Delay Controlled Elephant Flow Rerouting in Software Defined Network

Hnin Thiri Zaw, Aung Htein Maw
University of Computer Studies, Yangon, University of Information Technology
h.thirizawucsy@ucsy.edu.mm, ahmaw@uit.edu.mm

Abstract

As the network has limited resources, the traditional network with a single path routing mechanism will make the inefficient network resource utilization. Because the competitive utilization along the single path causes the performance degradation of traffic flows. Multipath routing is a good approach for inefficient resource utilization problem. It distributes the traffic load among parallel paths instead of single path. Moreover, the long flows which called elephant flows are needed to detect and handle in order to avoid traffic congestion. This paper proposes an effective solution by combining the elephant flow detection and rerouting based on end-to-end delays of available paths in Software Defined Network (SDN). The proposed method is implemented by using ONOS controller and Mininet emulator. The experimental results prove that 16.57%~78.03% throughput improvement and 26.18%~171.68% flow completion time (FCT) reduction for multiple elephant flows compared with the single path routing approach.

Keywords- Multipath, Elephant flow, Software defined network, SDN

1. Introduction

As the growth of deployment in online end-user applications such as VoIP, online gaming, video conferencing and so on, maintaining high throughput and low latency issues is important challenges for networks. To be efficient network resource utilization and fulfilled with quality of service (QoS) requirements, more and more traffic engineering (TE) applications are needed to innovate for better route decisions by measuring and controlling traffic flows. Reactive forwarding application of ONOS controller is a default single path application to make forwarding decisions whenever a new flow arrives at the switch [2]. The main process is that 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. The major drawback of the reactive forwarding method is that it makes route decisions without awareness traffic condition and QoS parameters (such as bandwidth, delay, packet loss and jitter), resulting in throughput degradation. The aim of the proposed method is to solve the drawback of the reactive forwarding method. This paper presents the overview of the proposed architecture to generate better route decision by considering traffic conditions (measuring elephant flows) and end-to-end delays of paths. The proposed method includes three main folds: (1) monitoring and detecting the elephant flow periodically (2) measuring end-to-end delays of available paths between source and destination where large flow happens and (3) rerouting the elephant flow to the least delay path. The proposed method implementation uses ONOS [10] controller and Mininet [8] emulated. sFlow [7] analyzer is used to monitor elephant flow by using packet sampling technology. The single path routing is used for mice flows by default. Once the elephant flow is detected from sFlow analyzer, end-to-end delays of all paths between source and destination nodes are measured and elephant flow is shifted to the least delay path.

The remainder of this paper is organized as follows. Section 2 presents related work overview. Section 3 gives the explanation about the overall architecture of the proposed method with three main tasks: large flow detection, end-to-end delay estimation and rerouting elephant flow. In Section 4, performance evaluation describes experiment scenario with throughput and packet loss. Section 5 presents the conclusion of this paper.

2. Related Work

The existing traffic rerouting models implement different strategies in the multipath forwarding mechanism. The authors in [3] propose the routing algorithm splits the elephant traffic 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] differentiate 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 OpenFlow protocol. When congestion occurs, it computes the least loaded path and reroutes some traffic flows from the congested path. TinyFlow [9] 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 TinyFlow is the elephant flow collision problem at the random egress ports at aggregate switches, resulting in poor bandwidth utilization.

In this paper, 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.

3. Overall architecture of proposed method

The overall architecture of the proposed method (see in Figure 1) is to reroute elephant flow based on average end-to-end delays of parallel paths between source and destination. The sFlow real time analyzer is used for monitoring and detecting elephant flows. In order to access the elephant flow information from our proposed method, the sFlow REST API is called in every 1 second. The new elephant flow event can be defined in the proposed rerouting method by comparing the timestamp values of elephant flow events since sFlow REST API provides flow information with time stamp values. According to the flow chart of Figur 1, as soon as the elephant flow is found, firstly it finds an available shortest path list in terms of hop counts between source and destination nodes. Then end-to-end delay of each path from path list is measured by sending out probe packets from the controller. From delay measurement module, the

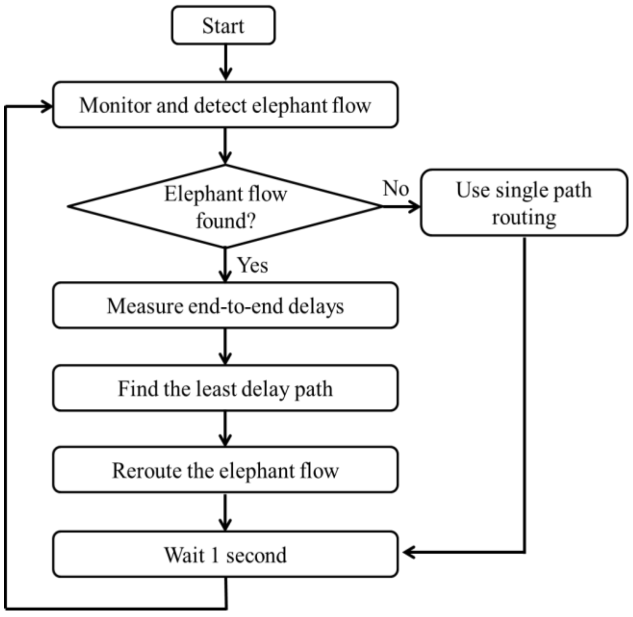
Start
Monitor and detect elephant flow
Elephant flow found?
No
Use single path routing
Yes
Measure end-to-end delays
Find the least delay path
Reroute the elephant flow
Wait 1 second

Figure 1. Flow chart of proposed method

average delays of each path can be calculated. After comparing the average end-to-end delays of available paths, the elephant flow is shifted to the least delay path to optimize throughput performance. For TCP traffic flow, the least and second least delay path are selected. In general, three main modules: monitoring and detecting elephant flows, measuring end-to-end delays and flow rerouting are developed for ONOS application.

3.1. Monitoring and detecting elephant flows

The proposed method uses sFlow analyzer for elephant flow monitoring and detection. This analyzer is a real time traffic analyzer for software-defined networking. It makes network traffic visibility in both physical and virtual devices (eg. Open vSwitch). sFlow uses packet sampling technology to analyze traffic statistics and it is based on the collector and agent architecture (see in Figure 2). The analyzer (or) collector receives a continuous stream of sFlow datagrams periodically from its agents which are embedded in network devices such as routers and switches. Therefore, sFlow solutions consist of two components (1) network equipments equipped with sFlow agents which monitor network traffic and generate sFlow data, and (2) sFlow application that receives and analyzes the sFlow data. Then the collector analyzes the utilization statistics of every traffic flow on all ports of devices. sFlow agents do very little processing. They simply package data into sFlow datagrams that are immediately sent to the sFlow collector. Once the utilization of traffic flow exceeds the specified threshold value, the collector converts them into metrics which are specified in keys of

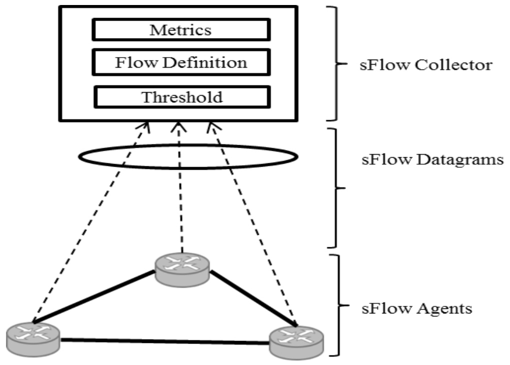

Figure 2. The Flow architecture

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

Figure 3. Flow definition of sFlow collector

flow definition. The output metrics are represented by JSON format which consisting of attribute-value pairs. According to the flow definition in Figure 3, the output information of elephant flow includes source and destination MAC addresses, IP addresses, TCP port numbers and names associated with the ports of a link. The elephant flow events of sFlow collector are queried by the proposed rerouting method via calling REST API: /events/json which is used to filter the threshold exceed events. Here the REST API calling interval is set from delay based rerouting application to sFlow analyzer is 1 second (less than 1 second affects the accuracy of delay estimation).

3.2. Measuring end-to-end delay

In delay measurement, three probe packets are needed to send for one path. Probe packet includes two parts (see in Figure 4): header and payload. The header field includes faked source/ destination MAC addresses and Ethernet type value (0x5577). The payload field includes a time stamp (sent time) value instead of traditional packet encapsulation. According to Figure 5, let's assume to find end-to-end delay between source S1 and destination S2. Firstly, the flow entries are needed to install to each device along the path proactively before sending the first probe.

The matching fields of flow entries are source/destination MAC addresses and Ethernet Type. The action output ports for flow entries are based on links of the path. For example, the action output port for S1 is 2 (source port of link) and the output port for S2 is default controller port c0 because S2 is the last device and there is no next link in the path. Table 1 and Table 2 show the flow entries for S1 and S2. Here, faked source MAC and destination MAC are assumed as 11:11:11:11:11:11 and 22:22:22:22:22:22 respectively.

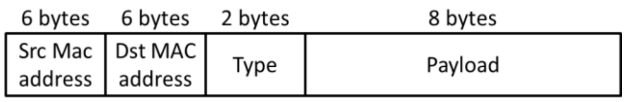

Figure 4. Frame format of probe

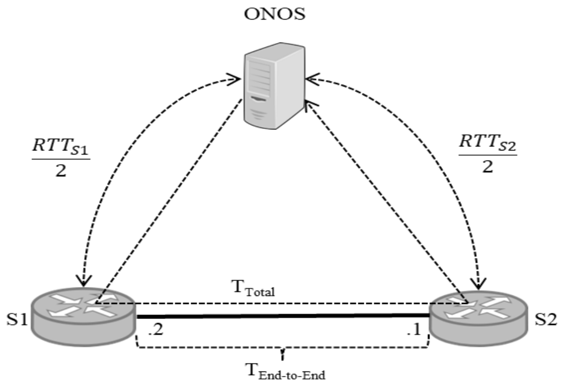

Figure 5. Delay measurement architecture

Table 1. Flow entry for S1

Source MAC	Destination MAC	EtherType	Action
11:11:11:11:11:11	22:22:22:22:22:22	0x5577	Port 2

Table 2. Flow entry for S2

Source MAC	Destination MAC	EtherType	Action
11:11:11:11:11:11	22:22:22:22:22:22	0x5577	C0

After flow entries installation, the first probe is sent through source switch to the destination switch along the path and back to the controller. When the first probe is received back, the controller records the packet arrival time $T_{arrival}$. Then the header information and payload are extracted to get packet sent time T_{sent}. From the first probe, the total delay time T_{total} (including $T_{arrival}$ and T_{sent}) can be learned. After the first probe, the next two probe packets are also generated from the controller to source switch S1 and destination switch S2 respectively like the first probe. From these two probes, the two round-trip-time between the controller and switches (RTT_{S1} and RTT_{S2}) can be found. It can be summarized as follow:

- 1st probe packet: measure T_{total} (T_{arrivl}-T_{sent}),

- 2nd probe packet: measure RTT_{S1}, and
- 3rd probe packet: measure $RTT_{S2.}$

Therefore, the equation for end-to-end delay $T_{end\text{-}to\text{-}end}$ cost can be derived following:

$$T_{end-to-end} = T_{total} - \frac{RTT_{S1}}{2} - \frac{RTT_{S2}}{2} \quad (1)$$

Since the delay estimation method is based on end-to-end delay, the half round-trip-time is assumed as the one-way delay in the calculation.

3.3. Rerouting flows

After delay estimation of available paths between source and destination where large flow occurs, the least delay path is selected and new flow entries are injected to respective devices through this path by using FlowRuleService which is provided from ONOS controller. For TCP traffic flow, the least delay path and second least delay path are chosen to optimize TCP throughput. The traffic selection fields of each flow entry address, destination MAC address, and TCP ports. When the traffic flow does not exceed the threshold, the route decision and flow entries are made by using the single path routing method. After utilization exceeds, the route decision and new flow entries are made by delay based elephant flow management. The old entries which are injected from single path mechanism will be removed automatically after 10 seconds, which is identified in idle-timeout. The idle-timeout is A flow table entry is removed if no packet matches the rule within a certain amount of time.

4. Performance evaluation

We evaluate the proposed delay aware rerouting method using emulated testbed as shown in Figure 6. Two laptop PCs are used for evaluating the performance results. The first PC (i.e., Core i5-5200U CPU @ 2.20GHZ with RAM 4GB, 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 4GB, Ubuntu 14.04) serves as mininet emulator and sFlow-rt collector.

4.1. Testbed Emulation

The network topology as shown in Figure 6 is created by using Mininet emulator (version 2.2.1) which can create the virtual network and provide hundreds and even thousands of virtual hosts. The topology is inspired by leaf-sine topology which is one of modern data center architectures. In leaf-spine topology, all leaf switches form access layer and meshed to range of spine switches. ONOS controller (version 1.8) is used among other kinds

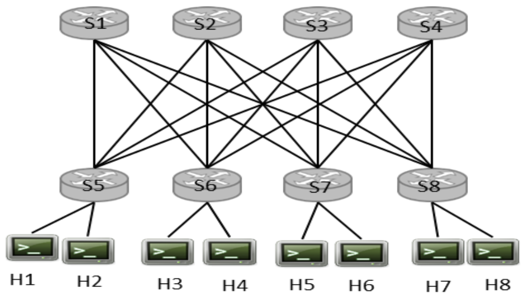

Figure 6. Emulated leaf-spine topology

of SDN controllers because of its performance, high-level abstractions and API. ONOS is distributed system which is designed for scalability and high availability. Iperf [12] tool is also used to generate TCP traffic and evaluate throughput and flow completion time (FCT). The FCT of a flow is the time difference between the time when the first packet of a flow leaves the source and the time when the last packet of the same flow arrives at the destination [5].

4.2. Parameter settings and evaluation results

Experimental scenarios are based on the two different parameter settings for the testbed topology (see in Figure 6). The proposed method is evaluated by generating four different numbers of TCP elephant flows to stress the network as shown in Table 5. In the first settings, the up-link speed is 10 Mbps and the down-link speed is 60 Mbps. The window size (or) socket buffer size at the receiver is 65535 bytes and sender is used default window size. The threshold value of elephant flow is greater than (or) equal 1 Mbps and packet sampling rate is 1 in 10 packets. In the second settings, the up-link speed is 2 Mbps and the down-link speed is 12 Mbps. The window size (or) socket buffer size at the receiver is the same as first parameter setting. The threshold value of elephant flow is greater than (or) equal 0.2 Mbps and packet sampling rate is 1 in 2 packets. In both parameter settings, the amount of data transfer for Iperf testing is 150 MB. There are four paths (P1, P2, P3, P4) between every source and destination in Figure 5. Different delays are used to test in both settings. Table 3 and Table 4 show the summarized parameter settings in details.

Table 3. Parameter setting I

Parameter	Value
Link speed	Up:10 Mbps, Down:60 Mbps
Threshold	1 Mbps
Sampling rate	1 in 10
Window size	65535 Bytes
Latency	P1, P2, P3, P4 :[20, 50, 80, 110] ms

Table 4. Parameter setting II

Parameter	Value
Link speed	Up:2 Mbps, Down:12 Mbps
Threshold	0.2 Mbps
Sampling rate	1 in 2
Window size	65535 Bytes
Latency	P1,P2,P3,P4:[2.78,20.2,24.6,6.8] ms

Table 5. Multiple elephant flow information

Number of flows	Source Host→Destination Host
1	H8→H1
2	H3→H1, H4→H2
4	H3→H1, H4→H2, H5→H1, H6→H2
6	H3→H1, H4→H2, H5→H1, H6→H2, H7→H5, H8→H6
8	H5→H1, H5→H3, H6→H2, H6→H4, H7→H1, H7→H3, H8→H3, H8→H4
10	H3→H1, H4→H2, H5→H1, H5→H3, H6→H2, H6→H4, H7→H1,H7→H3, H8→H3, H8→H4
12	H1→H3, H2→H4, H3→H1, H4→H2, H5→H1, H5→H3, H6→H2, H6→H4, H7→H1,H7→H3, H8→H3, H8→H4

The results of the proposed method are compared with the single path method. In Figure 7, the delay based rerouting method has the throughput improvement 16.57%~78.03%. This is because the proposed method reroutes the elephant flows to the least delay path while the single path method only uses the shortest paths for all traffic flows. In Figure 9, although the throughput improvement is 49.84%~79.03% for 2 and above 6 TCP elephant flows, the proposed method has the same result with the single path method for 1 and 4 TCP flows. The same results occur when the single path routing chooses the least delay path. In Figure 8 and 10, the results show that 26.18%~171.68% FCT reduction of proposed method. Therefore, it has been studied that the more elephant traffic flows in the network, the proposed scheme still outperforms evidently. According to throughput improvement, the proposed method is more outperformed when the link speed is 2 Mbps. The proposed rerouting scheme can reduce the performance degradation problem (in terms of throughput) of single path routing, i.e. poor bandwidth utilization without awareness of path condition and traffic types. The delay based traffic rerouting 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 singe path routing method, the proposed method includes: differentiation elephant flows, estimation end-to-end delay of available

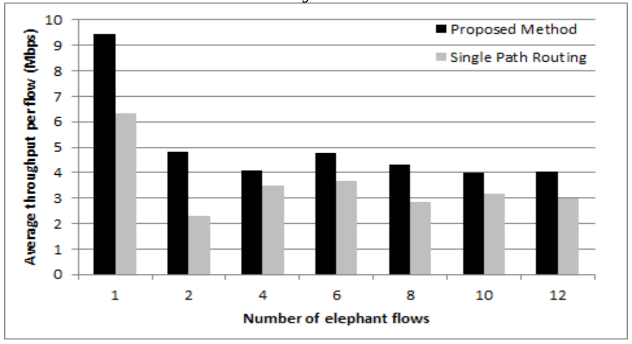

Figure 7. Throughput results for parameter setting I

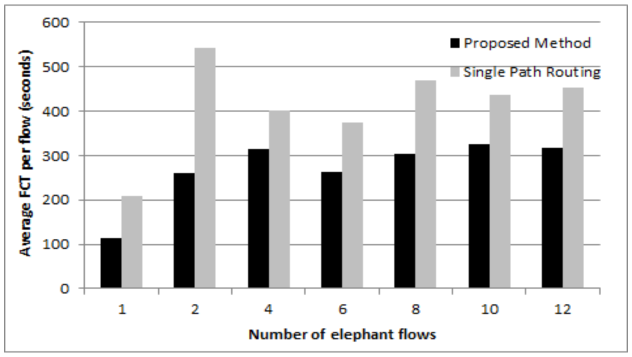

Figure 8. FCT results for parameter setting I

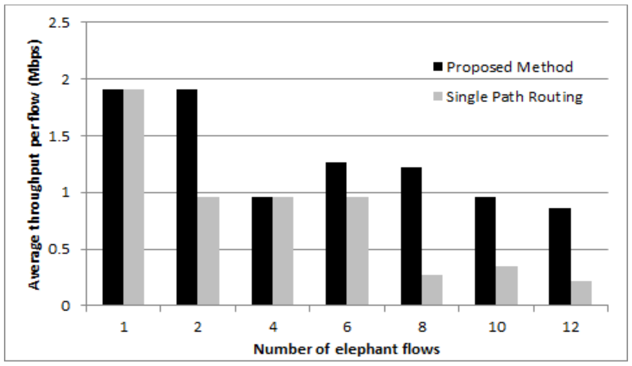

Figure 9. Throughput results for parameter setting II

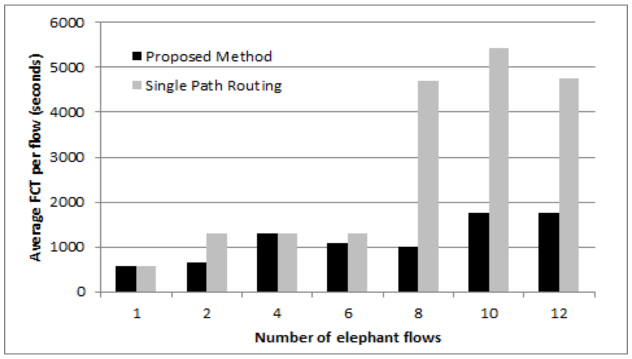

Figure 10. Throughput results for parameter setting II

paths between specified source and destination and reroute the elephant flows to the least delay path. The objective of proposed method is to improve network performance by measuring and managing traffic dynamically. The experimental results show throughput and FCT between the single path routing and delay based rerouting method as shown in Figure 7, 8, 9 and 10. The delay based rerouting scheme effectively uses least delay paths with the available bandwidth to avoid the congestion link. Hence, it has been obtained the better throughput and FCT after using the path delay based rerouting method.

5. Conclusion

The proposed delay based elephant flow rerouting method is implemented by using OpenFlow version 1.0 and it works on layer 2. Consideration for available bandwidth utilization is beyond the scope of paper and the future work will be considered it. The difference from traditional single path routing method is that the proposed method differentiates types of flows and reroute the elephant flow to least delay path in order to optimize throughput. According to experimental results, the proposed method improves the throughput results 16.57%~78.03% and 26.18%~171.68% FCT reduction as compared with the single path routing approach.

6. References

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

[2] A. Bianco, P. Giaccone, , R. Mashayekhi, M. Ullio, V. Vercellone, “Scalability of ONOS reactive forwarding applications in ISP networks”, Computer Communications, 2017, pp. 130--138.

[3] S. Hegde, S. G. Koolagudi, S. Bhattacharya, “Scalable and fair forwarding of elephant and mice traffic in software defined networks”, Computer Network, 2015, pp. 330--340.

[4] M. Al-Fares, S. Radhakrishnan , B. Raghavan, N. Huang, A. Vahdat, “Hedera: Dynamic Flow Scheduling for Data Center Networks”, In NSDI, 2010, pp. 19--19.

[5] F. Carpi, A. Engelmann, A. Jukan, “DiffFlow: Differentiating Short and Long Flows for Load Balancing in Data Center Networks”, In Global Communications Conference (GLOBECOM), 2016, pp. 1--6.

[6] M. Gholami, B. Akbari, “Congestion control in software defined data center networks through flow rerouting”, In Electrical Engineering (ICEE), 23rd Iranian Conference on, 2015, pp. 654--657.

[7] *Peter Phaal, March 2013 [Online]. Available from:* http://blog.sflow.com/2013/03/ecmp-load-balancing.html

[8] B. Lantz, B. Heller, N. McKeown, “ A network in a laptop: rapid prototyping for software-defined networks”, in SIGCOMM, 2010, pp. 19 .