Proceedings of
2019 International Conference
on Advanced Information Technologies

ICAIT 2019

Yangon, Myanmar
6th & 7th November, 2019

Organized by

University of Information Technology
Ministry of Education, Myanmar

2019 International Conference on Advanced Information Technologies (ICAIT 2019) November, 2019

Contents

Flow Path Computing in Software Defined Networking

Ohmmar Min Mon, Myat Thida Mon
University of Information Technology, Yangon, Myanmar
ommm@uit.edu.mm, myattmon@uit.edu.mm

Abstract

With the rapid growth of current network, the demand of resources is growing significantly. Insufficient bandwidth results unstable throughput in a network. Software-defined network (SDN) has been proposed to provide optimal routing decision in the presence of congestion. Equal Cost Multi-path (ECMP) routing cannot guarantee optimal resource utilization. ECMP causes the long flow collision in network because it does not consider the network parameter such as bandwidth. In this paper, the Flow Path Computing Algorithm (FPCA) is proposed to minimize the network congestion by rerouting the flows in SDN. This algorithm mainly focuses on rerouting the traffic flows over the alternative path when the network congestion is detected. If the flow demand is exceeded 10% of link bandwidth, the algorithm computes the light loaded path based on the port statistics and it shifts from the congested path to light loaded path. Simulation results are presented to show the effectiveness of the FPCA algorithm over ECMP.

Key Words- Software Defined Network, Equal Cost Multi-path, OpenFlow

1. Introduction

There are many applications such as email, video conferencing with most of the traffic on the Internet. Enormous traffic forces congestion to deliver the data in the network links. Traffic engineering (TE) offers the enormous solution to improve the network performance. Control plane and data plane are combined in dedicated device in traditional networking. The control plane in traditional network implements the network protocols and accomplishes the signaling between the network devices. The data plane forwards the packets from one port to another port based on the routing table. When congestion occurs, the nodes are difficult to update the information of other nodes in the network.

Software-defined networking (SDN) accomplishes the existing function by splitting the control plane and forwarding plane in the network. It enables the network to be programmable and controllable. OpenFlow protocol use for the interactions between the controller and the switches and it controls the network traffic by creating the network to innovate the network application. It provides admission to data plane and programs the software to forward the data between the OpenFlow switches. The controller makes a decision to follow the rules to the switches and determines the appropriate path which the packets will take over the network.

In this paper, we compare our FPCA with static ECMP hashing. Although existing network routing tends to the dynamic way, routing in the network using OpenFlow depends on the static path information. The congestion in the network because of large flows disturbs the network state and utilization of links. ECMP is a routing method to protect link failure without loss of traffic. ECMP allows the traffic to divide among equal cost paths. There is a limiting factor that maximum ECMP paths are 8 and 16. It finds the appropriate path in the network with this limiting factor in the large networks. It is used for short flows and it reduces the flow completion time. So it becomes a collision by assigning long lived flow to the same ECMP path. It is also based on the hash function by splitting equal size partitions and steers packets based on the information along the path. Although ECMP provides a good performance by providing static load balancing, it can collide over the same path in the network. The main idea of this paper is efficiently reroute the congested flows into the new alternative path based on the collected port statistics from the switches.

The rest of the paper is designed as follows:

The related work is provided in section two. Section three provides the background of traffic engineering in SDN. Section four motivates the flow path computing scheme of this paper with the background of fat-tree network topology and proposes the solution. The experimental setup and performance evaluation are explained in Section five. The final part of this section concludes this paper and illustrates the ideas of future work.

2. Related Work

In this section, we introduces about some related works to this paper.

The work in [1] explores the effect of statistics collection occurrence on the network load and the correctness of the results. This system uses the

information in terms of actual link usage to calculate available bandwidth on network link in real time environment. This system calculates the utilization of the link and link capacities through statistics in the network.

Authors in [2] considered a method to control the congestion by monitoring monitors the port statistics based on the OpenFlow switches and reroutes some of the flows in the links which are congested. This method does not consider the QoS constraints of other flows.

S. Song, et al. [3] proposed the flow congestion avoidance algorithm to predict the congestion by the controller, the utilization information is used. Link utilization is calculated in SDN controller by dividing the maximum bit rate and recalculated rerouting algorithm is applied to switches which would be configured by using OpenFlow configuration protocol.

To allocate optimal routing paths for multiple tasks at the same time, BCMPO problem [4] is proposed to allocate bandwidth resources with Genetic Algorithm. The main challenge is to redirect traffic flows to get optimal path when there is congestion problem.

An efficient method based on SDN [5] proposed for reduction of congestion in data center networks using linear programming, and a greedy bin packing heuristic for rerouting of selected flows in the switches with the congested links. This paper uses the Fat-Tree topology to allocate traffic load. This paper reduces power consumptions and link congestions but does not consider rerouting the traffic.

Authors of [6] explore the use of SDN switches by reducing the link utilization to handle a link failure. To avoid congestion in case of link failures, the controller reroutes traffic through non damaged paths by using pre-set tunnels when a node occurs failure.

Tajiki et al. [7] proposed an efficient congestion avoidance method based on the flow requirements. This paper is compared with ECMP to prevent congestion and the waste of resources by allocating resources.

The authors [8] proposed Mahout to monitor and identify long flows in TCP Socket buffer. Because of the changing of network conditions, congestion will occur while Mahout controller selects the path. It cannot get the optimal path over TCP connection using the traditional ECMP.

Network utilization-aware load balancing scheme CONGA [9] recommended the switches to gather capacities for flowlets by splitting the flows and to estimate of the congestion condition of the paths from the switches. CONGA has a necessity to alter the stack of the end-host for selecting the optimal path. Therefore, it increases the operation costs.

3. Overview of Traffic Engineering in SDN

Traffic engineering (TE) technology is the evolution of technology and accomplishment of SDNs. There are many TE architectures of SDN networks such as ATM-TE, IP-based TE, MPLS-based TE. TE techniques in SDN have been presented to increase the throughput of larger networks, to optimize network performance and traffic delivery compared to the traditional approach such as MPLS-based TE. SDN provides centralized control of network state and supports flexible software programming. TE in SDN techniques should be used to place the traffic for efficient network utilization to allocate network resources efficiently for network congestion avoidance. Figure 1 describes a typical SDN architecture. Traffic engineering techniques are characterized with flow management, fault tolerance, and traffic analysis.

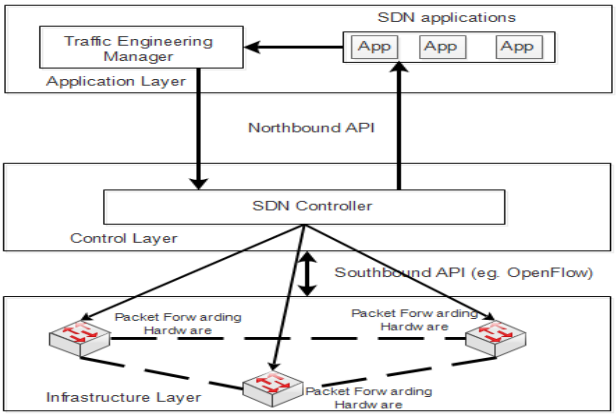


Figure 1. Traffic Engineering using SDN

TE mechanisms should have an ability to detect and recover the failure from congestion and reroute them over information about the network. Many researchers have been suggested to overcome this drawback. When the SDN controller identify the failed link, it computes new routes and sends the suitable flow entries to the switches immediately. The information from the controller is updated to their forwarding tables by the switches. The controller queries statistical information from the switches and if it detects the congestion in the network, it estimates the flow demands and it reroutes from the congested path to light loaded path accordingly. In traditional network, TE with ECMP depends on computing the route and on splitting traffic among the paths in the network. TE in large IP networks relies on configuring static link weights via the ECMP mechanism. We consider the aspects of TE with ECMP for numerous generated traffic patterns.

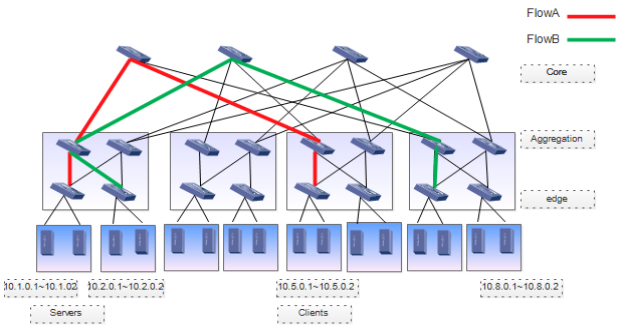


Figure 2. ECMP Collision Problem in Fat-tree Network topology

A routing technique applying a hash function ECMP is the widely used technique to improve multipath routing networks. In ECMP, a hash value is used for direct hashing to allocate a flow. It can cause next-hop several times in the network and it splits the different colliding flows to route over different paths. Therefore, it may be unutilized link bandwidth to route the different flows over different paths.

If multiple large flows are assigned to a shared link, these flows are challenging for the data rate although there are many paths with free data rate. Traditionally, ECMP uses paths having equal costs. ECMP is used in large network when the flow between source and destination is routed over the shortest path. The switches forward of flows by using ECMP, large flows can strike on their hash and it can cause performance degradation in a large size network. Figure 2 shows the ECMP routing. All the packets from flow A and flow B can end up over the same port Aggregation switch 0 due to hash collision among large flows and it creates network bottleneck. In this example, flow A could have been forwarded to Core 1.

4. Flow Path Computing Scheme

The main idea of traffic engineering is to minimize congestion and reroute flows inside a network. We describe the SDN-based flow path computing method between source and destination in section 3. In this section, the route selection of the flows is done by using statistics from the controller to the switches. Congested links are identified as over-utilized links that cause network congestion and packet loss. The flow path computing architecture of Figure 3 mainly focuses on the traffic through the congested links that are rerouted to the light-loaded alternate path.

The FPCA enhances the overall routing performance for the traffic demands to reroute flows for traffic engineering purposes and to prevent network congestion. The controller dynamically estimates the natural demand and calculates new alternative path. This natural demand is less than the threshold value, 10% of link capacity. The controller detects large flows based on flow statistics and links statistics from the network devices. Once the controller recognizes the flows to detect the large flows according to the threshold value and links information from the switches, it accomplishes natural demand estimation and adjusts the flow to reroute to new light loaded paths according their flow demands. The controller checks the flow demand for the paths that satisfy the flow by Equation (1). We compare our FPCA with static ECMP hashing.

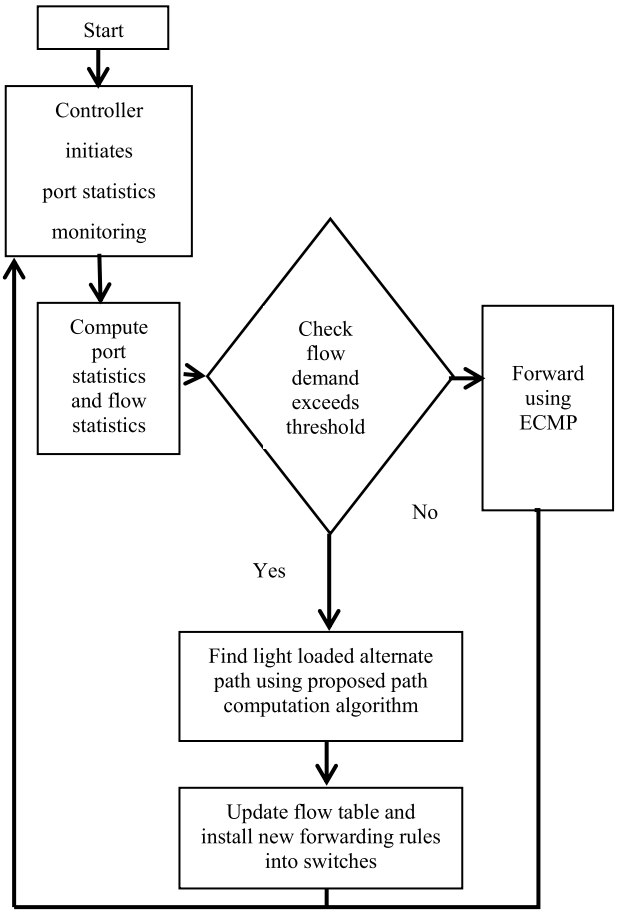


Figure 3. Flow Path Computing Architecture

Our simulator models the fat-tree network topology as a network graph G(V, E) as shown in Figure 5 with directed edges. In this case, we define the Flow Demand in Equation. (1):

$$FlowDemand = \frac{FlowSpeed * 8.0}{MAX-CAPACITY * 1000} \quad \text{------(1)}$$

And we define the port speed as Equation. (2) and (3):

$$NowPortStats = tx\_bytes + rx\_bytes \quad \text{------(2)}$$

$$PortSpeed = \frac{NowPortStats - PrePortStats}{MonitoringPeriod} \quad \text{-------(3)}$$

Here, we have to get the free bandwidth in Equation (4):

$$GetFreebw = \max(LinkCapacity - PortSpeed * \frac{8.0}{1000}) \quad \text{-------(4)}$$

The algorithm calculates existing flow's demand using flow statistics. If the flow demand is exceeded 10% of link bandwidth, it computes the light loaded path based on port statistics and the existing flow is shifted from the congested path to light loaded path. We show the pseudo code for flow path computing algorithm (FPCA) in Figure 4. Then f_D represents flow demand, bw_{max} defines the maximum bandwidth, bw_f represents feasible bandwidth and C_h defines the congested path.

```
Input:  link bandwidth, Topology: G(V,E)
Output: Number of selected path
p= selected path;
Initialize: L.capacity =bwmax;
Begin
Loop
For each psrc→dst in link do
        if bwf + fD ≤ L.capacity then
        p ← bwf + fD
        return p
        else
        p= Ch(hash),
        return p = psrc→dst(Ch)
end Loop
end
```

Figure 4. Flow Path computing algorithm (FPCA)

5. Experimental Setup

For testing flow path computing system, we created the topology in Mininet. The VM image has a 64-bits Ubuntu 16.04 installed as the guest OS. The system was run with Core TM (i5) 1.6 GHz CPU and 4 GB of RAM. These are the minimum requirements to run the environment. There are various topologies such as leaf-spine, fat-tree, Jellyfish and so on to interconnect various nodes in computer networks. The idea of Fat-Tree is to provide effective communication. Consequently, our proposed approach use Fat-Tree topology to deliver multiple paths as an environment.

We use the fat-tree network topology of (k=4) with 20 (4-port) switches and 16 hosts. It consists of K-port switches that contain k pods. Each pod is connected to two layers of k/2 switches. Each aggregation consists of (k/2)2 core switches with one port attached to each k pod. Each edge switch is connected to (k/2) hosts and the remaining k/2 port of edge switches is connected to a (k/2) aggregation switch. Firstly, we test the experiment with FPCA using iperf. Subsequently, the link between switches set the maximum capacity of 100 Mbps, the maximum throughput for a flow can send is nearly 100 Mbps. The throughput for different flow in both cases is measured.

The evaluation was performed on the Mininet testbed and the OpenFlow controller configured to communicate with the data plane. In our testing, all data flows are built with iperf tool. The received throughput is observed from iperf's statistics. The script in this system is based on Python to generate flows.

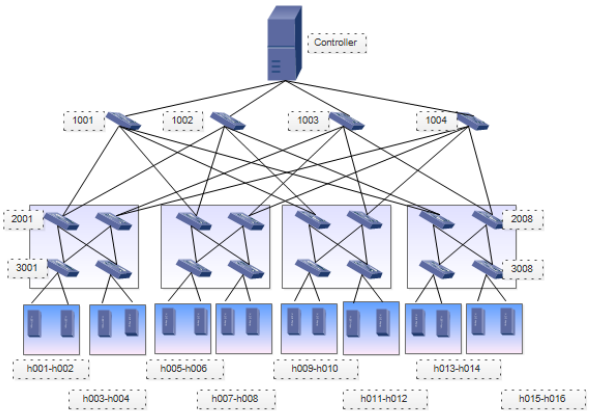
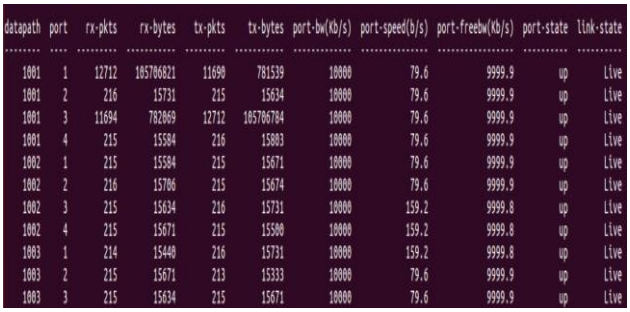


Figure 5. Experimental testbed with Fat-tree Network topology

Table 1. Congestion Method

Congestion Threshold	Condition
Flow Demand> 10% of link capacity	Congested
Flow Demand =< 10% of link capacity	Normal

In this system, the algorithm calculates existing flow's demand using flow statistics as shown in Figure 6. If the flow demand is exceeded 10% of link bandwidth, it computes the light loaded path based on port statistics and finally, the existing flow is shifted from the congested path to light loaded path as shown in Table 1.



datapath	port	rx-pkts	rx-bytes	tx-pkts	tx-bytes	port-bw(Kb/s)	port-speed(b/s)	port-freebw(Kb/s)	port-state	link-state
1001	1	12712	185786821	11690	781539	10000	79.6	9999.9	up	Live
1001	2	216	15731	215	15634	10000	79.6	9999.9	up	Live
1001	3	11694	782869	12712	185786784	10000	79.6	9999.9	up	Live
1001	4	215	15584	216	15883	10000	79.6	9999.9	up	Live
1002	1	215	15584	215	15671	10000	79.6	9999.9	up	Live
1002	2	216	15786	215	15674	10000	79.6	9999.9	up	Live
1002	3	215	15634	216	15731	10000	159.2	9999.8	up	Live
1002	4	215	15671	215	15580	10000	159.2	9999.8	up	Live
1003	1	214	15440	216	15731	10000	159.2	9999.8	up	Live
1003	2	215	15671	213	15333	10000	79.6	9999.9	up	Live
1003	3	215	15634	215	15671	10000	79.6	9999.9	up	Live

Figure 6. Port statistics from the controller

5.1. Performance Evaluation

Experimental results are based on the different parameters for the network topology. This paper shows the performance of our solution in terms of flow completion time and throughput. The FPCA is performed by generating different number of flows as shown in Figure 7, 8, 9, 10 and Table 2.

Table 2. Number of Flows

No. of Flows	Source-Destination
1	H9->H1, H10->H2
2	H9 and H10->H2
4	H9->H1, H10->H1, H11->H2, H12->H2

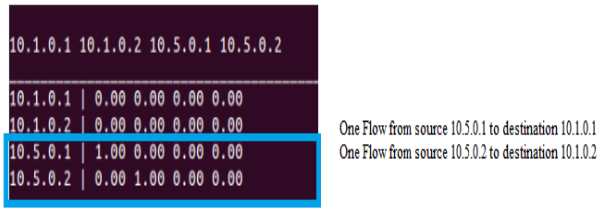


Figure 7. Demand estimation with one flow

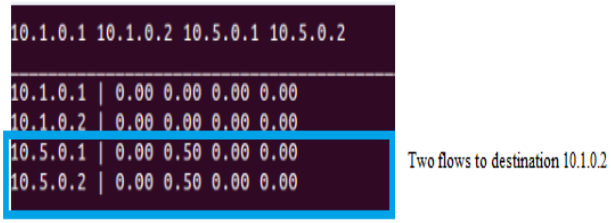


Figure 8. Demand estimation with two flows

We compare FPCA with ECMP to present the goal of the performance. Assuming that flow f sets the primary path (3005 → 2006 → 1003 → 2002 → 3001) as shown in Figure 9 and Table 3, the alternate paths used to protect every link on the primary path of a flow obtained using FPCA Algorithm is shown in Table 1.

Table 3. Primary Flows

Source-Destination	Primary Path
H9->H1, H10->H2	3005 → 2006 → 1003 → 2002 → 3001

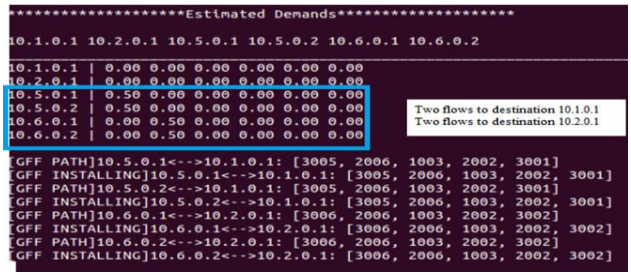


Figure 9. The primary path between the switches

We identify two types of collisions caused by hashing. First, flows 1 and 2 interfere locally at switch Aggregation2 due to a hash collision and are capped by the outgoing link to Core2(1003). In this example, flow 1 could have been forwarded to Core1(1002). The algorithm tries to select the best rule that can protect the flow without interfering other backup paths. Therefore, the congested traffic at switch (1003) will be rerouted through (3005 → 2005 → 1002 → 2001 → 3001) as shown in Figure 10 and Table 4.

Table 4. Rerouted Flows

Source-Destination	Rerouted path
H9->H1, H10->H2	3005 → 2005 → 1002 → 2001 → 3001

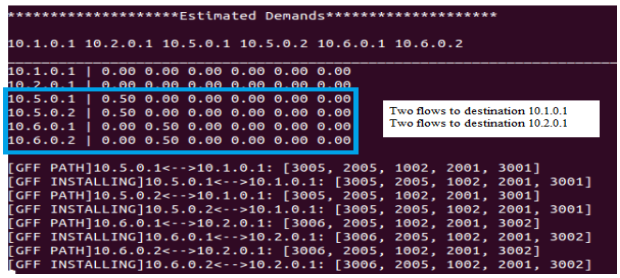


Figure 10. Rerouted path between the switches

The list of traffic flows between the switches using fat-tree network topology in the experiments as shown in Table 5.

Table 5. Traffic Flows between the switches

Source	Destination	FCT (sec)	Throughput (Mb/s)
H9	H1	8.8	95.6
H10	H1	8.8	95.7
H11	H2	9.0	92.8
H12	H2	8.9	95.6

After testing, we showed the experiment flow between (H9->H1, H10->H1, H11->H2, H12->H2) and its throughput was shown in Figure 11. According to the testing, we compare FPCA against ECMP. FPCA achieves better performance than ECMP. FPCA1 and FPCA2 have 39.54% and 37.76% more than ECMP1 and ECMP2. Throughput of ECMP decreases up to 60% due to hash collision of ECMP and increasing the number of flows and congestion in the network.

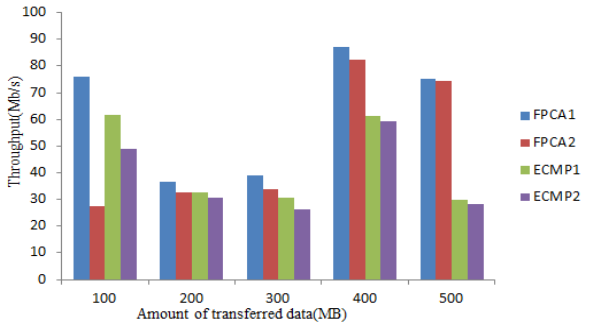


Figure 11. Throughput between different traffic types

We verified the comparison of the flow completion time in the two above-mentioned cases as shown in Figure 12. The FCT in ECMP increases from 13.6 to 141.1, while FPCA increases slightly from 11 to 55.8. For flow completion time, FPC achieves better performance than ECMP. FPCA1 and FPCA2 decrease 39.46% and 37.7% than ECMP1 and ECMP2. ECMP flows increase significantly 140s due to congestion.

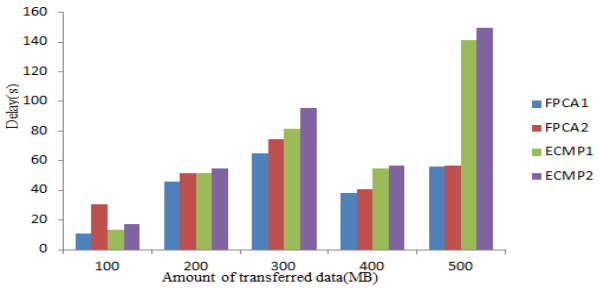


Figure 12. Flow completion time FCT between different traffic types

6. Conclusion

We presented the flow path computation algorithm. The proposed optimal path computation is based on flow demand calculation and port speed. If the existing flow's demand is exceeded 10% of link bandwidth, the flow is rerouted to light loaded path which is calculated using port speed. In this paper, FPCA was demonstrated better throughput and flow completion time compared to ECMP. In our experiments, FPCA increases the network throughput able to 2x. This paper has implemented a solution that can minimize the network congestion by rerouting the flows over the alternative paths in SDN. Based on the simulation results, the performance of our proposed algorithm outperforms flow hashing-based ECMP load-balancing. In our future work, we will extend the idea of long flow collision problem by encouraging for future progress.

7. References

[1] D. J. Hamad, K. G. Yalda and I. T. Okumus, "Getting traffic statistics from network devices in an SDN environment using OpenFlow". ITaS, September 2015, pp. 951-956.

[2] M.Gholami, B. Akbari. "Congestion Control in Software Defined Data Center Networks Through Flow Rerouting", 2015 IEEE, pp . 654-657.

[3] S. Song, J. Lee, K. Son, H. Jung, and J. Lee, "A Congestion Avoidance Algorithm in SDN Environment", International Conference on Information Networking (ICOIN), 2016 IEEE, pp. 420-423.

[4] Y. Liu, Y. Pan, M. Yang, W. Wang, C. Fang, and R. Jiang, "The multi-path routing problem in the Software Defined Network", 11th International Conference on Natural Computation (ICNC), 2015 IEEE, pp. 250-254.

[5] Y. HanS.S. Seo, J. Li, "Software Defined Networking-based Traffic Engineering for Data Center Networks", The 16th Asia-Pacific Network Operations and Management Symposium pp. 1-6. IEEE.

[6] C.Y. Chu, K. Xi, M. Luo, and H. J Chao, "Congestion-aware single link failure recovery in hybrid SDN networks", Conference on Computer Communications (INFOCOM), 2015 IEEE, pp. 1086-1094.

[7] M.M Tajiki, B. Akbari, M. Shojafar, M., S.H Ghasemi, M.L Barazandeh, N. Mokari, L. Chiaraviglio, M. Zink, "CECT: computationally efficient congestion-avoidance and traffic engineering in software-defined cloud data centers". Cluster Computing, 2018, pp.1881-1897.

[8] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead
datacenter traffic management using end-host-based elephant detection," in 2011 Proceedings IEEE INFOCOM, Apr 2011, pp. 1629–1637.

[9] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, F. Matus, R. Pan, N. Yadav, G. Varghese, et al., "CONGA: Distributed congestion-aware load balancing for datacenters," ACM SIGCOMM, 2014.

University of Information Technology
Parami Road, Universities' Hlaing Campus,
Ward (12), Hlaing Township, Yangon, Myanmar
Phone: (+95)-1-9664254
Fax: (+95)-1-9664250
URL - http://www.uit.edu.mm/