# PDFR: Path Delay Based Flow Rerouting in Software-Defined Networks

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

*This paper presents path delay based flow rerouting (PDFR) algorithm for traffic engineering in software-defined network (SDN). The goal of PDFR is to reroute the elephant flow from ongoing path to least delay path in order to improve network performance. The evaluation results prove that PDFR has not only 21%~71.32% throughput improvement and 14.44%~49.25% flow completion time (FCT) reduction for elephant flows, but also 6%~40% FCT reduction in mice flows (short flows) compared with the single path method. Since PDFR works based on the delay metric of path, the future work will consider not only delay but also traffic loads of the path in order to achieve better performance results.*

***Keywords**: PDFR, Software-defined network (SDN), elephant flow, mice flow, rerouting.*

## 1. Introduction

As a well known data center network topology, the fat-tree topology contains various paths between 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. The hosts in the lowest layers are connected directly to one of the edge layer switches. The aggregate layer switches interconnects multiple edge layer switches together. All of the aggregate layer switches are connected to each other by core layer switches. Core layer switches are also responsible for connecting the data center to the Internet. For fat-tree topology, maximizing network throughput and minimizing transferring latency are two critical targets. So as to achieve them, there exist both equipment and programming approaches.

New technologies have changed because the nature of networking 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 also 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 decouples the data plane and control plane, allows network administrators to program the behavior of their networks with an external SDN controller, which has the ability to change the forwarding behavior of the network element directly [1]. The SDN is a network infrastructure with high adaptability and 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 exchanges control data of network traffic from the forwarding devices (such as switches and routers) to network operators. In an OpenFlow network, the OpenFlow controller handles high level routing decisions instead of the local switch, as is typically the case. The switch's CPU usage can then be utilized for faster packet transferring and other tasks. Administrators can work with the controllers to more effectively run the networks.

In this paper, we present a path delay based flow rerouting method for fat-tree topology. This method is adaptive to network traffic, and reroute elephant flows by examining current latencies or delays of each path between source and destination. The flow rerouting works as an application of the OpenFlow network operating system called ONOS [2], which runs on the controller host. The sFlow [9] analyzer detects the elephant flows in network. The dynamic flow rerouting application needs to access the elephant flow information from sFlow and reroutes the traffic flow to the lowest delay path among various paths. In the evaluation, PDFR has been tested by using the Mininet network emulator [3]. The results of PDFR method have been compared with single path forwarding. The results show that the PDFR works effectively for fat-tree networks. It provides better bandwidth utilization and average latencies for elephant flows under same circumstances. Moreover, it also reduces the average latency for mice flows.

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 PDFR 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 average latencies for elephant folios and mice flows. 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 limitation of our previous work [11] is that it can be adapted only in leaf-spine topology and is not recommended for fat-tree topology. In this paper, the topology challenge has been solved in PDFR since the results are evaluated in k=4 three layer fat-tree topology. Hedera [5] 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 [6] 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. PDFR can avoid reordering problem since it is flow-based rerouting. Another work of traffic rerouting in [7] 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 [8] 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 PDFR algorithm 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. PDFR Algorithm

In this section, path delay based flow rerouting algorithm PDFR is mentioned. Moreover, two main modules: elephant flow detection and end-to-end delay measuring are also presented.

### 3.1. Algorithm Description

The basic idea of PDFR approach 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 PDFR, the sFlow REST API is called periodically. The new elephant flow event can be defined in PDFR by comparing the timestamp values of elephant flow events since sFlow REST API provides flow information with time stamp values. According to Algorithm 1, as soon as the elephant flow is found, firstly it finds an available shortest path list in terms of hop counts between source S and destination D nodes. Then end-to-end delay (d) of each path from path list is measured by sending out probe packets from the controller. In our implementation, the number of estimation N is 10 for each path. When we get total delay $D_{total}$ for each path, the average delay $D_{avg}$ can be calculated. After comparing the average end-to-end delays of available paths, the elephant flow is shifted to the least delay path

**Algorithm 1** Path delay based flow rerouting algorithm
**Input:** mice flow, elephant flow
**Output:** least_delay_path
**Initialize:** $D_{total}$ = 0, least_delay_path = MAX_VALUE

```
if Elephant flow exists then
    Find available shortest path-list between S and D;
    for each path p in path-list do
        for each state i ∈ 1, 2, ..., N do
            Measure end-to-end delay d_i for p;
            D_total ← D_total + d_i ;
        end for
        Calculate average delay D_avg[p] = D_total / N ;
        least_delay_path = min(D_avg[p],least_delay_path);
    end for
    Install flow entries to least_delay_path;
else
    Use reactive forwarding;
end if
```

to optimize throughput and flow completion time. For TCP traffic flow, the least and second least delay path are selected. In general, three main modules: monitoring and detecting elephant flows, estimating end-to-end delays and flow entry installation are developed for ONOS application.

### 3.2. Elephant Flow Detection

To monitor and detect the elephant flows, the sFlow analyzer is used for PDFR algorithm. 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

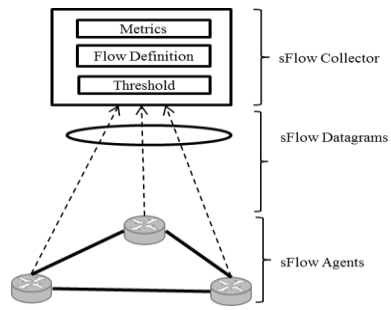


**Figure 1. sFlow architecture**

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

**Figure 2. Flow definition**

architecture in Figure 1. The analyzer (or) collector receives a continuous stream of sFlow datagrams periodically from its agents which are embedded in network devices. Then the collector analyzes the utilization statistics of traffic. Once the utilization of traffic flow exceeds the threshold, the collector converts them into metrics which are specified in keys of flow definition. The output metrics are represented by JSON format which consisting of attribute-value pairs. According to the flow definition in Figure 2, the output information of elephant flow includes source and destination MAC addresses, IP addresses, TCP port numbers and the names associated with the ports of a link. The elephant flow events of sFlow collector are queried by delay-aware rerouting method via calling REST API: /events/json which is used to filter the threshold exceed events. Here the REST API calling interval from path delay based flow rerouting application to sFlow analyzer is 1 second.

### 3.2. Path Delay Estimation

The PDFR application is developed in ONOS controller. In order to access the elephant flow metrics from PDFR, sFlow REST API: /events/json is called periodically. When the elephant flow information is found, the available shortest paths between source (S) and destination

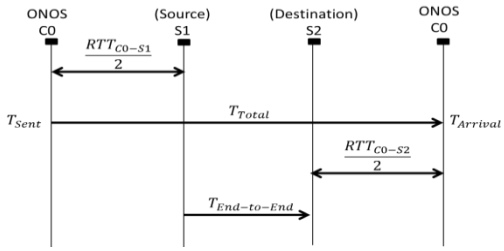


**Figure 3. Architecture of path delay estimation**

(D) are computed firstly. Then, the latency for each path in path list is estimated by sending out probe packets (see in Figure 3). In this figure, we assumed to find end-to-end delay between S1 and S2. Firstly, the first probe packet with the faked MAC address is sent out from ONOS controller through the desired path (S1-S2) and back to ONOS controller. $T_{Total}$ can be calculated from first probe by differencing packet sent time ($T_{sent}$) and packet arrival time ($T_{arrival}$). Then, the second probe and third probe are also sending out to source switch (S1) and destination switch (S2) respectively. From second and third probes, round-trip-time between C0 and S1 ($RTT_{C0\text{-}S1}$), C0 and S2 ($RTT_{C0\text{-}S2}$) can be found. The equation for end-to-end delay can be derived following:

$$T_{End-to-End} = T_{Total} - \frac{RTT_{C0-S1}}{2} - \frac{RTT_{C0-S2}}{2} \quad (1)$$

After estimating end-to-end delays of each path using equation (1), compare the average delays of available paths in order to get the least delay path. Finally, the new flow entries are installed to the least delay path and reroute the elephant flow to optimize throughput and flow completion time (FCT).

## 4. Evaluation

In this section, evaluation environment measurements and results are described. Tests are conducted by Mininet on Ubuntu host with the ONOS OpenFlow controller. Then network throughput and flow completion time (FCT) under random type of traffic pattern. Random traffic design is that a host sends packets to any

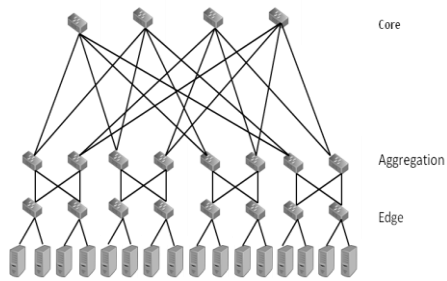


**Figure 4. k=4 three level fat-tree topology**

other host in the network with uniform probability. 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 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. Environment

Network Emulator*:* Mininet is used to model fat tree topology as shown in Figure 4. To evaluate PDFR algorithm, k=4 fat-tree network with 20 switches and 16 hosts is built.

### 4.2. Measurement

Iperf [10] is used to generate TCP traffic in the network. The random Iperf clients transmit data flows to random Iperf servers. The measurement has been tested for both elephant flows and mice flows. In measurement, elephant flow size 1 GB and mice flow size is 700 kB. The default window size is 1 MB for both tests. The other parameter setting for both tests is shown in Table 1. Then the results are measured by using two parameters:

- Throughput: successful data transfer rate, and
- Flow completion time (FCT): 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 [6].

### 4.3. Results

The results of PDFR are compared with the single path method. In Figure 5, the path delay based flow rerouting method has been tested the throughput improvement 21%~71.32%. This is because PDFR reroutes the elephant flows to the least delay path while the single path method only uses the shortest paths for all traffic flows. When the elephant flow is 1, the PDFR algorithm can allocate flow in a way that provides the maximum throughput they need.

**Table 1. Parameter setting for measurement**

| Parameter | Value |
|---|---|
| Link Speed | (100:100:100) Mbps |
| Threshold | 10 Mbps |
| Sampling rate | 1 in 100 |
| Polling interval | 5 seconds |
| Window Size | 1 MB |

Then the throughputs of both algorithms decrease while the number of elephant flow increases. But the PDFR algorithm keeps the higher average throughput under one more than elephant flows.

Figure 6 shows the average FCT for elephant flows in random traffic. In general the FCT goes higher with the random number of elephant flows in network increases. The PDFR algorithm has FCT reduction 14.44%~49.25% rather than single path method. When the number of elephant flow reaches to 20, the average FCT of PDFR is 269 seconds and the average FCT of single path method is 370 seconds. This fact implies that when hosts in the network send flows with close to their maximum capability, the traffic load becomes so high that the latency increases significantly. It has also investigated that the increase of FCT changes more rapidly than the decrease of average throughput.

Figure 7 shows the average FCT for mice flows in random traffic. The number of mice flows is generated from 50 to 300. The PDFR algorithm has FCT reduction 6%~40% rather than single path method. When the number of

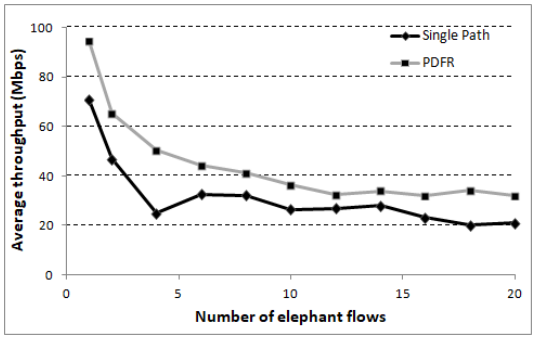


**Figure 5. Average throughput for elephant flows**

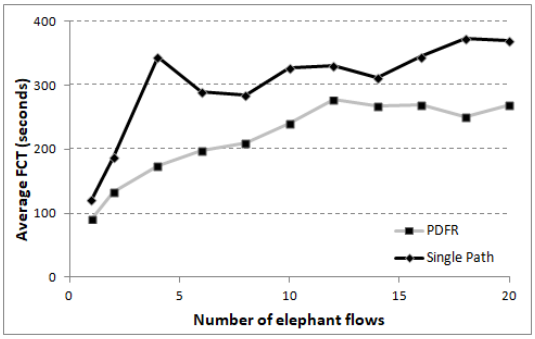


**Figure 6. Average FCT for elephant flows**

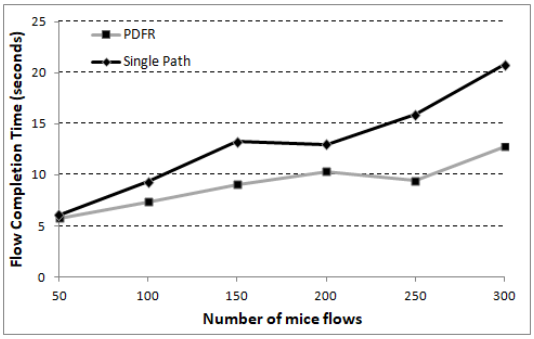


**Figure 7. FCT results for mice flows**

elephant flow reaches to 50 flows, the average FCT of PDFR algorithm has 6% reduction and when it reaches to 300 flows, FCT reduction becomes 40% rather than single path method.

## 5. Conclusion

In this paper, path delay based flow rerouting PDFR algorithm is presented to efficiently manage flows for fat-tree networks, which have various alternative paths among a single pair of every host. The algorithm makes route decisions based on real-time flow size detected via sFlow analyzer. The PDFR algorithm is implemented as a module of the ONOS controller, with three main functions: monitoring flow size, estimating path delays and rerouting flows. The results show that the PDFR algorithm is outperformed rather than the single path method in maintaining better throughput and avoiding FCT increasing for both elephant and mice flows under the random network traffic. As an ongoing research, both of path delay and the traffic load parameters will be considered to achieve better performance results.

## References

[1] N. McKeown, et al., "OpenFlow: Enabling innovation in campus networks," in *Proc. ACM Special Interest Group on Data Communication*, vol. 38, New York, NY, USA, April 2008, pp. 69-74.

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

[3] B. Lantz, B. Heller, and N. McKeown. A Network in a Laptop: Rapid Prototyping for Software-Definded Networks. ACM SIGCOMM, 2010.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] H. Xu, B. Li, "TinyFlow: Breaking elephants down into mice in data center networks", in Proc. IEEE LANMAN, 2014, pp. 1-6 .

[9] Peter Phaal, March 2013 [Online]. Available from: http://blog.sflow.com/ 2013/ 03/ ecmp-loadbalancing.html.

[10] http://software.es.net/iperf/.

[11] H.T. Zaw, A.H. Maw, "Delay controlled elephant flow rerouting in software-defined network", in ICAIT 2018, pp. 52-57.