2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT-2019), MAY 17th & 18th 2019

Performance Analysis of Various SDN Controllers In Mininet Emulator

Arun K. Arahunashi
PG Scholar
RV College of Engineering
Bengaluru, India
arunarahunashi10@gmail.com

Neethu S.
Assistant Professor
RV College of Engineering
Bengaluru, India
neethus@rvce.edu.in

Dr. H. V. Ravish Aradhya
Professor
RV College of Engineering
Bengaluru, India
ravisharadhya@rvce.edu.in

***Abstract*— The exponential increase in the number of internet users, managing the network flow becomes a difficult task for network administrators. These limitations are overcome by using Software Defined Networking (SDN). SDN separates the control plane and data plane hence it provides the control from centralized controller to the network. Thus it offers flexible and inexpensive solutions to the limitations of the network. This paper aims to give a comparison of performances analysis of various SDN controllers like Ryu, OpenDayLight, Floodlight and ONOS controllers. To compare the performances of these controllers two QoS parameters are considered they are average delay and throughput. Performances of these controllers are analyzed for three different network topologies for different number of connections. Network topologies considered are linear, tree and mesh topologies. So the performance analysis of these controllers gives an idea of selecting a better performing controller for implementation of different application for the network.**

Keywords*: *Software Defined Networking (SDN), Controller, Ryu, Floodlight, OpenDayLight, ONOS.

I. INTRODUCTION

SDN is a new network paradigm which separates the data plane and control plane. Control plane is independent, programmable and centralized. In a traditional switch, data plane and control planes are implemented in the same device. The main functionality of data plane is packet forwarding whereas control plane is responsible for controlling the traffic flow, implementing the policies and rules for the entire network.

In SDN, data plane present in the switch where as controller makes use of "Openflow" protocol to communicate with data plane. SDN provides more flexibility to the network because it is more of software based. Policies to the network and network traffic can be controlled very easily.

The layered architecture of SDN is as shown in Fig. 1, description of these layers are given below.

1) Application layer: SDN makes use of APIs to communicate with different applications and programs are installed in this layer. Applications may be business application and network management applications.
2) Control layer: It has a central part of the architecture controller. It takes Network related information from the hardware components and provides global view of the network to the application layer, It also manages network traffic by communicating with the infrastructure layer using Openflow protocol.
3) Infrastructure layer: Hardware components required to transfer packets over the network are present in this layer for example switches, routers etc.

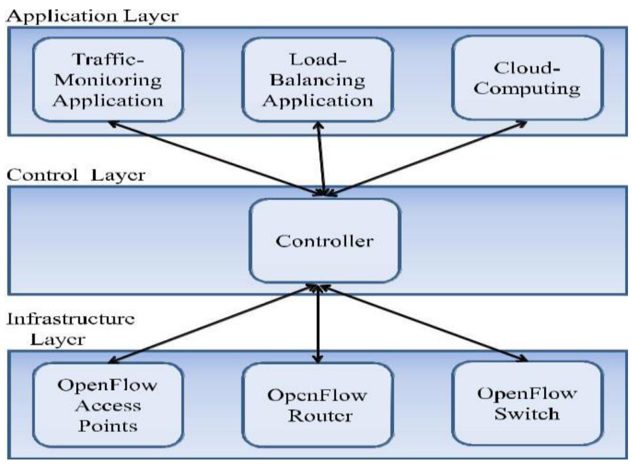


Fig.1. Three layered structure of SDN. [1]

There are many open source controllers are available. Different organizations prefer different controllers based on the requirement of an application. In this paper we have considered four open source controllers they are Ryu, OpenDayLight, Floodlight and ONOS controllers. Ryu is a python based controller where as other three controllers are JAVA based. These all four controllers supports multithreading, virtualization and TLS. All these controllers are available with REST API's. Ryu and Floodlight controllers are centralized controllers where as OpenDayLight and ONOS controllers are distributed controllers. Performances of all these controllers are analyzed using delay and throughput as QoS parameters.

The organization of this paper is as follows. Section II describes literature survey done for this paper. Section III describes methodology carried out for this paper. Section IV describes implementation. Section V shows results obtained for this paper and Section VI concludes the paper.

II. RELATED WORK

Lusani Mamushiane, Albert Lysko, Sabelo Dlamini have studied various SDN controllers like Ryu, Floodlight, ONOS and OpenDayLight and evaluated their performance in terms of latency and throughput using a tool called Cbench. Authors also presented feature based comparison of these controllers. Authors summarized the feature based comparison and suggested that ONOS and OpenDayLight controllers provides rich set of features whereas Floodlight controller provide very least amount of features and Ryu controller can be supported for small scale deployments based on its feature Coming to the performance analysis, controllers performance evaluated for two test cases with the help of Cbench tool. By tabulating the results authors concluded that in terms of latency Ryu and OpenDayLight provides best latency. In terms of throughput ONOS exhibited best throughput performance.[1]

C.Fancy, M. Pushpalatha have studied the features of POX and Floodlight controllers and compared their performances. Authors have considered Mininet emulator to evaluate the performance by varying the number of hosts for different network topologies. Authors have given a conclusion that Floodlight is more faster than POX in terms of delay as performance parameter. In terms of throughput Floodlight performs better than POX controller. [2]

PerumalrajaRengaraju, S. Senthil Kumar, Chung-Horng Lung proposed a firewall implementation for SDN using the MAC filtering. Then, authors compared the average packet delay performance for both IP filtering and MAC filtering firewall techniques. The results showed that the security support (packet allow/deny) was pretty much the same in both types of firewall technique, whereas the MAC filtering firewall outperforms the IP filtering firewall for the delay performance analysis. [3]

Pushpa. J, Dr. Pethuru Raj compared some of the SDN Controllers with their features; the controllers are POX, NOX, Floodlight, ODL, Open Contrail. The authors also compared the characteristics of centralized and distributed controller by focusing on performance, bandwidth and round-trip time of Floodlight and Open Daylight (ODL) controller . [4]

Ragaharini Jawaharan, Purnima Murali Mohan, Tamal Das and Mohan Gurusamy studied the performance of three SDN controllers — ONOS, OpenMUL and POX, that are implemented in three different programming languages, i.e., Java, C and Python, respectively, using Mininet and Wireshark packet analysis. Authors compared the performance of the controllers analyzed using Mininet-Wireshark packet analysis with that of the benchmarking tool Cbench (that uses fake control packets generated from switch instances) in terms of latency and throughput and showed that the latter significantly under-estimates the performance evaluation. Additionally, authors also studied a new performance metric, topology discovery time, that none of the current benchmarking tools capture. [5]

Yimeng Zhao, Luigi Iannone and Michel Riguidel made a reality check on the current performance achieved by mainstream open source controllers. The measurements were carried out in a controlled environment, where each controller was tested with its own optimized configuration, on the one hand allowing measuring peak performance, while, on the other hand allowing obtaining fair and reproducible results. Furthermore, besides optimizing the controller configuration, the system wide settings have been also tuned so as to maximize performance. From this comprehensive evaluation, advice on selecting and deploying controllers in real scenarios was derived. [6]

Manish Paliwal, Deepti Shrimankar, And Omprakash Tembhurne proposed a review on controllers in SDN. Authors discussed the centralized decision making capability of controllers and discussed different categories of controllers like centralized controllers and distributed controllers. Authors also evaluated controllers performances in terms of response time and throughput. According to authors, centralized controllers provides simplified architecture and efficient request handling but it fails in scalability. Distributed controllers performs better at scalability but require proper procedure while exchanging the message in the cluster. [7]

III. METHODOLOGY

This section discusses methodology for performance evaluation of controllers. Controller which performs better that can be chosen after analysing the performances of all controllers in terms of delay and throughput.

The main advantage of using virtual box for installation of controllers is virtualization. Uniform memory allocation can be done for all controllers. Controller's performance evaluation can be done by creating the different topologies in all controllers and QoS parameters are calculated by "ping" and "iperf" commands. The process is repeated for different connections by varying connections from 10 to 50 for different network topologies. Network topologies considered here are linear topology, tree topology and mesh topology. Mininet emulator has used to create network topologies.

The procedure carried out is explained as bel0w:

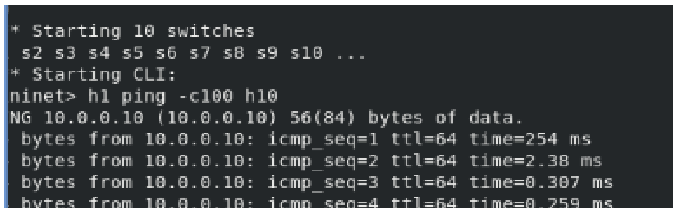
1. Installing all four controllers in virtual box.
2. Performing necessary build operation for each controller.
3. Creating different mininet topologies like linear, tree and mesh topologies for various connections from 10 to 50.
4. Calculating delay in terms of avg. RTT by sending packets from one host to another using **ping** command.
5. Calculating throughput by using **iperf** command.

IV. IMPLEMENTATION

Mininet topologies are created by using **sudo mn** command. There are some predefined topologies and some are custom topologies. In this project there topologies are considered, they are tree topology, linear topology and mesh topology. Linear and tree topologies are predefined topologies and mesh is a custom topology.

- **Linear Topology:** In linear topology a sing host connected to single switch and there is only one connection between switches.
- **Tree Topology:** Tree topology is also a predefined topology. This topology is also created by running the command. Tree topology is different from other topologies because it will get created based on the value of depth and fanout. Depth is nothing but the number of hierarchical stages and fanout is number of connections in each stage. There are five different depth and fanout combinations are considered such as [2,3], [3,3], [3,4], [4,3], [4,4]. Number of hosts created for each combination is based on the formula fanout to the power depth i.e. $\text{fanout}^{(\text{depth})}$.
- **Mesh Topology:** Mesh topology is a custom topology which is not predefined in mininet. Mesh topology can be created according to the requirement. Python programming language is used for creating the mesh topology. In mesh topology every host connect to all other hosts. Based on this fact python program has written for different connections which varies from 10 to 50.

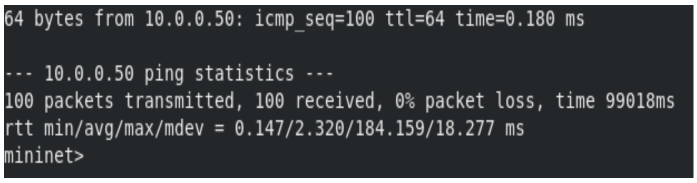
Requests can be created by using "ping" command. Ping command creates requests between two hosts by specifying source host and destination host. Number of requests transmitted is fixed to 100 requests. Fig. 2 shows how requests can be transmitted between two hosts.



```
* Starting 10 switches
 s2 s3 s4 s5 s6 s7 s8 s9 s10 ...
* Starting CLI:
ninet> h1 ping -c100 h10
NG 10.0.0.10 (10.0.0.10) 56(84) bytes of data.
 bytes from 10.0.0.10: icmp_seq=1 ttl=64 time=254 ms
 bytes from 10.0.0.10: icmp_seq=2 ttl=64 time=2.38 ms
 bytes from 10.0.0.10: icmp_seq=3 ttl=64 time=0.307 ms
 bytes from 10.0.0.10: icmp_seq=4 ttl=64 time=0.259 ms
```

Fig. 2: Creating requests using ping command

Once requests are transmitted between two hosts, ping command automatically calculates delay in terms of round trip time. It provides minimum, average, maximum and mean deviation of round trip time in terms of milliseconds. Fig. 3 shows the values obtained by this ping command. For performance analysis average round trip time is considered.



```
64 bytes from 10.0.0.50: icmp_seq=100 ttl=64 time=0.180 ms

--- 10.0.0.50 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99018ms
rtt min/avg/max/mdev = 0.147/2.320/184.159/18.277 ms
mininet>
```

Fig. 3: RTT obtained by ping command

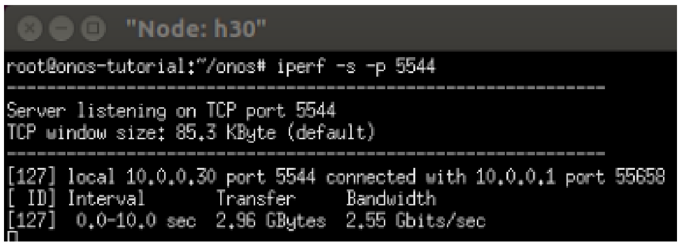
Throughput can be calculated by using iperf command. By using xterm tool, source and destination host terminals are opened. By using the below mentioned command port will be opened at destination terminal.

iperf –c –p <port_number>

From source terminal below mentioned command will be run for transmitting bandwidth.

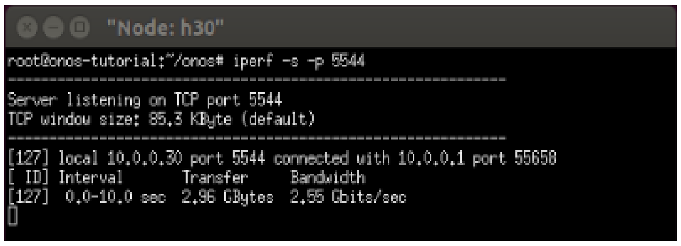
iperf –c <destination_ip_address> -p <port_number>

Source terminal provides bandwidth transmitted and destination terminal provides information about bandwidth received. Throughput is calculated by dividing the transmitted file size by time taken which is nothing but the bandwidth. So bandwidth obtained is in terms of kbps, Mbps or Gbps. In this paper throughput is considered in terms of percentage by dividing received bandwidth by transmitted bandwidth and multiplying it with 100.



```
root@onos-tutorial:~/onos# iperf -s -p 5544
------------------------------------------------------------
Server listening on TCP port 5544
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[127] local 10.0.0.30 port 5544 connected with 10.0.0.1 port 55658
[ ID] Interval       Transfer     Bandwidth
[127]  0.0-10.0 sec  2.96 GBytes  2.55 Gbits/sec
```

Fig. 4: iperf command used at source host terminal



```
root@onos-tutorial:~/onos# iperf -s -p 5544
------------------------------------------------------------
Server listening on TCP port 5544
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[127] local 10.0.0.30 port 5544 connected with 10.0.0.1 port 55658
[ ID] Interval       Transfer     Bandwidth
[127]  0.0-10.0 sec  2.96 GBytes  2.55 Gbits/sec
```

Fig. 5: iperf command used at destination host terminal.

The above mentioned steps are repeated for every network topology by varying number of connections from 10 to 50 and delay throughput details are tabulated. By comparing each controller for every connection, the controller which provides less delay than other controllers also which give better throughput in all network topology will be announced as a better performing controller.

V. RESULTS

In this paper, three network topologies such as linear, tree and mesh topologies are created using mininet emulator and performance analysis of controllers are analyzed using two QoS parameters delay and throughput. All three topologies are different in nature. Topologies are created by using **sudo mn** command by mentioning the topology name and controller ip address in the command. All four controllers are compared separately for each topology by varying the number of connections from 10 to 50. Results are shown as below.

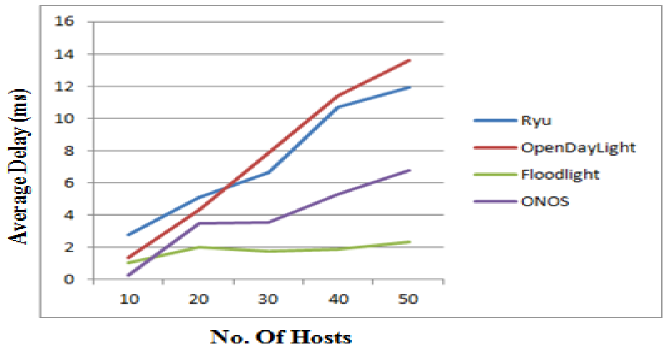


Fig. 6: Average Delay Comparison for Linear Topology for 100 Requests

Fig. 6 represents the delay comparison of four controllers in linear topology for connections from 10 to 50. Delay is measured in milli seconds for 100 requests. As we can see from the figure floodlight controller provides lesser delay than other controllers as compared to other controllers as number of connections increases.

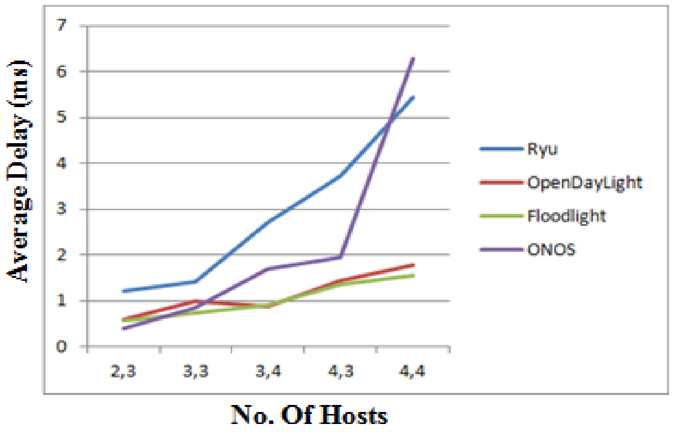


Fig. 7: Average Delay Comparison for Tree Topology for 100 Requests

Fig. 7 represents the delay comparison of four controllers in tree topology for different combinations of depth and fanout. Average Delay is measured in milli seconds for 100 requests. As we can see from the figure floodlight controller provides lesser delay than other controllers as compared to other controllers as number of connection increases.

Fig. 8 represents the delay comparison of four controllers in mesh topology for connections from 10 to 50. Average Delay is measured in milli seconds for 100 requests. As we can see from the figure floodlight controller provides lesser delay than other controllers as compared to other controllers as number of connections increases and ONOS controller getting crashed as mesh topology becomes complex.

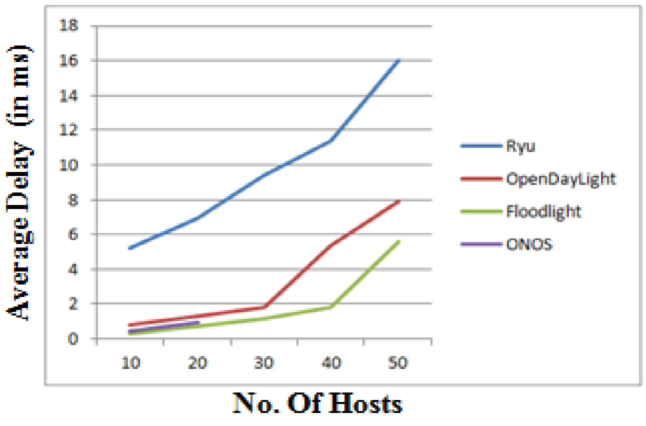


Fig. 8: Delay Comparison for Mesh Topology for 100 Requests

Fig. 9 represents the throughput comparison of four controllers in linear topology for connections from 10 to 50. Throughput is represented in percentage for 100 requests. As we can see from the figure other than OpenDayLight controller all other controllers gives 100% throughput.

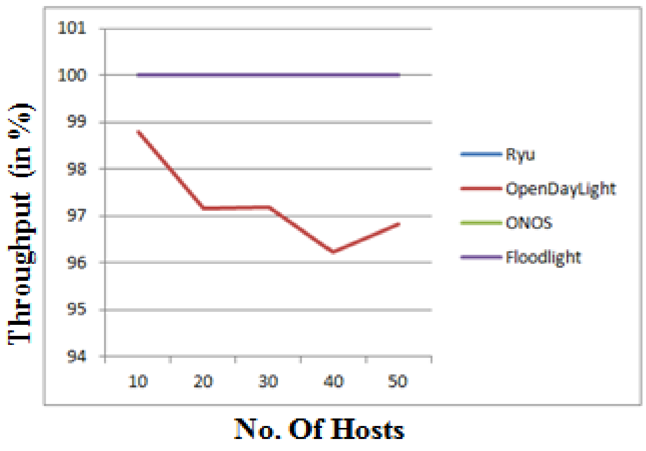


Fig. 9: Throughput (in %) Comparison for Linear Topology for 100 Requests

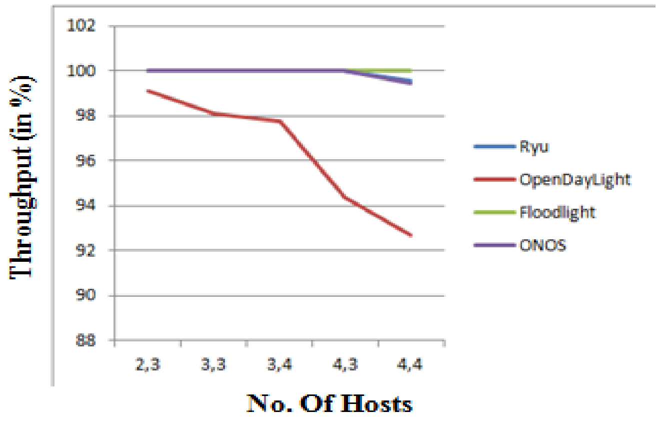


Fig. 10: Throughput (in %) Comparison for Tree Topology for 100 Requests

Fig. 10 represents the throughput comparison of four controllers in tree topology for different combinations of depth and fanout. Throughput is measured in percentage for 100 requests. As we can see from the figure, other than OpenDayLight controller all other controllers gives 100% throughput upto [4,3] connections and as number of connections becomes higher, throughput of ONOS and Ryu controllers get reduced.

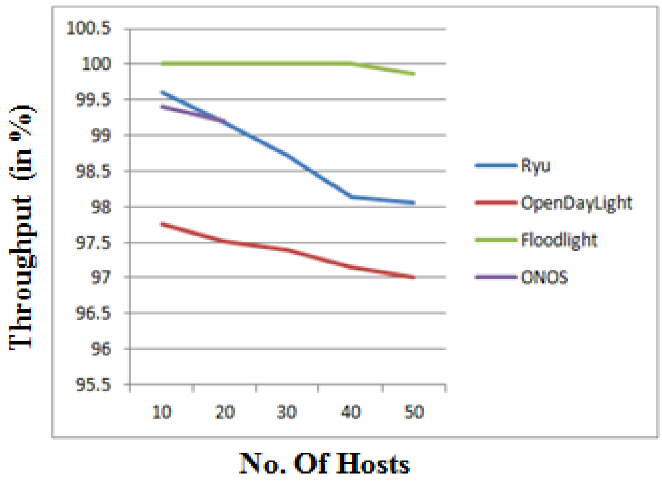


Fig. 11: Throughput (in %) Comparison for Mesh Topology for 100 Requests

Fig. 11 represents the throughput comparison of four controllers in mesh topology for connections from 10 to 50. Throughput is represented in percentage for 100 requests. As we can see from the figure Floodlight controller gives better throughput than other controllers and ONOS controller getting crashed as mesh topology becomes complex.

As results shown in all tables, it is evident that Floodlight controller gives very less delay compared to other controllers in all network topologies. Ryu and ONOS controllers performs equal to Floodlight controller in terms of throughput but their throughput performance getting reduces as network becomes complex as we can see in mesh topology. So it is evident that Floodlight gives better performance than other controllers in terms of delay and throughput. OpenDayLight controller gives poor performance than other controllers in all network topologies.

VI. CONCLUSION

In this paper, performance evaluations of four different controllers (Ryu, OpenDayLight, ONOS and Floodlight) are analyzed in mininet emulator. In order to analyze the performance, two QoS parameters were considered they are delay and throughput. To calculate QoS parameters ping and iperf commands were used. These QoS performances were analyzed for three different network topologies such as linear, tree and mesh topologies. QoS parameters are calculated for 100 requests transmitted between two hosts for all three network topologies by varying number of connections. The proposed methodology is generic and that can be applied for other different network topologies also. From the results obtained, it is evident that Floodlight controller gives better performance than other three controllers in terms of delay and throughput. By these results, Floodlight controller can be suggested as a better performing controller for any applications which are mainly dependent on QoS parameters like delay and throughput. As a future work other QoS parameters can also be analyzed for few more controllers and security is also one of the main aspect for SDN controllers and how malicious packets impacts the performance of SDN controllers can also be analyzed.

REFERENCES

[1] Lusani Mamushiane, Albert Lysko, Sabelo Dlamini "A Comparative Evaluation of the Performance of Popular SDN Controllers" *10th Wireless Days Conference (WD)*, pp.54-59, 2018.

[2] C.Fancy, M.Pushpalatha "Performance Evaluation of SDN controllers POX and Floodlight in Mininet Emulation Environment", *Proceedings of the International Conference on Intelligent Sustainable Systems*,pp.695-699, 2017..

[3] PerumalrajaRengaraju, S. Senthil Kumar, Chung-Horng Lung, "Investigation of Security and QoS on SDN Firewall Using MAC Filtering", *International Conference on Computer Communication and Informatics (ICCCI -2017)*, pp.758-764, 2017.

[4] Pushpa. J, Dr. Pethuru Raj, "Topology-Based Analysis of Performance Evaluation of Centralized Vs. Distributed SDN Controller", *19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp.1-8, 2017.

[5] Ragaharini Jawaharan, Purnima Murali Mohan, Tamal Das and Mohan Gurusamy, "Empirical Evaluation of SDN Controllers using Mininet/Wireshark and Comparison with Cbench", *International Conference on Current Trends in Advanced Computing*, pp.212-216, 2017.

[6] Yimeng Zhao, Luigi Iannone and Michel Riguidel "On the Performance of SDN Controllers: A Reality Check" *IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, pp.79-85, 2015.

[7] M. Paliwal, D. Shrimankar and O. Tembhurne, "Controllers in SDN: A Review Report," in *IEEE Access*, vol. 6, pp. 36256-36270, 2018.

[8] "Mininet", available at, http://mininet.org/

[9] "The OpenDaylight platform", available at, https://www.opendaylight.org/

[10] "Ryu installation and guide" available at https://ryu.readthedocs.io/latest/getting_started.html

[11] "ONOS installation and guide" available at https://wiki.onosproject.org/display/ONOS15/Download+packages+and+tutorial+VMs

[12] "Floodlight installation and guide" available at https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343544/Installation+Guide