Dynamic load balancing in SDN-based data center networks

Conference Paper · October 2017

DOI: 10.1109/IEMCON.2017.8117206

2 authors:

Umme Zakia
Simon Fraser University
23 PUBLICATIONS **197** CITATIONS
SEE PROFILE

Hanene Ben Yedder
Simon Fraser University
23 PUBLICATIONS **256** CITATIONS
SEE PROFILE

Dynamic Load Balancing in SDN-Based Data Center Networks

Umme Zakia and Hanene Ben Yedder
Simon Fraser University
Vancouver, British Columbia, Canada
Email: {uzakia, hbenyedd }@sfu.ca

***Abstract*— With the migration of computational powers and applications to the cloud, Data Center Networks (DCNs) have become the backbone of the underlying infrastructure. Operation of the data centers relies on huge computational resources and bandwidth, that often undergo high operational costs, frequent link congestions, and imbalanced traffic loads. Software Defined Networking (SDN) based traffic load management improves accessing resources by distributing traffic among multiple paths efficiently and in a timely manner. In data centers, SDN-based traffic management techniques control paths of incoming flows and optimize flows during their transmissions. In this paper, we propose an SDN-based dynamic load management algorithm for optimizing link utilization in DCNs while considering the flow priority. The algorithm finds the shortest paths from each host to others and calculates every link's cost. When congestion occurs in a certain path, it replaces the old path with the alternative best route that has the minimum link cost and lower traffic flow. Performance of the algorithm is evaluated by measuring throughput, delay and packets loss in a fat-tree DCN. Simulation results show improved performance in load balancing over time as the algorithm keeps on running.**

Keywords— Data center networks; dynamic load balancing algorithm; software defined networking;

I. INTRODUCTION

Most online services and social networks require server's reliability and high availability. Therefore, they often use multiple servers and employ hardware load balancers to direct requests from clients to servers. However, hardware load balancers are very expensive and suffer from the rigid policy levied by the vendor. Software Defined Networking (SDN) technology opens a new area to the internet management process and offers flexibility in networking techniques, such as load-balancing and intrusion detection. SDN enables users to define their own cost-efficient and flexible applications to meet system requirement [1].

Data Centers Networks (DCNs), the underlying infrastructure of the Internet, often experience imbalanced traffic load as few links endure congestions while most of the links are underutilized. Today, dynamic flow management is of critical significance in data centers because accessing the computational and storage resources proficiently are crucial for the Quality of Experience (QoE) of the end users. Load balancing allows distributing traffic evenly among multiple paths, and therefore, it enables the network to process more data flow in less time. Using SDN, workflow can be changed dynamically in data centers to accommodate workloads more efficiently. SDN simplifies and improves the network management by splitting the control plane and the data plane along with sophisticated control mechanisms.

SDN based-DCN traffic management techniques utilize various path optimization algorithms in the SDN controller. These algorithms are employed to control paths of incoming flow and optimize flow path during their transmissions [2]. Most of them reroute flows according to the load of each link by retrieving load statistics from each switch. Therefore, they achieve static load balancing while resolving congestion problems. However, flow priority is omitted and all flows are treated in the same way. So, in this paper, we propose an SDN-based dynamic load management algorithm for optimizing link utilization in DCNs while taking into account the traffic flow priority. Performance of the algorithm is evaluated in terms of throughput, delay, and packets loss.

The remainder of this paper is organized as follows. In Section II and III, we describe related work and the proposed algorithm. Sections IV and V describe simulation process and results, respectively. We discuss the obtained results in Section VI and conclude with Section VII.

II. RELATED WORK

SDN-based load balancing is relatively a new era where many research works are being conducted. Different models have been proposed to optimize network flow, to resolve the network congestion problems by changing paths of flows during flow transmissions and to achieve load balancing among different links. Simple static path optimization approaches include common algorithms such as Round Robin, random, least traffic, and least latency. The main technical features of SDN is proposed and discussed elaborately [3]. A framework, OpenQoS, for video flows paths optimizes during flow transmissions based on the priorities of flows and the available bandwidths of links [4]. OpenQoS focuses mainly on achieving the QoS (Quality of Service) requirement and maintaining flow priorities instead of load balancing. An energy aware SDN routing algorithm is based on particle Swarm Optimization technique [5]. The algorithm determines the optimal energy efficient path and helps reduce the energy consumption of the networking resources.

LABERIO [6], a SDN-based dynamic path optimization algorithm reassigns flow to the path whose available bandwidth is the maximum among all possible paths. However, LABERIO considers instant values of the variance of link

978-1-5386-3371-7/17/$31.00 ©2017 IEEE

loads to trigger its path optimization algorithm, and therefore causes overhead in the network. In SDN-based data center networks for dynamic load-balanced path optimization (DLPO) [7], a multi-link and a single-link algorithm are proposed for data center networks. The multi-link DLPO algorithm balances link loads in the whole network while the single-link algorithm performs hop to hop flow rerouting to avoid highly loaded links. Another SDN based algorithm utilizes the hierarchical feature of the networks and real-time traffic statistics [8]. This algorithm implements a dynamic routing algorithm in the load balancer and distributes traffic of upcoming network flows equally among alternative paths to reduce latency. An SDN-based traffic engineering algorithm that considers both load balancing and power consumptions is designed for DCN [9]. The algorithm searches the optimal topology composition required to serve the traffic load and reduces the power consumption by turning off elements that are not included in this subset topology. Traffic load is thereafter balanced between links and switches of the selected subset.

III. Proposed Algorithm

We propose a dynamic load balancing algorithm that performs load balancing in a DCN topology by calculating minimum transmission costs of the links at a given time. DCN presents high multipath capabilities, so the Dijkstra's algorithm is implemented to find multiple paths of same length and reduce the search to a small region in the topology. The traffic flows are ordered according to their priority. Among selected paths, the path with least cost and load is selected and traffic flow is forwarded on that route. The new flows rules are then pushed to Open Virtual switches (OVSs) to update switch forwarding tables. The performance of the algorithm is evaluated in a Fat-Tree DCN topology by collecting operational data of the links and switches. The pseudocode of proposed algorithm is shown below:

Algorithm 1: Heuristic for Traffic Load Balancing

Input: T (Traffic Matrix), DCN Topology, Link Capacity

Output: Minimizing maximum link utilization path allocation of flows in T

1: For all possible flow f in T do
2: List flow f in ascending order according to priority
3: List all possible paths from f_{src} to f_{dst}
4: $list_p$ = Apply the Dijkstra's algorithm to find multiple paths of minimum length
5: For all path p in $list_p$ do
6: List $_{maximum\ link\ utilization}$ [p] = maximum link utilization of p
7: End For
8: $p_{selected}$ = $list_p$ [index of minimum in list $_{maximum\ link\ utilization}$]
9: Assign f to $p_{selected}$
10: For all links l in $p_{selected}$ do
11: Update flow switch table
12: End For
13: End For

IV. Simulation Process

The Floodlight OpenFlow controller is used as the control plane [10]. For the data plane, the Mininet topology generator [11] simulates the well-known fat-tree DCN topology [12].

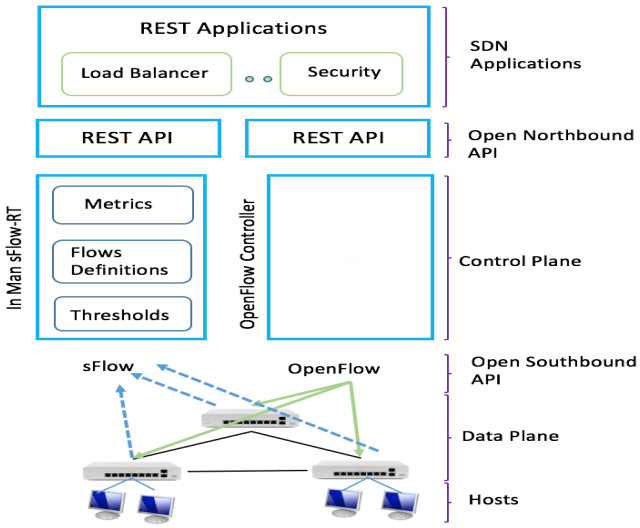

Fig. 1. SDN controller with REST API with load balancer application.

The Floodlight OpenFlow controller interacts with the OpenFlow switches in the Mininet using the OpenFlow protocol, as illustrated in Fig. 1. The Mininet emulated network consists of virtual hosts, OVSs, controllers, and links. Wireshark and Perf are used as network analyzer tools [13] [14].

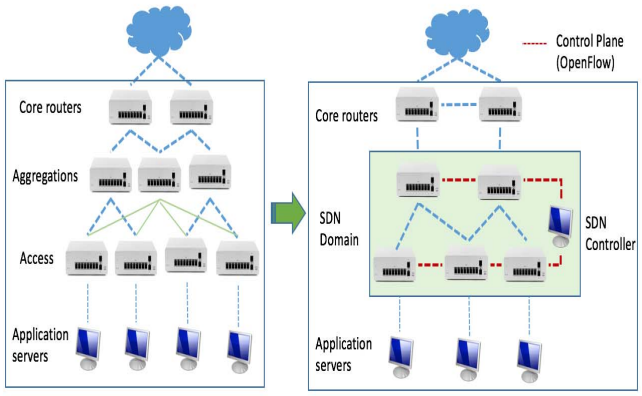

Fig. 2. SDN enabled data centers network architecture.

The proposed algorithm is applied as an SDN application using the REST API (Representable State Transfer API) as the northbound API. The REST API collects operational information of the topology and devices as well as instantaneous traffic and port statistics.

A. Simulation steps

The goal of the proposed load balancing algorithm is to improve traffic management in the DCN when congestion occurs. Initially, the switches and links with normal packet transmission (without applying load balancing) are observed and recorded under the phase as 'Before Load Balancing (BLB)'. Subsequently, when paths become congested, the load balancing algorithm is applied. Again, the switches and links are observed to evaluate the performance of the algorithm for traffic management under the phase 'After Load Balancing (ALB)'.

We connect the floodlight OpenFlow controller to the Mininet through the well-defined port 6653. The fat-tree DCN is generated with 8 hosts, 10 switches and 20 links by running the custom topology code written in python in the Mininet, as shown in Fig. 3. The 'Ping all' command is executed to check the reachability of all hosts in the defined network. The REST API collects information about hosts and switches such as IP addresses, MAC Addresses, port mapping, link bit rate, transmission rate, receiving rate etc. about the network for traffic management. REST API exhibits them as json data format on the 8080 port. Now, we apply the load balancing algorithm in python code and obtain path information to perform traffic management. Total link costs and the shortest paths are calculated for all these paths among the hosts.

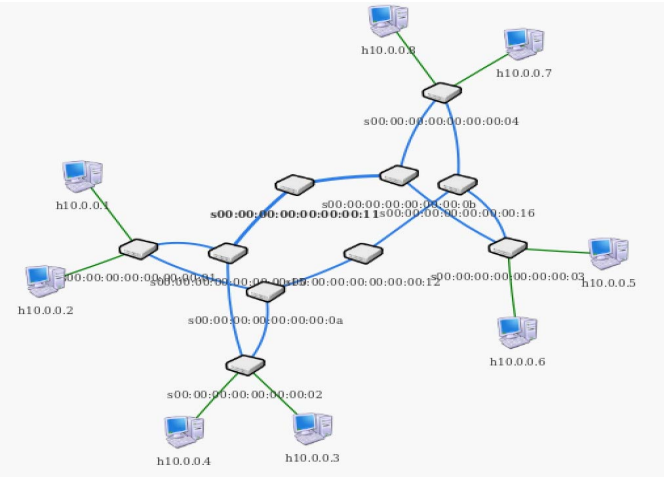

Fig. 3. Floodlight REST API exhibits the Fat-tree generated by Mininet.

The flows are ordered based on priority and minimum transmission cost (bits-per-second) of the links at the given time. When congestion occurs in a path, alternate best path is decided based on the minimum distance and link cost. The static flows are then pushed into each switch of the selected best path. Information such as In-Port, Out-Port, Source IP, Destination IP, Source MAC, Destination MAC are fed to the flows. Load balancing becomes dynamic as we keep the algorithm running for continuous update of the information.

V. Simulation Results

A. Before Load Balancing (BLB)

The load balancing algorithm can run between any hosts in the defined network. For clear understanding of the procedure, we choose as source: host 1 (h1) and destinations: host 3 (h3) and host 4 (h4). Host h1 transmits packets through the open virtual switch (OVS) s1-eh1 (switch s1 port 1); switch s1 then forwards the packets towards h3 and h4 via s1-eh3 (switch s1 port 3), as shown in Fig. 4. Wireshark is used to capture the paths that switch s1 selects to reach destinations.

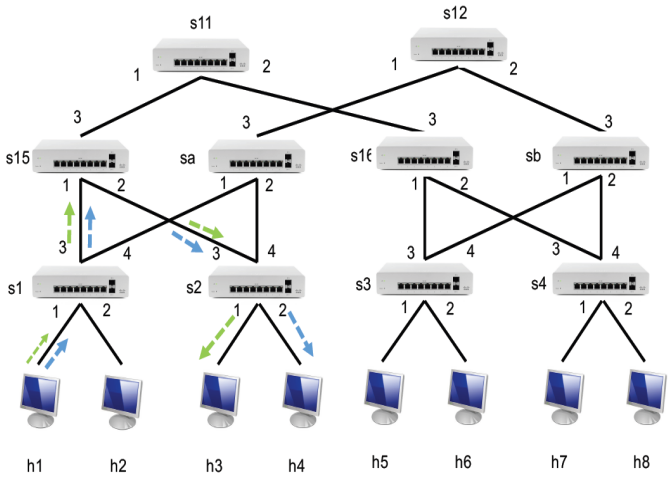

Fig. 4. Observed traffic between host 1 to host 3 and 4 in the fat-tree DCN before load balancing (BLB).

1) Collecting Traffic load and path

Traffic packets are sent from h1 to h3 and h4 by ping command that uses ICMP packets of 53 bytes. The wireshark captures the ICMP packets sent to the destination hosts through switch s1. Both the hosts h3 and h4 are receiving the packets from h1 through the switch s1-port: eth3. Fig. 5. shows that no traffic is captured in the links from s1-port: eh1 to s1-port: eh4, which suggests imbalanced loads among the links.

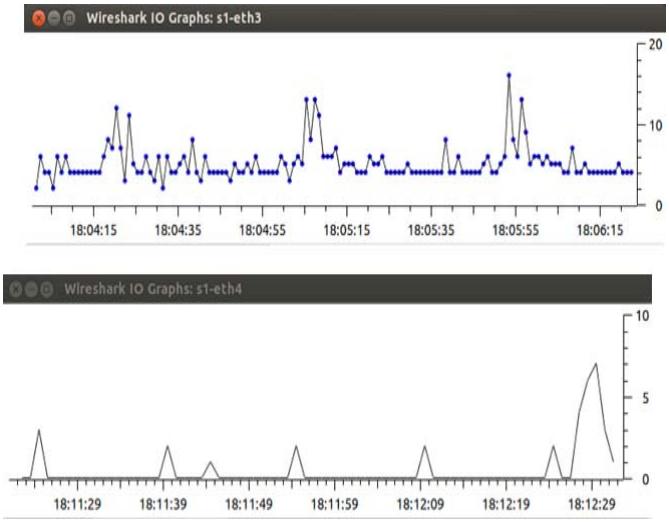

Fig. 5. Wireshark captures traffic between h1, and h3, h4.

The traffic flows from h1 to h3 and h4 are routed through the switch s1 out port 3 (s1-eth3) only, no flow is found in the switch1 out port 4 (s1-eth4). Small traffic picks shown in port 4 (s1-eth4) is due to the DLDP (device link detection protocol) packets sent frequently from the REST API to collect statistics.

2) Measuring Delay and packet loss:

The round-trip time (RTT) is measured for h1 to h4. Total time (ms) to transmit packets with their transmission loss, RTT (minimum, average and maximum) are collected for sampling five times over the period of transmission, as shown in Fig. 6. The transmission continues without load balancing, which results in higher RTT that is not optimized.

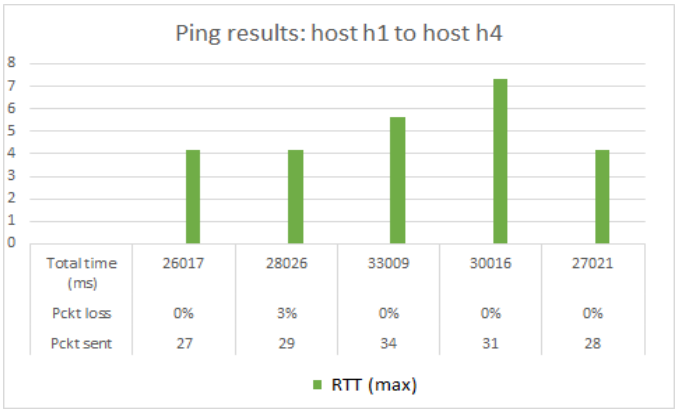

Fig. 6. Delay and packet loss between h1 and h4 before load balancing (BLB).

B. *After Load balancing (ALB)*

The load balancing algorithm is executed to find the alternate best paths when the initial path from source to destinations becomest congested. To verify this proposition, the path from h1 to h3 is congested purposely to observe the performance of load balancing between h1 and h4. The algorithm finds the optimal path for h1 to h4 via s1-eh4 (switch s1 port 4), as shown in Fig. 7.

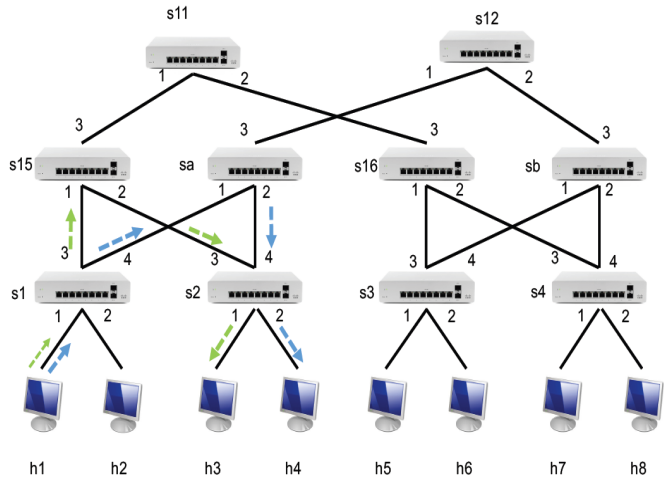

Fig. 7. Observed traffic between host 1 to host 3 and 4 in the fat-tree DCN after load balancing (ALB).

The output of the load balancer algorithm shown in Fig. 8 demonstrates the paths from the source to the destination, the calculated link costs, the priority of the links and the new path updated.

```
########### RESULT ###########


Paths (SRC TO DST)

{'02::15::01': ['00:00:00:00:00:00:00:02', '00:00:00:00:00:00:00:15', '00:00:00:00:00:00:00:01'], '
02::0a::01': ['00:00:00:00:00:00:00:02', '00:00:00:00:00:00:00:0a', '00:00:00:00:00:00:00:01']}

Final Link Cost

{'02::15::01': 868293, '02::0a::01': 15}

Host priority

{'10.0.0.4': 4444, '10.0.0.1': 1111, '10.0.0.3': 3333}


Shortest Path update:  02::0a::01  from src  10.0.0.1 to dest  10.0.0.4


#####################################
```

Fig. 8. Observed results of balancing the traffic in the fat-tree DCN between source h1 and destinations h3, h4 after load balancing (ALB).

1) Collecting Traffic load and path

Wireshark captures the packet flows path from h1 to h4 through the interfaces s1-eh3 and s1-eh4. Before load balancing, packets were routed through the s1-eh3 only. Now, the load balancing algorithm is pushing flows to the s1-eh4 to reach h4 because the link (h1 to s1-eh3) is already congested for packets from h1 to h3. This demonstrates that the algorithm is working properly by using a substitute route to avoid congestions. Fig. 9 shows that traffic is evenly shared between s1-eh3 and s1-eh4.

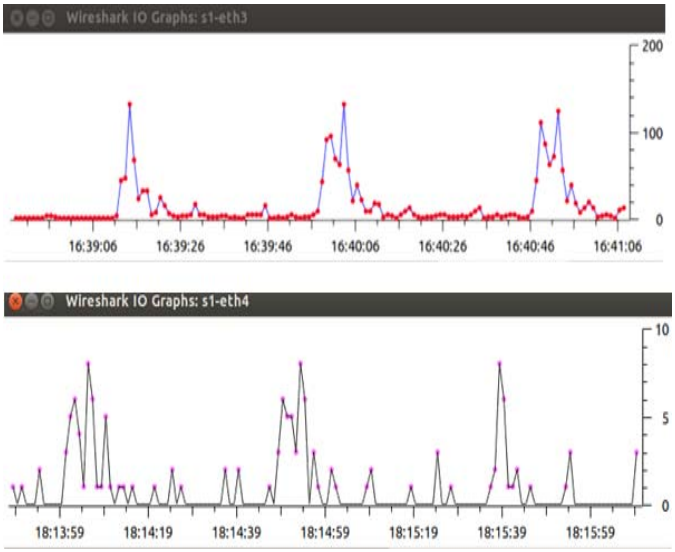

Fig. 9. Wireshark captures traffic between h1 and h3 and h4. Both the out-ports of switch s1-eh3 and s1-eh4 are used for trafficking packets.

2) Measuring Delay and packet loss

Again, the total time (ms) to transmit packets with their transmission loss, RTT (minimum, average and maximum) are

collected by sampling five times over the period of transmission with load balancing. Fig. 10 shows that the RTT reduces over the transmission period and becomes optimized.

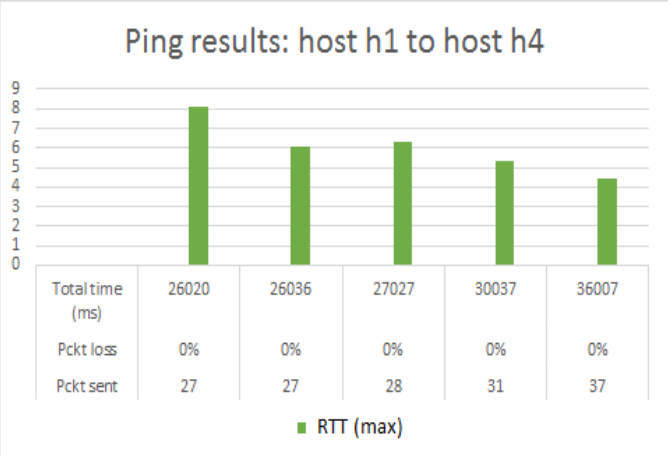

Fig. 10. Delay and packet loss between h1 and h4 after load balancing (ALB).

C. *Measuring Throughput in terms of Bandwidth (considering both BLB and ALB)*

Five samples of the bandwidth utilized by h1 and h4 before and after load balancing are collected using iperf. It is observed that bandwidth consumption is higher before load balancing (BLB). As soon as the load balancing algorithm starts working, it sets alternate paths and reduces load in the path of h1 to h4. As a result, the bandwidth consumption gradually starts increasing after load balancing (ALB) which results in better throughput, as shown in Fig. 11.

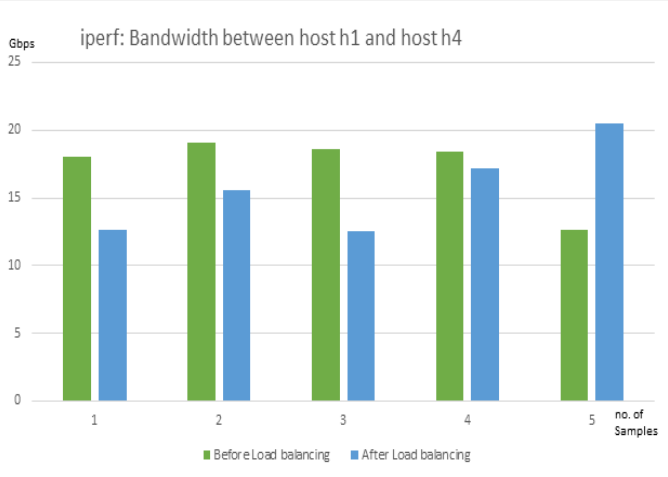

Fig. 11. Bandwidth consumption of the link between h1 and h4 before and after load balancing as captured by iperf.

VI. DISCUSSION

Before applying the load balancing algorithm, we observed the traffic between host 1 (h1) as source and hosts 3 and 4 as destinations (h3 and h4, respectively). Despite the multipath capability of the simulated fat-tree topology, traffic flows from s1 switch to s2 switch use the same path through s1-eh3 instead of using a second path with the same cost (s1-eh4) and more alternative paths with higher costs. Therefore, few links experience congestions while the majority of links remain underutilized resulting in an imbalanced traffic load.

Simulation results show that the load balancing algorithm handles better packet trafficking. The proposed algorithm dynamically collects data and link statistics from the REST API in terms of available links and transmitted bit-per-second for each hosts and switches. It finds the shortest path with least cost and load, and updates the switch table with new flow rules. In the simulation process the load balancing algorithm pushed flows to the s1-eh4 to reach h4 as the s1-eh3 link experienced higher utilization. Hence, the load balancing algorithm replaces the old path with the alternate best route with minimum link cost and shares the traffic evenly among the different shortest paths.

It is revealed in the simulation process that the load balancing algorithm enables lower delays by reducing round trip time, a higher throughput by increasing available link bandwidth, and lower packet loss. This proposition indicates that the proposed dynamic load balancing algorithm can enhance end user quality of experience and enable a better usage of the underlying infrastructure.

VII. CONCLUSION

Data Center Networks (DCNs) are the backbone of the Internet services. Now-a-days, SDN significantly simplifies and improves DCN management. Performance of data centers rely on proper resources utilization for timely services to the end users. Traditional load management employed in DCNs suffer as few links experience congestions while majority of the links are underutilized. SDN-based DCN traffic management techniques offer better flow management. In this paper, we propose and implement an SDN-based dynamic load management algorithm for optimizing link utilization in DCNs. The algorithm finds the shortest paths from each host to others and calculates every link's cost while considering flow priority. When congestion occurs, it replaces the old path with the alternate best route with minimum link cost that has shortest distance. As the code keeps running, it improves load balancing performance with reduced delay and increased bandwidth. The algorithm focuses to improve link utilization only; while in future we would include node utilization for effective network virtualization.

REFERENCES

[1] R.Masoudi and A. Ghaffari, "Software defined networks: A survey," J. Network and Computer Applications, vol. 67, pp. 1-25, May 2016.

[2] M. Randles, D. Lamb, and A. Taleb-Bendiab, "A comparative study into distributed load balancing algorithms for cloud computing," in *Proc. Inter. Conf. on Advanced Information Networking and Applications*, Perth, Australia, April 2010 pp. 551-556.

[3] A. Yassine, H. Rahimi, and S. Shirmohammadi, "Software defined network traffic measurement: Current trends and challenges," *IEEE Instrumentation & Measurement Magazine*, Vol.18 no.2, pp. 42-50, Mar. 2015.

[4] H. E. Egilmez, S. Civanlar, and A. M. Tekalp. "An optimization framework for QoS-enabled adaptive video streaming over OpenFlow networks," J. *IEEE Transactions on Multimedia*, vol. 15, no. 3, pp. 710-715, April 2013.

[5] S. Subbiah and P. Varalakshmi. "Energy-aware network resource allocation in SDN. " in *Proc. Inter. Conf. on Wireless Communications,*

Signal Processing and Networking, Chennai, India, Mar. 2016, pp. 2071-2075.

[6] H.Long, Y.Shen, M. Guo and F. Tang, "LABERIO: Dynamic load-balanced routing in OpenFlow-enabled networks," in *Proc. Inter. Conf. on Advanced Information Networking and Applications*, Barcelona, Spain, Mar. 2013, pp. 290-297 .

[7] Y. L. Lan, K. Wang, and Y. H. Hsu, "Dynamic load-balanced path optimization in SDN-based data center networks," in *Proc. IEEE Inter. Conf. on Communication Systems, Networks and Digital Signal Processing*, Prague, Czech Republic, July 2016, pp. 1-6.

[8] Y. Li and D. Pan, "OpenFlow based load balancing for Fat-Tree networks with multipath support," in *Proc. IEEE Inter. Conf. on Comm.*, Budapest, Hungary, June 2013, pp. 1-5.

[9] Y. Han, S. S. Seo, J. Li, J. Hyun, J. H. Yoo, and J. W. K. Hong, "Software defined networking-based traffic engineering for data center networks," in *Proc. IEEE Int. Conf. on Network Operations and Management Symposium*, Taiwan, pp. 1-6, Sept. 2014.

[10] Floodlight OpenFlow Controller [online]. Available: http://www.projectfloodlight.org/floodlight/.

[11] Mininet: An Instant Virtual Network on Your Laptop (or Other PC) [online]. Available: http://www.mininet.org/.

[12] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, Oct. 2008.

[13] Wireshark. Go Deep. [online]. Available: https://www.wireshark.org/.

[14] iperf - The TCP, UDP and SCTP network bandwidth measurement tool [online]. Available: https://www. iperf.fr/.