International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-9 Issue-1, May 2020

Software Defined Networks using Mininet

Pramod B Patil., Kanchan S. Bhagat., D K Kirange, S. D. Patil

Abstract: ***A software defined network (SDN) is combined with centralized management by providing separation of the network, data and control planes. Different cloud computing environments, enterprise data centers and service providers are using this important feature. By implementing software defined data centers. Here in this paper we have used Mininet to demonstrate the applicability of SDN for different scalability. We study the performance of two SDN controllers – RYU and POX, that are implemented in Python using Mininet and D-ITG, Distributed Internet Traffic Generator. During this study we have used two network topologies, single and linear. The performance parameters used are maximum delay, average jitter, average bitrate. Experimental results demonstrated that the linear topology with RYU controller performs better as compared to single topology with POX controller for different network sizes.***

Keywords: POX, RYU, D-ITG, SDN

I. 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]. Large data centers require scalability. SDN is able to transform today's static networks and support large scalable networks. The virtualization is also needed to account for automated, dynamic and secure cloud environment. SDN is able to provide virtualization. [2]. It becomes a difficult job to configure and install network elements without experienced IT persons. Different simulators are required for simulating the networks consisting of interations among its elements such as switches, routers, etc. Supporting this is highly difficult to achieve. Therefore; a new network model is required to support these agility requirements.

Manuscript received on April 02, 2020.
Revised Manuscript received on April 20, 2020.
Manuscript published on May 30, 2020.

* Correspondence Author

Dr. Pramod B Patil*. Principal, Dr. Rajendra Gode Institute of Technology & Research, Amravati. Member, Executive Council, ISTE, New Delhi, India. ppamt@yahoo.com

Mr. Kanchan S. Bhagat., Assistant Professor and PG Co-Ordinator TME Society's J T Mahajan College of Engineering Faizpur ksbhagat@rediggmail.com

Dr. D K Kirange, Associate Professor TME Society's J T Mahajan College of Engineering Faizpur dkirange@gmail.com

Dr. S. D. Patil, Lecturer, Governemtn Polytechnic Jalgaon sdkirange@gmail.com

SDN is characterized by four factors namely,

A. Separation of control plane and data plane

B. A centralized view of the network with centralized controller

C. Open interfaces between the devices in the control plane and the data plane

D. the network can be programmed by the use of external application network

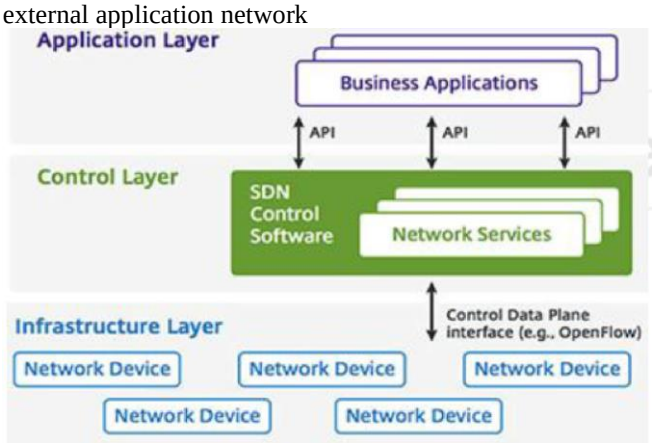


Figure 1: Architecture of SDN

In Section 2 of this paper different RYU, POX and SDN controllers are discussed. Section 3 describes the utilized simulation tool which is called Mininet, in addition, it presents MININET characteristics. In section 4, steps for Using D-ITG Traffic Generator in Mininet are discussed. Section 5 deals with the experimental setup and performance evaluation. Section 6 conclude the paper and give suggestions for future work.

II. SDN CONTROLLERS

SDN Controllers (aka SDN Controller Platforms) in a software-defined network (SDN) is known as the "brains" of the network. SDN controller is and application acting as a centralized control point in the SDN network. It also manages the flow control to the switches/ routers. The applications and business logic via northbound API for deployment of intelligent networks is also managed by controller. Recently, organizations are deploying more SDN networks, the Controllers are tasked with federating between SDN Controller domains, use of common application interfaces, including OpenFlow and open virtual switch database (OVSDB). Various networks tasks in and SDN controller platform are performed by a collection of "pluggable" modules. The tasks include investigation of the devices within networks, finding the capabilities of the devices and gathering network statistics, etc. More advanced capabilities can be supported by providing extensions with more functionality including routing algorithms to perform analytics and orchestrating new rules throughout the network.

A. POX Controller

POX controller is responsible for providing communication with SDN switches with the help of the OpenFlow or OVSDB protocol. POX can be used by the developers for creating an SDN controller with the help of Python programming language. Python is a most popular tool for developing and designing software defined networks and network applications programming. POX is the best SDN controller which can be used with the help of stock components that come bundled with it.

More complex SDN controller can also be created by creating new POX components. Different network applications can also be written that address POX's API.

Components of POX

When POX is started from the command line, POX components are invoked. These are additional Python programs. These the network functionality in the software defined network is implemented by using these components. Some stock components of the POX are already available..

The POX controller can be start with the selected stock components, by entering the following command on a terminal session connected to the Mininet VM.

~$ sudo ~/pox/pox.py forwarding.l2_pairs \

Following POX features are listed by the NOXrepo website: "Pythonic" OpenFlow interface.

- sample components for path selection, topology discovery, etc are reusable.
- POC is able to run anywhere. "It can bundle with install-free PyPy runtime for easy deployment
- Specifically targets Linux, Mac OS, and Windows.
- Topology discovery.
- The GUI and visualization tools of POX are same as NOX.
- Performance of POX application is better than NOX applications written in Python.

B. RYU Controller

Ryu is a component-based software defined networking framework. The software componnts with well defined API of RYU helps developers for creating different network management and control applications. Different protocols such as OpenFlow, Netconf, OF-config, etc. are supported by RYU for managing network devices. Ryu supports Openflow extensions fully 1.0, 1.2, 1.3, 1.4, 1.5 and Nicira Extensions. All of the code is freely available under the Apache 2.0 license.

The RYU's code is fully written in Python. It is freely available under the Apache 2.0 license and open for anyone to use. Different network management protocols supported by the Ryu Controller are NETCONF and OF-config, as well as OpenFlow. OpenFlow is one of the first and most widely deployed SDN communications standards. It also supports Nicira extensions.

III. MININET

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[3] [4] [5] .

A. Characteristics of MININET

- Flexibility: MININET is able to set new topologies and new features in software, with the help of programming languages and common operating systems.
- Applicability
- Interactivity: the simulated network must be deployed in the real network as if it happens in real networks.
- Scalability, large networks must be scaled with the help of the prototyping environment using hundreds or thousands of switches on only a computer.
- Realistic: the prototype behavior must be able for representing real time behavior with a high degree of confidence, so code modification should not be required for using applications and protocols stacks
- Shareable : The other collaborators should be able to reuse the created prototypes, which can then run and modify the experiments.

The followed steps are followed after successful insallation:

Step 1: Run following command to get access to Mininet terminal.

sudo mn

above command will start mininet terminal)

A default topology will be loaded which has two hosts that are connected to a switch (OVSSwitch) and a default controller (ovs-controller).

The terminal will start displaying something like this:

mininet>

Step 2: Use following commands inside the Mininet terminal to get information about topology:

1. For displaying nods: nodes
2. For displaying links: net
3. To dump all nodes information: dump
4. For getting more CLI commands use help.

Step 3: Information of any particular host can be obtained by the following command inside the Mninet terminal:

host-name ifconfig -a

Example: h1 ifconfig -a

Step 4: To ping from one node to another node:

host-name-1 ping host-name-2

Example: h1 ping h2

(above command will ping from h1 to h2)

Step 5: To ping all hosts from every host:

pingall

(above command will ping all hosts from every host)

Step 6: Some inbuilt topologies are are provided by Mininet like linear, single, minimal, reversed, torus and tree.

For using them, open Mininet from your bash terminal with the following command:

sudo mn --topo linear,5

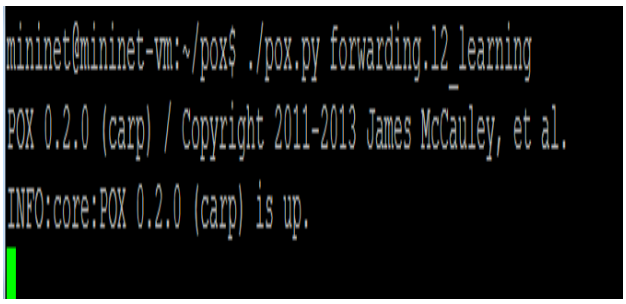
(creates a topology of 5nodes, each connected with a separate switch)

sudo mn --topo single,6

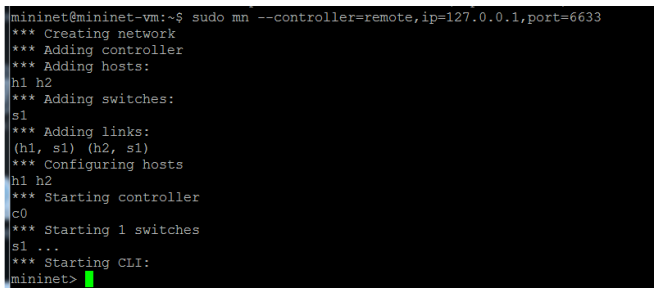
(creates a topology 6 nodes, each connected with a single switch)

IV. USING D-ITG TRAFFIC GENERATOR IN MININET

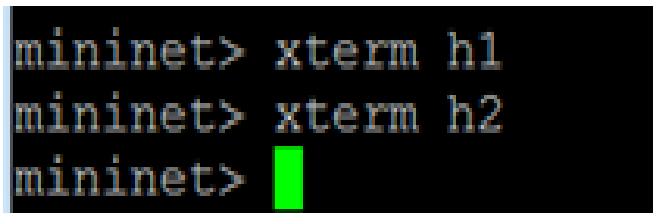
1. Login into Mininet
2. $ cd ~/pox
3. $ ./pox.py forwarding.l2_learning



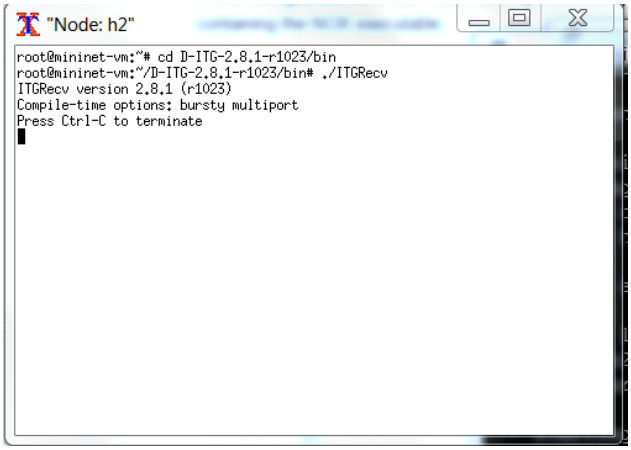
4. Now run these commands into the first PuTTY session.
5. $ cd ~
6. $ sudo mn --controller=remote,ip=127.0.0.1,port=6633



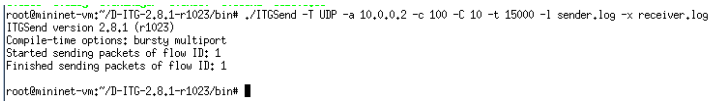
7. $ xterm h1
8. $ xterm h2



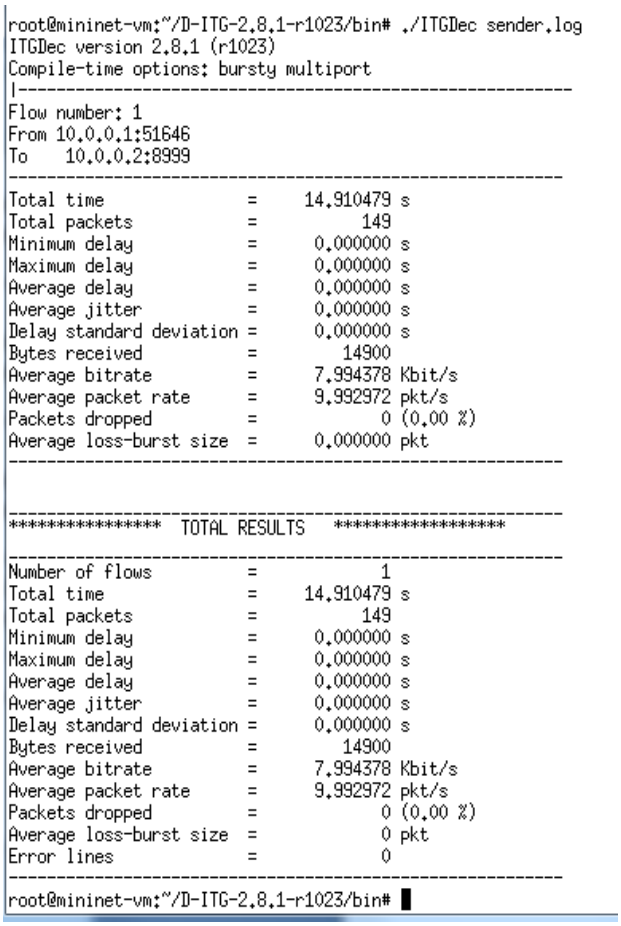
9. Now in the xterm window of h2, run these commands.
10. $ cd D-ITG-2.8.1-r1023/bin
11. $ ./ITGRecv



12. Now in the xterm of h1, run these commands.
13. $ cd D-ITG-2.8.1-r1023/bin
14. $ ./ITGSend –T UDP –a 10.0.0.2 –c 100 –C 10 –t 15000 -l sender.log –x receiver.log



15. Now to analyze the logs, run these command.
16. Run this in the xterm of h1.
17. $ ./ITGDec sender.log



18. Similarly run this on h2.
19. $ ./ITGDec receiver.log

V. EXPERIMENTING WITH SDN

To evaluate the system, we created single controller topology in mininet. We created a topology with 50,100,200,300,400 and 500 hosts and remote controllers using mininet. We are using a script to flood the capacity of the switches. After flooding the capacity, we are analyzing the behavior of the switches with respect to the controller. For filling the capacity, we sending a different number of packets from hosts to different hosts simultaneously to understand the network behavior under the load. We have evaluated the performance using POX and RYU controllers.

A. Single Topology Evaluation

For the single topology, we configured the controller in mininet with port no and IP address of the machine. The POX controller is invoked using command

"./pox.py forwarding.l2_pairs openflow.of_01 --port=6633"

in one machine to establish the connection between the switch and the controller. So, to evaluate the performance, we fill the capacity with a different number of packets. We flood switches by simultaneously sending different packets. Calculate the maximum delay, average bitrate, and load jitter by observing the flow of the packets. We observe the switch controller communication to later compare it with the other topology. Once we start the traffic bridge between the hosts and client with the help of xterm and D-ITG tool we see how the controller responds and switch handles the traffic.

After setting up the topologies and conducting the experiments we have calculated the important parameters affecting the network traffic. We calculated the maximum delay to emphasize the behavior and differentiate between single and linear topology network performance. Another parameter we calculated is the average jitter. One of the important factors for networking infrastructure is to be able to handle the load in an effective manner. As the load in any network can be unpredictable and can any time reach the maximum. So, we calculated the average jitter to highlight the reaction of the topology. We have also evaluated the performance of the network using average bitrate.

We flooding the capacity of the switches with different flows. We sent multiple packets using D-ITG tool to different hosts, observed the flows, and plotted the graph for the different parameters affecting the network performance. We analyzed the output of the D_ITG tool and plotted a graph for different values. We have evaluated the performance for POX and RYU controllers.

Below the graph in figure 2 is a maximum delay for the value collected for different packets for a single controller. X-axis represents the number of hosts for each experiment and y-axis represents the delay calculated for different packets in seconds.

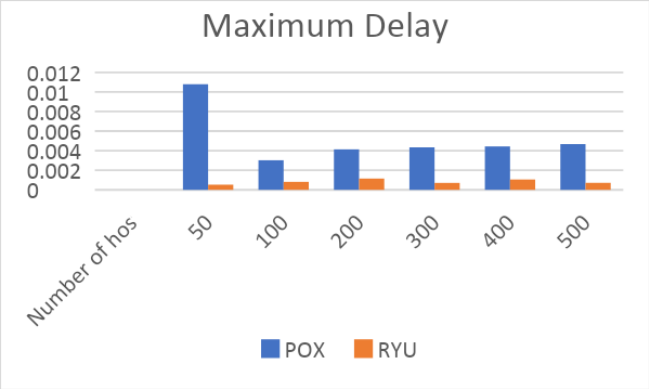


Figure 2: Maximum Delay for Single Topology

TABLE I. MAXIMUM DELAY

Number of Hosts	*POX Controller*	*RYU Controller*
50	0.0108	0.000525
100	0.003022	0.000815
200	0.004134	0.00115
300	0.00435	0.000708
400	0.004434	0.001055

The maximum delay for RYU controller is less as compared to POX controller for different number of hosts.

Below the graph in figure 3 is a average jitter for the value collected for different packets for a single controller. X-axis represents the number of hosts for each experiment and y-axis represents the average jitter calculated for different packets in seconds.

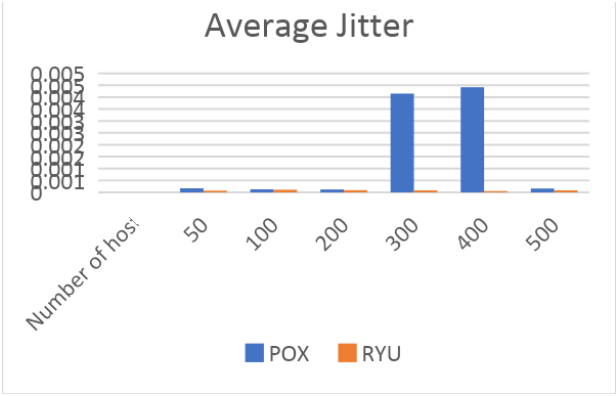


Figure 3: Average Jitter for Single Topology

Table II. Average Jitter

Number of Hosts	*POX Controller*	*RYU Controller*
50	0.000173	0.000073
100	0.00013	0.000111
200	0.000122	0.000093
300	0.004147	0.000085
400	0.004413	0.000056

As depicted in the above graph and table the average jitter is less for RYU controller as compared to POX controller for simple topology.

Below the graph in figure 4 is average bitrate for the value collected for different packets for a single controller. X-axis represents the number of hosts for each experiment and y-axis represents the average jitter calculated for different packets in seconds.

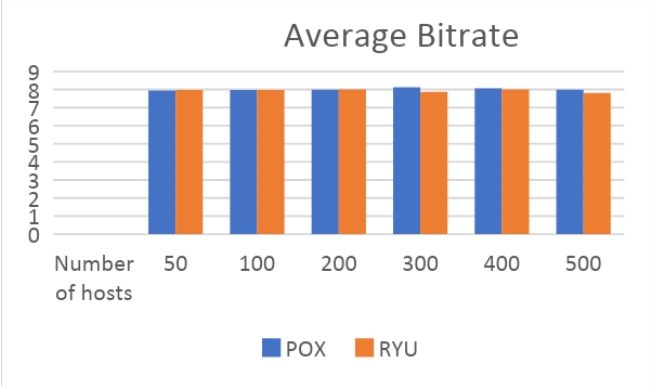


Figure 4: Average Bitrate

TABLE III. AVERAGE BITRATE

Number of Hosts	*POX Controller*	*RYU Controller*
50	7.936863	7.97688
100	7.966276	7.971505
200	7.981863	7.993634
300	8.114428	7.866923
400	8.065564	7.997105
500	7.985953	7.81005

B. Linear Topology Evaluation

For the linear topology, we configured the controller in mininet with port no and IP address of the machine. After setting up the topologies and conducting the experiments we have calculated the important parameters affecting the network traffic.

Below the graph in figure 5 is a maximum delay for the value collected for different packets for a single controller. X-axis represents the number of hosts for each experiment and y-axis represents the delay calculated for different packets in seconds.

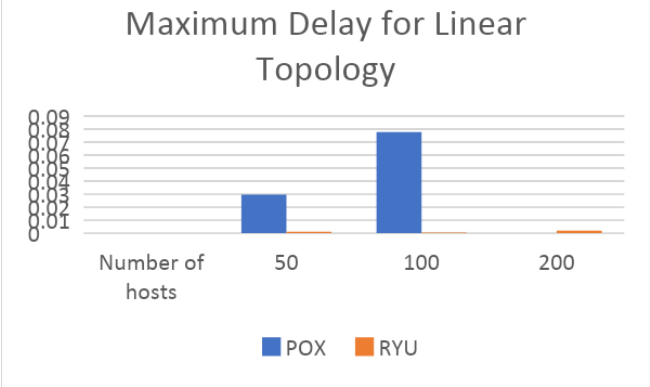


Figure 5: Maximum Delay for Linear Topology

TABLE IV. MAXIMUM DELAY FOR LINEAR TOPOLOGY

Number of Hosts	*POX Controller*	*RYU Controller*
50	0.029656	0.001138
100	0.07761	0.0005
200	0.000054	0.00198

For Linear topology also the maximum delay is less for RYU controller as compared to POX controller considering 50,100 and 200 number of hosts.

Below the graph in figure 6 is a average jitter for the value collected for different packets for a single controller. X-axis represents the number of hosts for each experiment and y-axis represents the average jitter calculated for different packets in seconds.

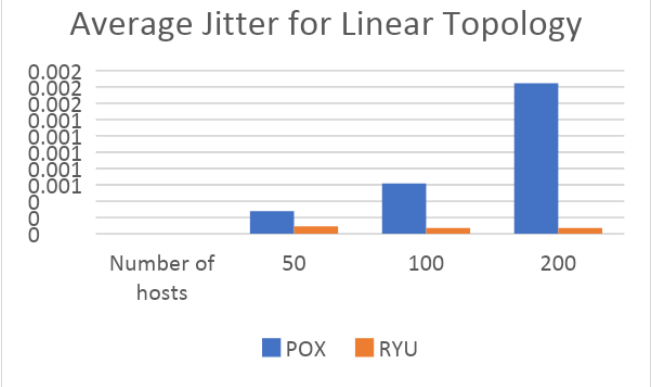


Figure 6: Average Jitter for Linear Topology

TABLE V. AVERAGE JITTER FOR LINEAR TOPOLOGY

Number of Hosts	*POX Controller*	*RYU Controller*
50	0.000279	0.000092
100	0.000618	0.000073
200	0.001844	0.000071

Average Jitter is also low for RYU controller as compared to POX controller.

Below the graph in figure 7 is average bitrate for the value collected for different packets for a single controller. X-axis represents the number of hosts for each experiment and y-axis represents the average jitter calculated for different packets in seconds.

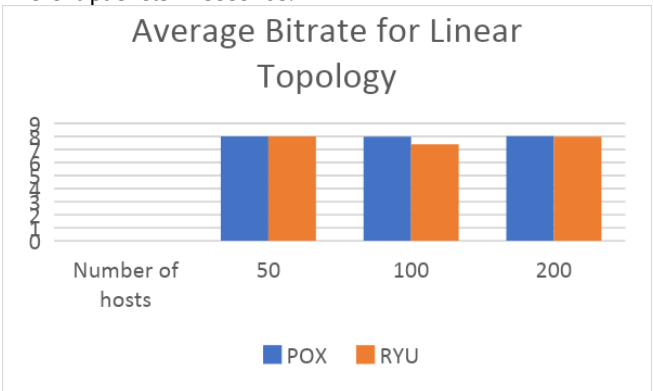


Figure 7: Average Bitrate for linear topology

Table VI. VERAGE BITRATE FOR LINEAR TOPOLOGY

Number of Hosts	*POX Controller*	*RYU Controller*
50	8.000043	7.988608
100	7.979508	7.400054
200	8.014894	7.977353

C. Comparing the Performance of Single and Linear Topology

Figure 8 shows the maximum delay performance of single and linear topology using POX controller.

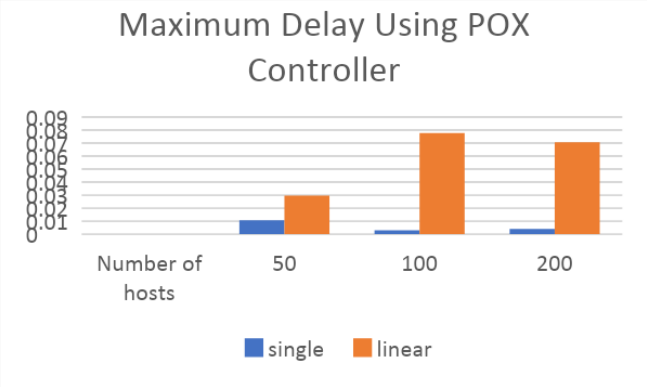


Figure 8: Maximum Delay Using POX Controller

Using POX controller the maximum delay is less for single topology.

Figure 9 shows the maximum delay performance of single and linear topology using RYU controller.

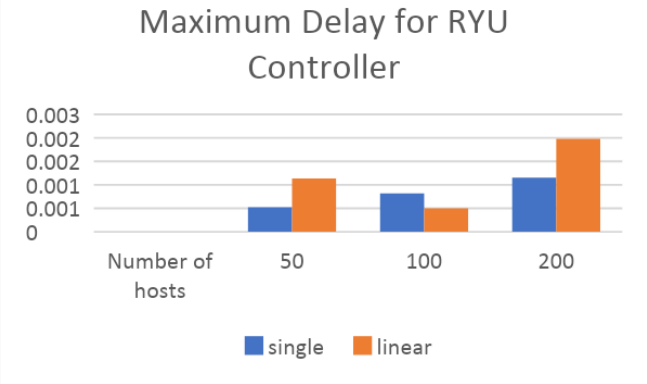


Figure 9: Maximum Delay Using RYU Controller

Using RYU controller the maximum delay is less for single topology.

Table VII depicts the performance of the single and linear topology using average jitter for POX Controller.

Table VII. AVERAGE JITTER FOR POX CONTROLLER

Number of Hosts	*Single topology*	*Linear topology*
50	0.000173	0.000279
100	0.00013	0.000618
200	0.000122	0.001844

As depicted in table VII the average Jitter is less for Single topology as compared to linear topology.

Table VIII depicts the performance of the single and linear topology using average jitter for RYU Controller.

Table VIII. AVERAGE JITTER FOR RYU CONTROLLER

Number of Hosts	*Single topology*	*Linear topology*
50	0.000073	0.000092
100	0.000111	0.000073
200	0.000093	0.000071

As depicted in table VIII the average Jitter is less for Linear topology as compared to linear topology.

VI. CONCLUSION AND FUTURE SCOPE

The Software Defined Networks are widely propagated and becoming the most promising technology. As the demand is increasing it's important to take into consideration the challenges. With the help of the results of the experiment we have concluded the following results

- As per maximum delay, the maximum delay for RYU controller is less as compared to POX controller.
- The maximum delay for single topology is less as compared to linear topology.
- The average jitter for RYU controller is low as compared to POX controller.
- The average jitter for single topology is less as compared to linear topology.
- The performance of POX and RYU controllers for single and linear topologies is also evaluated for different number of hosts.

Hence the RYU controller with linear topology is found to perform better for different network sizes

In future we will evaluate the performance of different SDN controllers. We will use different network simulation tools and performance analysis metrics.

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, "Softwaredefined networking: A comprehensive survey," Proceedings of the IEEE, vol. 103, pp. 14-76, 2015.
3. Mininet: An Instant Virtual Network on your Laptop (or other PC), available online: http://mininet.org/ ,access on april 2019.
4. Introduction to Mininet - mininet/ mininet wiki – GitHub, available online: https://github.com/mininet/mininet/wiki/Introduction-toMininet ,April 2019.
5. 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.
6. N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: a network programming language," SIGPLAN Not., vol. 46, no. 9, pp. 279–291, Sep. 2011
7. N. Sarrar, S. Uhlig, A. Feldmann, R. Sherwood, and X. Huang, "Leveraging zipf's law for traffic offloading," SIGCOMM Comput. Commun. Rev., vol. 42, no. 1, pp. 16–22, 2012.
8. Pentikousis K, Wang Y, Hu W. Mobileflow: Toward softwaredefined mobile networks[J]. IEEE Communications Magazine, 2013, 51(7):44-53.
9. Tang W, Liao Q. An SDN-Based Approach for Load Balance in Heterogeneous Radio Access Networks[C]// Computer Applications and Communications. IEEE, 2014:105-108.

10. T. Galinac Grbac, C. M. Caba, and J. Soler, “Software Defined Networking demands on software technologies,” in 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2015 - Proceedings, 2015, pp. 457–462.
11. B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, “A survey of software-defined networking: Past, present, and future of programmable networks,” IEEE Commun. Surv. Tutorials, vol. 16, no. 3, pp. 1617–1634, 2014.
12. Liu Nian-zu and Chen Xiao. Overlay multicasting at a path-level granularity for multi homed service nodes[C]. International Conference on Information Science and Technology (ICIST 2011), Nanjing, 2011: 939-946.
13. De Couto, Douglas SJ, et al. "A high-throughput path metric for multihop wireless routing." Wireless networks 11.4 (2005): 419-434.
14. A. Blenk, A. Basta, J. Zerwas, M. Reisslein, and W. Kellerer, "Control Plane Latency With SDN Network Hypervisors: The Cost of Virtualization," IEEE Transactions on Network and Service Management, vol. 13, pp. 366-380, 2016.