International Journal of Science and Research (IJSR)
ISSN: 2319-7064
ResearchGate Impact Factor (2018): 0.28 | SJIF (2018): 7.426

Implementation of SDN Controller as Load Balancing Using Datacenter Topology

Zahraa Alaa Baqer[1], Mohammed Najm Abdullah[2]

[1, 2]Department of Computer Engineering, University of Technology, Baghdad, Iraq

Abstract: ***The load balancing is the most important issue which helps to reduce traffic in datacenter topology. However balancing the traffic load and optimizing better path are very important .In this paper we used Dijkstra's Algorithm to implement the load balancing in software define network (SDN) controller. Mininet and floodlight controller were used to setup the network environment. The result show that the load balancing algorithm is increased bandwith and reduced latency of the network, in order to achieve a better performance and better resource utilization of the network. .***

Keywords: Software Define Network (SDN), Floodlight Controller, Dijkstra's algorithm, Mininet, Dynamic Load Balancing, OpenFlow(OF), Jperf

1. Introduction

Software define network (SDN) is an emerging technology and is a shortcut to the next generation of infrastructure in network engineering. SDN requires some mechanisms for the centralized controller to communicate with the distributed data plane as shown in Figure 1. In this way, the controller is a basic segment of the SDNs design that add achievement or disappointment of SDN. SDN is controlled by software applications and SDN controllers rather than the traditional network management consoles and commands that require a lot of administrative overhead and could be tedious to manage on a large scale[1].

SDN can transform today's static networks into more flexible, programmable platforms to provide scalability to support large data centers. It also provides virtualization that is needed to support automated, dynamic and secure cloud environment[2].

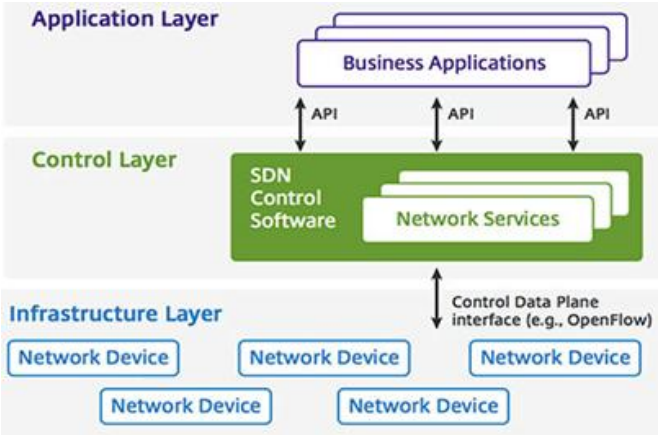

Figure 1: The Architecture of SDN

In this paper, we proposed a dynamic load balancing algorithm that are implemented in SDN controller based on datacenter network topology. We are using a Dijkstra's algorithm to compute the shortest paths of the same length and link cost between nodes based on the hop count. In load balancing algorithm, we can find the best path of each of the link present that can adapt to the topology changes. The floodlight controller used to implement the load balancing algorithm and compare it with before load balancing algorithm, which are used the fat tree datacenter network topology that run on the Mininet emulation tool.

2. Related work

SDN is a hot topic for research these days; meaning that there are many published papers about different SDN related investigation. Some of those related to the problem statement are discussed next.

J.-R. Jiang et al., 2014 [3] proposed a load balancing algorithm and a multicast algorithm in SDN on the basis of the extended Dijkstra's algorithm for a graph derived from the underlying SDN topology. Which used Pyretic to implement the proposed algorithms with the Mininet emulation tool. The simulation results show that the proposed load balancing algorithm outperforms others in terms of the end-to-end latency, response time, throughput, and standard deviation.

G. Senthil and S. Ranjani, 2015 [4] proposed an SDN approach using OpenFlow protocol which was implemented to improve efficiency using round robin load balancing. Here the http demands from various customers will be coordinated to various predefined characterized http servers depending on round robin algorithm.

Y.-L. Lan et al., 2016 [5] used SDN-based datacenter networks for dynamic load balanced path optimization, using EstiNet network simulator to build datacenter topology. Load balancing, which changes paths of flows during flow transmissions, achieves load balancing among different links, and efficiently resolves the congestion problem in datacenter networks.

M. Beshley et al., 2017 [6] proposed the model of adaptive routing of heterogeneous traffic with respect to the current requirements regarding quality of service provisioning. The testbed uses the Mininet to investigate the behavioral characteristics of SDN and Open vSwitch as OpenFlow enabled switch. Traffic is generated by the Iperf application.

A network testing tool capable to create TCP and UDP traffic between nodes.

Sminesh C. N. , 2019 [7] implemented a proactive traffic analysis based on load balancing using OF messages. The SDN control plane periodically monitors flow level statistics and utilization in network, which improve throughput and the network efficiency.

3. Technical background

3.1 SDN controller

Controller decides where and how to make forwarding decision based on application that is written on top of floodlight controller. In the SDN, the controller is a global view of the network topology. Here we selected floodlight controller as a suitable open source, Java based OF controller in SDN,which is available under the Apache 2.0 license. The Floodlight is intended to work with the developing number of switches, routers, virtual switches, and access points that support the OF standard as shown in Figure 2. Southbound interface in controller provides information to the switch and northbound interface allow floodlight controller to interact with various applications. It provides a framework for communicating with SDN switches using OF protocol. OF is the most common protocol used in SDN approach which are used to communicate the controller with all the network elements[8].

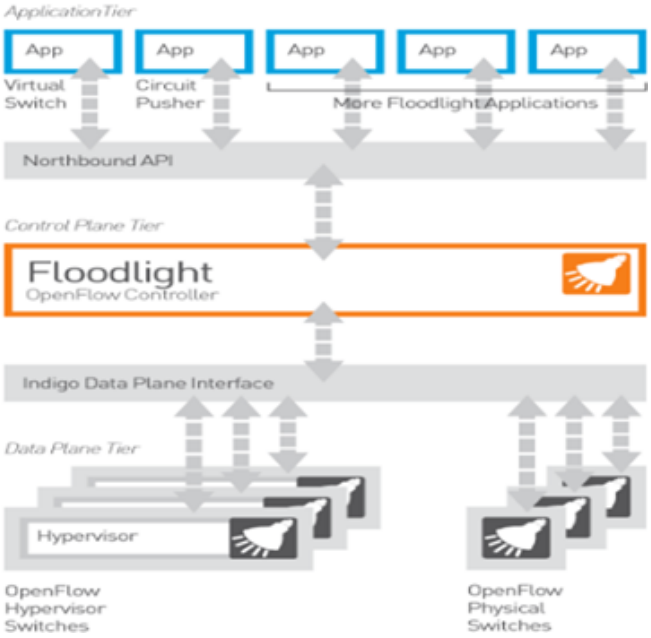

Figure 2: The structural design of floodlight controller[8].

3.2 Mininet emulation

It is a network emulator which makes a system of virtual hosts, switches, controllers, and connections. Mininet has run standard Linux arrange programming, and its switches bolster OF for profoundly adaptable custom steering and SDN technology. That can easily connect with system by software CLI (Command Line Interface) and API (application program interface). Mininet is utilized generally in view of: quick to begin a straightforward system, supporting custom topologies and bundle sending, running genuine projects accessible on Linux, running on PCs, servers, virtual machines, having sharing and recreating capacity, simple to utilize, being in open source and dynamic advancement state. Mininet uses process based virtualization to emulate entities on a single OS kernel by running a real code, including standard network applications, the real OS kernel and the network stack. Therefore, a project that works correctly in Mininet can usually move directly to practical networks composed of real hardware devices. The code that is to be developed in Mininet, can also run in a real network without any modifications. It supports large scale networks containing large number of virtual hosts and switches[9] [10] [11] .

3.3 Jperf

Jperf stands for java perf. It is a graphical tool open source written in java that represents a GUI for running iperf (Internet Performance Working Group) without having to bother learning the CLI options. Which executed between any two hosts in client/server model to analysis the traffic for TCP or UDP. The server displays the result of test network performance while the client acts to send the traffic to server via IP and port number of server. Jperf is an exceptionally valuable and dependable apparatus to quantify the feasible throughput and jitter in system interface. Jperf is a simple framework for writing and running automated performance and scalability tests, also be used to measure bandwidth,packet loss, delay, jitter, and other common network problems [12].

Jperf is a useful tool which can be used to measure performance on IP networks, as shown in Figure 3. The test results are automatically graphed and presented in a format that is easy to read. Jperf provides many benefits over iperf which is a command line only application. Besides being reliable and easy to use, jperf is completely free. The utility is fully open source and runs on both Windows and Linux systems[13].

Jperf tool can be used in this paper for measuring the performance of SDN technology in Mininet emulation, which can be run and used by the following steps:

- Download jperf tool version 2.0.2 on Ubuntu [14].
- The 'java' (JRE 1.5+) executable has to be in the system path.
- Decompress the file of jperf by double-clicking and from the terminal navigate to the untarred directory.
- Set execution permissions on the jperf.sh script (execute 'chmod u+x jperf.sh').
- In the terminal, run the following script ‘./jperf.sh’.

After running jperf software between two devices in the network, the results can be achieved by selecting the following options, as shown in Figure 3:

1) Choose iperf mode:
   For client, use the IP for server address and port number.
   For server, use listened port number.
2) In application layer options, close the numbers transmitted in seconds.

3) Select the output format such as Gbits, Gbytes, Mbits, Mbytes, Kbits or Kbytes for the application layer options.
4) In transport layer options, choose the protocol used for measuring the performance of networks such as TCP or UDP.
5) Run the software by closing 'run iperf' button.
6) Save the results.

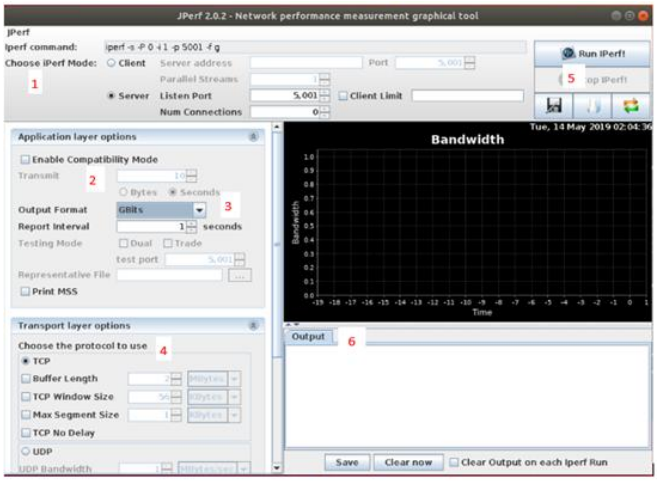

Figure 3: Jperf tool.

3.4 OpenFlow(OF)

The OF is a protocol between SDN controller and switching devices that defines an API between forwarding layer and controller in SDN. The OF protocol is supported network devices that correspondence between the control and data planes which access the forwarding data layer to provide an external application of network devices[15].

The network operting system is connected with SDN device by control communicaition in controller as seen in Figure 4. There are three parts associated in this protocol which are secure channel, groub table and more than one of flow tables that can communicate by OF swiches as shown in Figure 5. Secure channel is responsible for communicating a virtual switch like Open Virtual Switches(OVS) with SDN controller remotely, also for exchanging OF messages between an OF switch and OF controller. In turn, each switch consists of a series of tables, implemented in hardware or firmware to manage the flows of packets through the switch. All incoming packets from a particular flow are matched with the flow table. The flow table describes the functions that are to be performed on the packets. There may be one or more flow tables. A Group table does many actions on one or more flows. Flow table directs a flow to the group table. This can define the capacities and protocols used to centrally manage switches via a centralized controller. OF uses the concept of flows to identify network traffic based on pre-defined match rules which can be statically or dynamically programmed by the SDN control software[2, 16, 17].

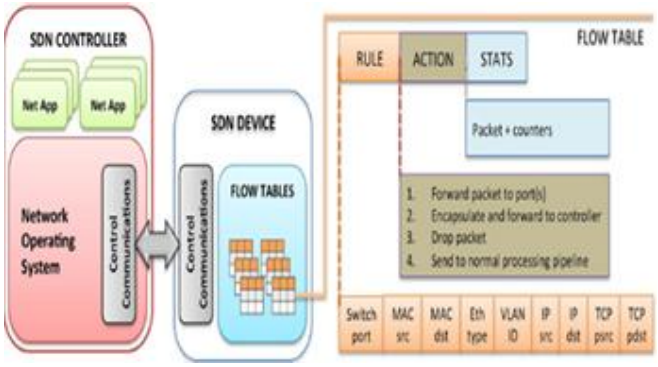

Figure 4: The OpenFlow protocol with SDN controller

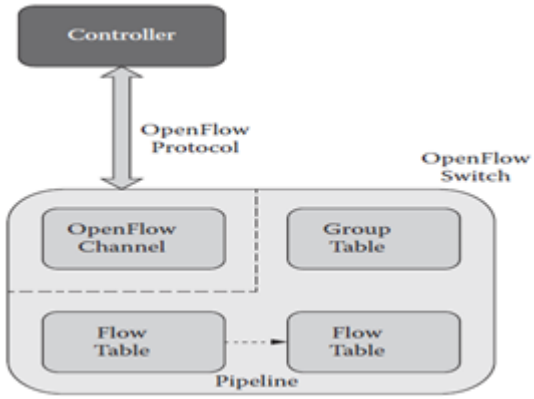

Figure 5: The OpenFlow protocol with OpenFlow switch

4. System model for SDN

In this section, we present the system model for SDN using a dynamic load balancing algorithm for SDN in detail, also presented the network design and implemented based on datacenter network topology.

4.1 Proposed system

The proposed algorithm is Dijkstra algorithm that used to compute the shortest path between two nodes over network based on the hop count, which enable to minimize the search to a littel region in fat tree topology, and our basic aim is to achieve the efficient result and higher performance.

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 OVS to update switch forwarding tables. The performance of the algorithm is evaluated in a fat tree datacenter topology by collecting operational data of the links and switches. The pseudocode of proposed algorithm is shown below:

Algorithm 1: Dynamic Load Balancing

```
Input: Traffic Matrix(T), Datacenter Network Topology(DCN),
Link Capacity
Output: Minimizing MLU path allocation of flows in T
    1. For all flow f in T do
    2. List flow f in ascending order to priority
    3. Listp list all possible path from fsrc to fdst
    4. Listp = apply the Dijkstra Algorithm
    5. For all Pathp in Listp do
    6. List MLU[P]= MLU of path p
    7. End for
    8. Psel=Listp [index of minimum in list MLU]
    9. Assign f to Psel
    10. For all link l in Psel do
    11. Update flow switch table
    12. End for
    13. End for
```

Dijkstra algorithm can be used for both directed and undirected graphs, and G is the graph data structure generated by networkx library. Source is the starting point and target is the ending point. The start point, end point, and weight are all optional parameters. Algorithm 2 shows how to compute shortest path between nodes in the graph by networkx.

Algorithm 2: Dijkstra's algorithm by networkx function

```
all_shortest_paths(G, source, target, weight=None,
method='dijkstra')
G = nx.Graph()
nx.add_path(G, [2, 8, 1])
nx.add_path(G, [2, 7, 1])          As shown in Figure 6.
print([p for p in nx.all_shortest_paths(G, source=2, target=1)])
```

Networkx is a Python language software package for the creation, manipulation, and study of the structure, dynamics, and function of complex networks. Networkx supports the creation of simple undirected graphs, directed graphs, and multigraphs; built-in many standard graph theory algorithms, including Dijkstra algorithm, where nodes can be arbitrary data, such as image files. With networkx you can load and store networks in standard and nonstandard data formats, generate many types of random and classic networks, analyze network structure, build network models, design new network algorithms, draw networks, and much more. It supports arbitrary boundary value dimensions, feature-rich, and easy to use. To install the networkx graph on ubuntu, use the following command[18]:

$ sudo apt-get install networkx

Start python for importing the package of networkx by the following command:

import networkx as nx
G = nx.Graph()

Table 1 shows the parameters that are used in the algorithm with networkx.

Table 1: Key notation in the algorithms

Parameter	*Description*
G	Networkx graph
source	Starting node for path
target	Ending node for path
weight	(None or string, optional (default = None)) – If None, every edge has weight/distance/cost 1. If a string, use this edge attribute as the edge weight. Any edge attribute which is not presented defaults to 1.
method	(string, optional (default = 'dijkstra')) – The algorithm to compute the path lengths. Supported options: 'dijkstra
Returns paths	A generator of all paths between source and target.
Return type	generator of lists
T	Traffic Matrix
DCN	Datacenter Network Topology
MLU	Maximum Link Utilization
L	Link for path selected
fsrc	Flow of source
fdst	Flow of destination
Listp	list of all possible path
Psel	Path selected

4.2 Network design

The network design in the second scenario uses datacenter network topology(Fat Tree Topology) to implement load balancing in the SDN application. Datacenter network topology is created by a custom topology in Mininet emulation such that analysis of load balancer script is possible. The fat tree topology with k=3 (k for number of layers) arrangement is considered in the work and implemented as shown in Figure 6. The topology here consists of 12 hosts and a total of 15 switches with 3 core switches and 6 switches for each of the aggregation and the edge layer. We wrote a python script in which we made API calls to addlink(), addHost(), and addswitch().

- addlink():It is used to add link either to connect switch-switch or to connect switch-host.
- addHost():It is used for addition of hosts in the network topology by stating its IP address.
- addswitch():It is used to add new switches in the network topology.

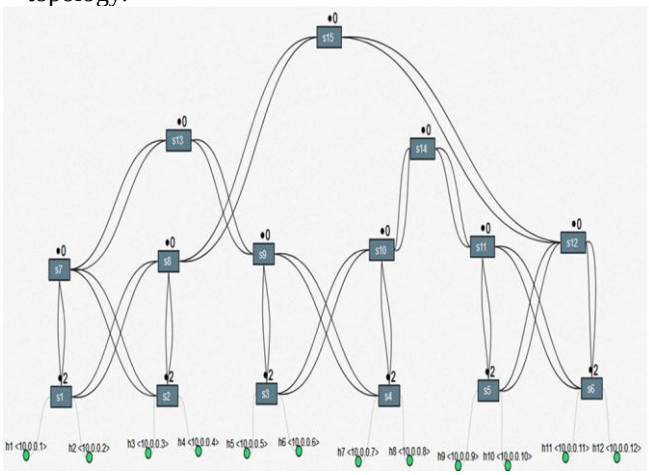

Figure 6: Datacenter network topology used in simulation.

4.3 Implementation the proposed system

The main steps to implement the proposed algorithm based SDN environment are outlined in the flowchart shown in Figure 7, and it can explain the steps for implementation the load balancing algorithm in details as follows :

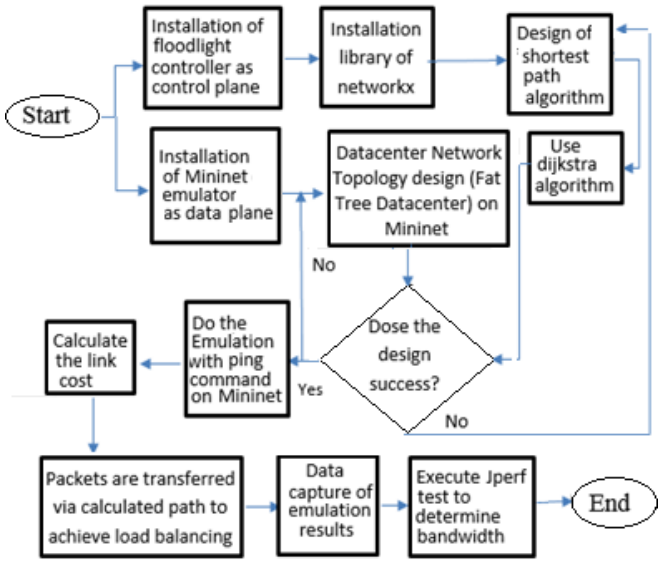

Figure 7: Steps to implemented load balancing algorithm using SDN.

- Created a custom topology (Fat tree datacenter network) using python language in mininet with (12) hosts and (15) switches. In the first run the floodlight controller outside of mininet and the executed topology in mininet network using remote controller with IP address for controller to set it to use.
- Test the connectivity for network performance till there is (0) % dropping in the packets. Testing is done by running the command ping from source to destination or pingall command on mininet to see the testing ping connectivity of the hosts in the network.
- Run the load balancing algorithm between two host, in this paper using h1 (10.0.0.1) and h4 (10.0.0.4) for testing.
- Check output port of switches before load balancing by ping h1 to h4 to verify the packet flow using Wireshark. Also check the bandwidth utilization by using iperf command between h1 and h4.
- Enter hosts (h1 and h4) and neighbor host (h3) after run the python script for load balancer.
- Create dictionaries to store information like:
  IP and Mac.
  Host, switch ports.
  Link ports (source and destination, source and destination port).
  Paths (source to destination).
  Final link cost (first to second switch).
- Retrieve device information like IP address, MAC, ports, switched and store these in corresponding dictionaries.
- Get the information of all connected switches with their links. Extract the source, destination switches and their corresponding port numbers.
- Calculate route/path from source to destination. First, the graph will give the shortest paths based on minimum hop counts from source to destination and link costs will be calculated for these paths using networkx, which finds the path with the least load and forwards traffic on that path. Store the paths in path dictionary with key as switch IDs.
- After fetching route information, access the REST API and retrieve transmission rate and add it to the cost variable. This will give link costs for all the links for the switches in the shortest paths.
- Find the minimum cost path from source to destination and add a flow rule based on these link costs.
- After load balancing: Use jperf command and ping from h1 to h4 and observe the increase in bandwidth, which verifies that the load balancer is working. Estimate the ping time from source to destination and verify that ping time decreases after running the load balancer script.

5. Result and discussion

The SDN Load Balancing is achieved using Dijkstra's algorithm to find the shortest path based on the number of hop and using networkx package python for graphs, which can be shown by using four different methods:

1) Ping statistics of hosts between which load balancing is done
2) Bandwidth of the path using iPerf and jperf
3) The best path using Wireshark
4) Total link costs and the shortest paths are calculated for all these paths among the hosts.

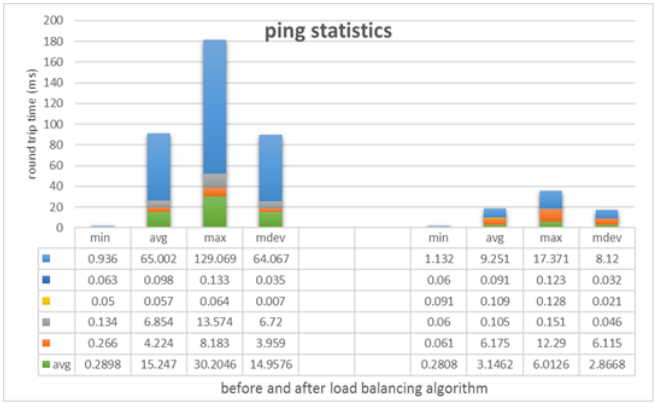

Figure 8: Results for ping statistics before and after load balancing algorithm.

Figure 8 shows the results of the run of the ping command, where h1 and h4 are chosen to test RTT before and after using the load balancing algorithm. The average ping time before Load Balancing was (15.247) ms, but after Load Balancing the average time reduced to (3.1462) ms, as shown in the figure above.

The path bandwidth was checked before and after load balancing. **Iperf and jperf** tools are used to measurement bandwidth for TCP or UDP. In TCP measurements, the results show that the bandwidth of the path before load balancing was much less than the bandwidth of the best path selected after load balancing as shown in Figure 9 and Figure 10.

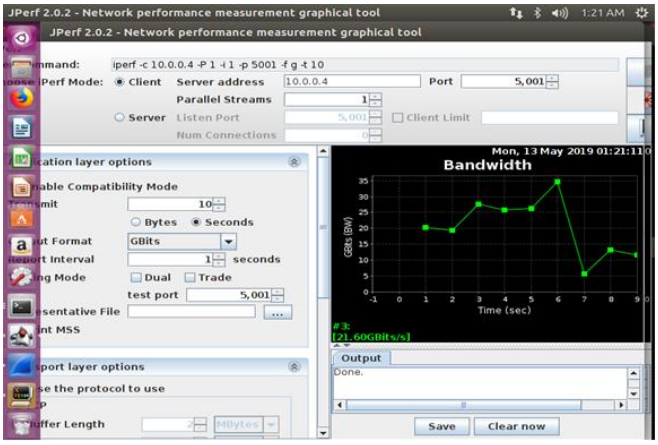

Figure 9: Bandwidth of the Path before Load Balancing

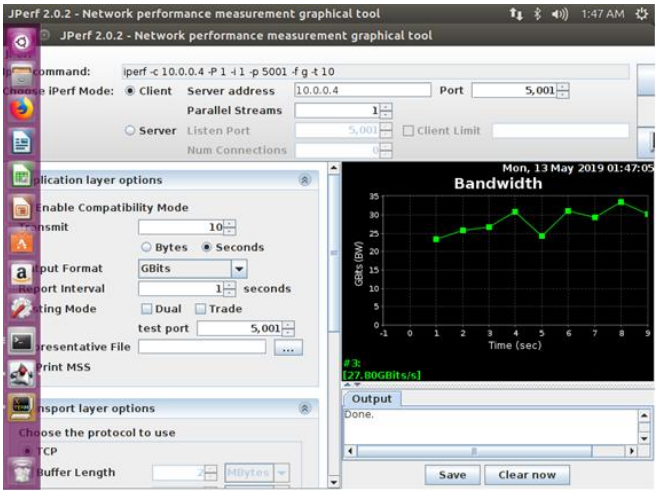

Figure 10: Bandwidth of the Path after Load Balancing

In UDP measurements, the bandwidth and jitter tests using jperf tool. The results show that the bandwidth before and after running load balancing is (0) Gbits/second, since the SDN controller acts as a load balancing and in the UDP no transmission was used to checked jitter and packet loss. The results of jitter measurement show that the jitter before load balancing is (1.18) ms and was decreased after load balancing to become (0.05) ms, therefore the path before load balancing was much higher than the jitter of the best path selected after load balancing as shown in Figure 11 and Figure 12.

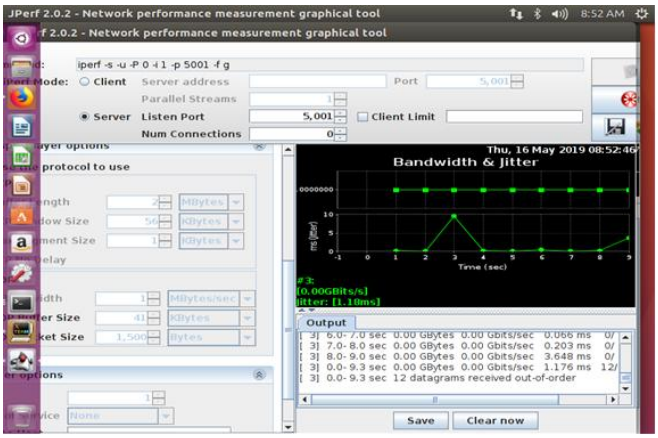

Figure 11: UDP before Load Balancing.

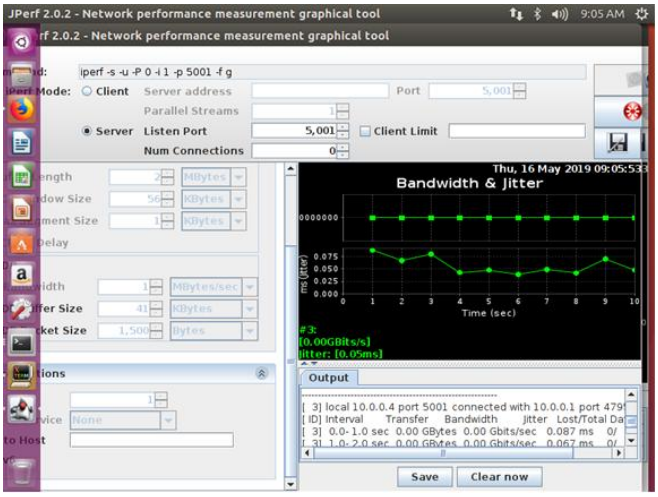

Figure 12: UDP after Load Balancing.

6. Conclusion

The load balancers are successfully implemented in the SDN controller under the fat tree datacenter topology. In this paper we implemented Dijkstra's algorithm with OF switch in SDN environment using Mininet emulation tools and floodlight controller. The network performance was tested before and after running the load balancing algorithm. As the results show, implementing a load balancer algorithm with SDN controller do perform better throughput and decreacing the latency of the network, to accomplish a much better performance of the network.

References

[1] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys & Tutorials*, vol. 17, pp. 27-51, 2015.

[2] D. Kreutz, F. M. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, pp. 14-76, 2015.

[3] J.-R. Jiang, W. Yahya, and M. T. Ananta, "Load Balancing and Multicasting Using the Extended Dijkstra's Algorithm in Software Defined Networking," in *ICS*, 2014, pp. 2123-2132.

[4] G. Senthil and S. Ranjani, "Dynamic Load Balancing using Software Defined Networks," *International Journal of Computer Applications*, 2015.

[5] Y.-L. Lan, K. Wang, and Y.-H. Hsu, "Dynamic load-balanced path optimization in SDN-based data center networks," in *2016 10th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*, 2016, pp. 1-6.

[6] M. Beshley, M. Seliuchenko, O. Panchenko, and A. Polishuk, "Adaptive flow routing model in SDN," in *2017 14th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM)*, 2017, pp. 298-302.

[7] S. CN, "A Proactive Flow Admission and Re-Routing Scheme for Load Balancing and Mitigation of

Congestion Propagation in SDN Data Plane," *International Journal of Computer Networks & Communications (IJCNC) Vol*, vol. 10, 2019.

[8] R. Wallner and R. Cannistra, "An SDN approach: quality of service using big switch's floodlight open-source controller," *Proceedings of the Asia-Pacific Advanced Network*, vol. 35, pp. 14-19, 2013.

[9] Mininet: An Instant Virtual Network on your Laptop (or other PC), available online: http://mininet.org/ ,access on april 2019.

[10] Introduction to Mininet - mininet/ mininet wiki – GitHub, available online: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet ,April 2019.

[11] K. Kaur, J. Singh, and N. S. Ghumman, "Mininet as software defined networking testing platform," in *International Conference on Communication, Computing & Systems (ICCCS)*, 2014, pp. 139-42.

[12] Iperf - The TCP/UDP bandwidth measurement tool. [Online]. Available at: http://iperf.sourceforge.net ,April 2019.

[13] L. Mazalan, S. S. S. Hamdan, N. Masudi, H. Hashim, R. A. Rahman, N. M. Tahir, *et al.*, "Throughput analysis of LAN and WAN network based on socket buffer length using JPerf," in 2*013 IEEE International Conference on Control System, Computing and Engineering*, 2013, pp. 621-625.

[14] Jperf installation https://code.google.com/archive/p/xjperf/downloads , last visited May 2019.

[15] K. Suzuki, K. Sonoda, N. Tomizawa, Y. Yakuwa, T. Uchida, Y. Higuchi, *et al.*, "A survey on OpenFlow technologies," I*EICE Transactions on Communications, v*ol. 97, pp. 375-386, 2014.

[16] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using openflow: A survey," I*EEE communications surveys & tutorials, v*ol. 16, pp. 493-512, 2014.

[17] R. Masoudi and A. Ghaffari, "Software defined networks: A survey," J*ournal of Network and Computer Applications, v*ol. 67, pp. 1-25, 2016.

[18] NetworkX, 2019. Available in: http://networkx.readthedocs.io/en/networkx-1.11/. Last visited on May 2019