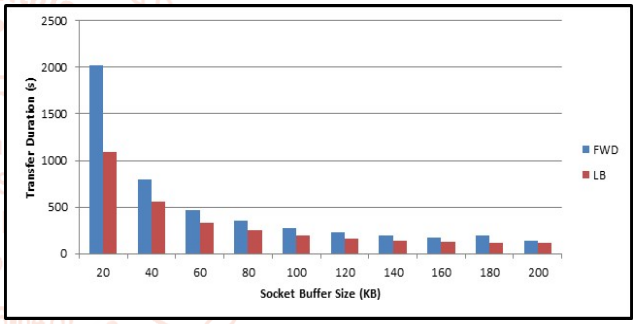

Figure8. Transfer duration for different delay (40:50:80:110)

When there is very low latency in network, our method and reactive forwarding method can be same result after 140KB buffer size. If large delay difference exists between available paths, our method can optimize the network performance (in terms of throughput and transfer duration) more than reactive forwarding method.

D. Conclusion

The traffic engineering method is presented in software-defined network by emulating layer 2 topology. The proposed method leverages an SDN infrastructure to support delay estimation and traffic rerouting. Unlike the traditional reactive forwarding method, our proposed method includes: differentiation elephant flows, estimation end-to-end delay of available paths between specified source and destination and reroute the elephant flows to the least delay path. The objective of our proposed method is to improve network performance by measuring and managing traffic dynamically. From the experimental results, we investigate that the proposed method optimizes the network throughput and transfer duration than the reactive forwarding method if there exists large delay difference among available paths.

References

[1] O. M. E. Committee, "Software-defined networking: The new norm for networks," ONF White Paper, vol. 2, pp. 2–6, 2012.

[2] 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, April 2017.

[3] S. Hegde, S. G. Koolagudi, S. Bhattacharya, "Scalable and fair forwarding of elephant and mice traffic in software defined networks," Computer Networks, vol. 92, pp. 330–340, December 2015.

[4] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," In NSDI, vol. 10, pp. 19–19, April 2010.

[5] F. Carpio, A. Engelmann, A. Jukan, "Diff Flow: Differentiating Short and Long Flows for Load Balancing in Data Center Networks," in Proc. IEEE GLOBECOM, vol. 10, pp. 1–6, April 2016.

[6] M. Gholami, B. Akbari, "Congestion control in software defined data center networks through flow rerouting," in Proc. IEEE ICEE, pp. 654–657, May 2015.

[7] H. Xu, B. Li, "Tiny Flow: Breaking elephants down into mice in data center networks," in Proc. IEEE LANMAN, pp. 1–6, 2014.

[8] http://www.onosproject.org/.

[9] Traffic monitoring using sFlow, http://www.sflow.org/.