Volume: 5, Issue: 6
Page: 67-80
2021

International Journal of
Science and Business
Journal homepage: ijsab.com/ijsb

IJSAB International

Performance Evaluation of Different SDN Controllers

Shavan Askar & Faris Keti

Abstract:

The unprecedented growth in internet applications' requirement makes adopting smart network architectures such as Software Defined Networking inevitable. Software-Defined Networking (SDN) is a new network architecture that separates the tight coupling between the control plane and the data plane that exists in traditional networks. The purpose of this separation is to enhance the controllability, security, and management of network resources. A controller is the core element of any SDN network, since SDN use as an alternative of traditional networks, different controllers have been developed, such as Beacon, Floodlight, RYU, OpenDay Light, ONOS, NOX, and POX. Due to the diversity of SDN applications and the use of controllers, the choice of the best-fitted controller has become an application-dependent process, therefore, this study evaluates different SDN controllers in trem of their effect on the SDN QoS performance. A comparison between the POX and RYU, which are widely used controllers, was conducted. Mininet and Miniedit was utilized as an emulation tool to compare the performance of POX and RYU controllers. For emulation purpose, Mininet, iperf3, ping, and the tested controllers POX and RYU were ran on the emulation machine that comes with the following specifications; Intel ® Core™ i5-6200 U CPU @ 2.30 GHz × 2 (2 cores) and 4 GB RAM with Ubuntu 14.04 LTS-64-bit operating system. An evaluation for both scenarios, POX and RYU, was conducted to invetiage the performance in term of QoS parameters, namely Throughput, Round-Trip Time, and Jitter using TCP, UDP, and ICMP traffic. The obtained results show that RYU controller performed better compared to POX controller, whereas, the RYU controller recorded 1.24% to 5.35% higher average throughput. In addition, RYU controller obtained 0.5 to 1 ms less delay and around 0.02 ms less jitter values than its counterpart POX controller.

IJSB
Accepted 5 April 2021
Published 7 May 2021
DOI: 10.5281/zenodo.11164550

Keywords: *software-defined networking, controllers, POX controller, RYU controller, Mininet, QoS.*

About Author (s)

Shavan Askar (Corresponding Author), Assistant Professor, Erbil Polytechnic University, CEO Arcella Telecom, Erbil, Iraq. Email: shavan.askar@epu.edu.iq.
Faris Keti, Electrical and Computer Engineering, College of Engineering, University of Duhok, Duhok, Iraq.

1. Introduction

Software-Defined Networking (SDN) is used to enhance network programmability and to simplify network management. SDN is a new network paradigm that allows the separation of the control and data plane functionalities of traditional networks, which leads to a more dynamic, flexible, automated, and manageable architecture. SDN networks have three layers: the infrastructure layer (Data Plane), the control layer (Control Plane), and the application layer (Management Plane). The control plane is a controller that acts as a central management entity. The controller makes decisions regarding network management, quality of service (QoS), and security (Askar, 2017; Fizi & Askar, 2016; Askar, 2016; Keti & Askar, 2015; Sulaiman & Askar, 2015; Fares & Askar, 2016; Mohammed, 2020).

The SDN controller communicates with switches (forwarding only devices) in the data plane using open and standardized interfaces known as south-bound interfaces and protocols such as OpenFlow. The controller controls the forwarding elements (data plane switches) to perform a wide range of functionalities, such as routing, switching, firewalling, network address translation, and load balancing (Aziz & Askar, 2021; Mondal et al, 2020).

The demand for real-time Internet services and applications has increased alongside communication system innovations and the new trends of applications that utilize the Internet and other communication systems, such as augmented reality, autonomous cars, online gaming, and Internet of Things (Loren & James, 2020; Askar et al, 2011; Ahmed & Askar, 2021; Mohammed & Askar, 2021; Aghdam, 2020; Selvaraj et al, 2020; Otoom et al, 2020). These services require special attention in terms of throughput, delay, and jitter (Abdulkahleq & Aska, 2021; Khalid & Askar, 2021). The utilization of SDN to support QoS parameters make it possible to implement new services and overcome the challenges and restrictions imposed by traditional networks. Therefore, researchers have started to tackle every single element in computer network architecture to try increase the endurance of the network to handle such highly demand applications, fog computing, the use of machine learning, and SDN are promising technologies (Samann et al, 2021, Husain & Askar, 2021; Atiqure et al, 2020; Aziz et al, 2020; Eti & Bari, 2020; Rakhmatulin, 2020). SDN networks became a promising solutions to this challenge, however, its controllers is considered the core element of any SDN network. Since different SDN controllers can results into different performance results, a careful consideration should be taken ahead of choosing an SDN controller. This research compare different SDN controllers and shows how they produce different QoS performance (Xue, 2020; Al Majeed et al, 2014) (Ali & Askar, 2021; Hamad & Askar, 2021)

There have been a variety of controller platforms developed in software by different vendors to satisfy this task. For example, NOX (Amin et al, 2018) was developed by Nicira was made open-source and donated to the research community in 2008. POX (Xia et al, 2019) has so many similar features with its predeccesore controller NOX, however POX is written with Python programming language while NOX was written with C++ programming lanaugae. Another SDN controller is called Trema (Yu, 2019) was developed by NEC in which researchers and developers can add user modules using either or C or Ruby programming languages. RYU (Baskoro et al, 2019) on the other han is an SDN controller that was created by NTT labs, it is written with Python programming language. There is also an open flow controller that was developed with Java programming language called Floodlight (Li et al, 2019), this controller was inspired from Beacon controller developed at Stanford University (Thomson & Ken, 2013).

In this paper, we provide a comparison between two well-known python based SDN controllers, specifically to gain insight into their performance (Blank & Pymoo, 2020). The

Mininet emulator (Chaurasia et al, 2019) was used to evaluate the performance of the POX and RYU controllers. To create the SDN network topology, Mini-Edit was used, which is a graphical user interface and simple network editor provided by Mininet.

The remainder of this paper is structured as follows. Section 2 presents relevant works in this area. Section 3 describes the POX controller and Mininet emulator. Section 4 describes the emulation environment and introduces results along with their analysis. Section 5 concludes the paper.

2. Related Works

SDN offers a flexible and scalable architecture. Figure 1 shows a simple architecture of an SDN platform. As shown in Figure 1, SDN networks consist of three main layers; data layer, control layer, and application\management layer. The data layer comprise of network devices such as routers, OpenFlow switches, wireless devices. The operation of these devices differs from their function at traditional networks; in SDN, they are merely forwarding devices while the intelligence unit that is responsible for making decisions is located at the controller. The case is different with traditional networks that come with network devices with their software or control unit built inside them. SDN allows network administrators of configuring and managing network's traffic which contributes into better utilization for network resources (Akhilesh et al, 2016; Hyojoon & Nick, 2013).

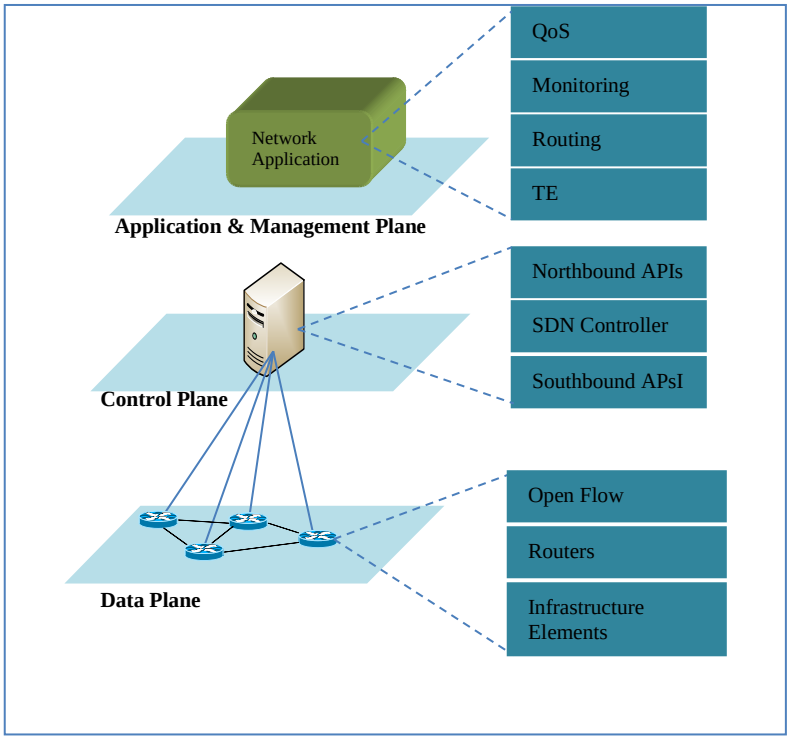


Figure 1. Software-Defined Networking (SDN) architecture (Askar, 2016).

The separation of the control plane and the data plane can be realized through a well-defined programming interface between the switches and the SDN controller (south-bound Application Programming Interface (API)) and the controller and the management plane (north-bound API). The controller exercises direct control over the state in the data plane elements via this well-defined API. The most well-known example of a south-bound interface is OpenFlow (Nick et al, 2008). An OpenFlow switch has one or more tables of packet handling rules (flow table). Each rule matches a subset of the traffic and performs certain actions (dropping, forwarding,

modifying, etc.) on the traffic (Alsaeedi et al, 2019). Depending on the definition of network policies in the management plane, the rules are installed by a controller application to instruct an OpenFlow switch to behave like a router, switch, firewall, or perform other roles (e.g., load balancer, traffic shaper, and in general those of a middlebox) (Torres et al, 2020). The novelty of SDN was strong enough to excite Google, Facebook, Microsoft, Yahoo, Verizon, and Deutsche Telekom to fund the Open Networking Foundation (ONF) (Campanella, 2020). The main goal of their funds was the promotion and adoption of SDN through open standards development (Tao et al, 2017). SDN architecture is a promising paradigm for future networks and can solve many challenges and issues that persist in current traditional networks; as such, vendors have started to develop controllers complying with ONF standards. As a result, many SDN controllers were developed, which need to be studied to compare their performance in different environments.
Table 1 presents some of the well-known SDN controllers along with their platform support and the programming language used to develop them.

Table 1. SDN controllers.

Project	Platform Support	Programming Language	Developer	Description
NOX	Linux	C ++	Nicira	First OpenFlow Controller
POX	Linux, MAC OS, and Windows	Python	Nicira	Python version of NOX, used for faster development and prototyping of new network applications.
RYU	Linux	Python	NTT, OSRG Group	Network controller with APIs for creating applications.
Beacon	Linux, MAC OS, and Windows	Java	Stanford	Java-based OpenFlow controller.
Flood Light	Linux, MAC OS, and Windows	Java	BigSwitch	Derived from Beacon.
OpenDay Light	Linux	Java	Linux foundation	All-purpose SDN controller.
ONOS (Giorgetti et al, 2020)	Linux	Java	ONF, Linux foundation	Open source SDN controller for building next-generation SDN/NFV solutions.
Trema	Linux	C, Ruby	NEC	OpenFlow Controller, Extensible to distributed controllers.

The NOX controller is considered to be the first OpenFlow controller that was written and developed in the C programming language. The other SDN controller is OpenDay Light (Sakic & Kellerer, 2018; Taehong et al, 2019), it was resulted by collaborative work of some telecommunication companies in a synergetic project that was organized by Linux Foundatoin. OpenDay Light , Flood Light , and Beacon (Sequeira et al, 2017) controllers were written and developed in Java, which requires more time to develop applications, unlike POX and RYU, which are Python-based, and thus requires less time. POX and RYU are both written in Python and both are used for the faster development and prototyping of new network applications. The node performance was evaluated through the SDN-based RYU controller only. In this paper, the POX and RYU controllers were to evaluate the QoS parameters and performance of the SDN architecture by using different methodology than the previous works. In Section 4, the obtained results of the POX controller were compared with those of the RYU controller obtained in (Sequeira et al, 2017).

3. POX, RYU, and the Mininet Emulator

POX and RYU are open-source SDN controllers. They are mostly used for speeding up development and the prototyping of new network applications, especially applications that support the OpenFlow protocol. POX is known for its ability to convert inexpensive, dumb merchant silicon devices into hubs, switches, routers, or middleboxes (e.g., firewall and load balancers) (Vladimir, 2018). Recently, computer networks experienced significant growth in terms of requirements and infrastructure improvement, which requires new strategies and tactics to investigate and evaluate new network architecture (Jiaying et al, 2019; Papavassiliou, 2020). In response, the virtual mode strategy was implemented to prototype and emulate a large scale computer network, and one of the most well-known tools for this is Mininet. Mininet has many capabilities, including the ability to emulate different kinds of network elements such as nodes, layer-2 switches, layer-3 routers, and links. It runs on a single Linux kernel and utilizes virtualization to emulate a complete network using a single system. The main characteristics of the Mininet emulator are shown in Table 2.

Table 2. Mininet characteristics.

Characteristics	Description
Flexibility	Networks implementation and features can be conducted in programming software codes on common operating systems.
Scalability	Prototyping environment must be capable of scaling up to the size of a large network.
Shareable	Created prototypes should be easily shareable with other collaborators.
Applicability	Conducted prototypes should also be usable in real networks.
Realistic	Prototype behavior should represent real-time behavior with a high degree of confidence.
Interactivity	Management and running time of the simulated network must match that of real-time as if it happens in real networks.

MiniEdit is used to create and visualize SDN topologies. Using MiniEdit, network topologies can be created visually in the Mininet emulation tool, or an existing Mininet topology (created in code) can be loaded into MiniEdit for editing. It is also possible to emulate the networks directly through the MiniEdit's GUI (Graphical User Interface), in this case Mininet is invoked as its backend for the emulations. Mininet, iperf3, ping, and the tested controller (RYU) ran on the same machine. The machine used to perform the experiments had an Intel ® Core™ i5-3210 M CPU @ 2.50 GHz × 4 (4 cores) and 4 GB RAM with Ubuntu 14.04 LTS-64-bit operating system. In our comparison, Mininet, iperf3, ping, and the tested controller (POX) ran on the same machine. The machine used to perform the experiments had an Intel ® Core™ i5-6200 U CPU @ 2.30 GHz × 2 (2 cores) and 4 GB RAM with Ubuntu 14.04 LTS-64-bit operating system. The two used machines in both cases are almost the same in terms of characteristics, except for the CPU, where the first study had 2.50 GHz with 4 cores, and our study had 2.30 GHz with 2 cores. We reevaluated the RYU controller using the same setup as the POX controller to ensure precise comparison results.

4. Experimental Tests and Results Analysis

To evaluate the performance of the SDN-based nodes through POX and RYU controllers, a Mininet virtual machine was used. Within Mininet, Mini-Edit was used to create a network topology with a single OpenFlow switch (S1), three nodes (h1, h2, h3), and a POX controller as a remote controller.

The following two instructions explain how Mini-Edit was reached and how POX and RYU were set as a remote controller for each scenario:
Setting the POX controller:

```
$ sudo python ./mininet/examples/miniedit.py
$ cd pox
Pox$ python ./pox.py forwarding.l2_learning
```

Setting the RYU controller

```
$ git clone git://github.com/osrg/RYU.git
$ cd RYU; python./setup.py install
```

The created SDN network topology by Mini-Edit is shown in Figure 2.

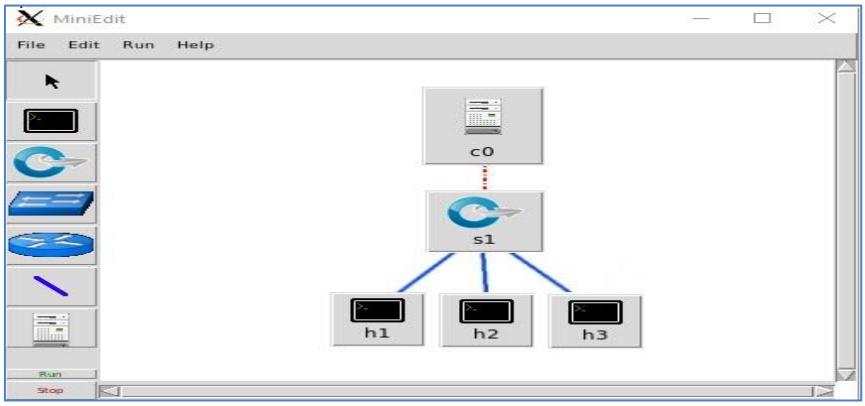


Figure 2. Created network topology with Mini-Edit.

After the implementation of network design, its performance was evaluated by calculating the main network's QoS parameters such as bandwidth, throughput, round-trip delay, and jitter. For that purpose, TCP, UDP traffic, and ICMP were used by utilizing iperf and ping benchmarking tools. The following subsections present the obtained results for each metric. The obtained results were then compared with their counterpart, using the RYU controller.

4.1. Average Peak Throughput

To examine the bandwidth performance, iperf3 benchmark utility was used to generate TCP traffic. The Iperf3 command was executed for 20 iterations of 10 seconds each and link capacity of 25Gbps to measure the TCP traffic between the nodes. Figure 3 shows the average peak throughput for both POX and RYU controllers.

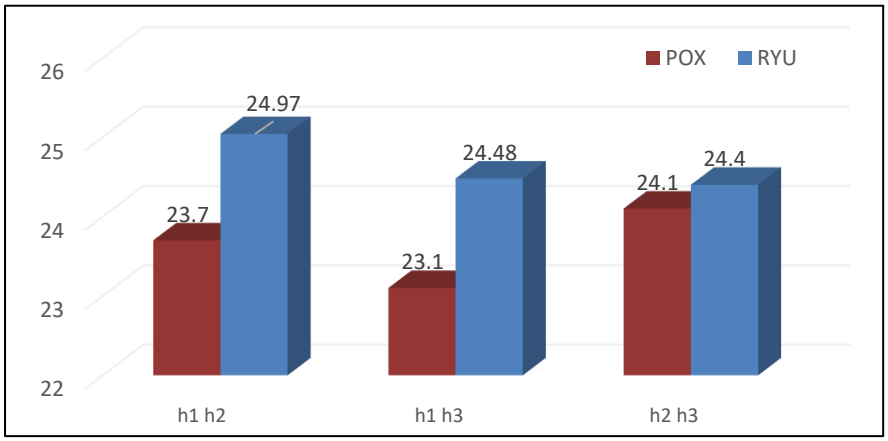


Figure 3. Average peak throughput by the RYU and POX controllers.

Figure 3 shows that for the POX controller, the maximum amount of data transferred in one second between h1 and h2 was 23.7 Gb, h1 and h3 was 23.1 Gb, and h2 and h3 was 24.1 Gb. For RYU the amount of data transferred in one second between h1 and h2 was 24.97 Gb, h1 and h3 was 24.48 Gb, and h2 and h3 was 24.4 Gb.

4.2 Throughput Metric

For throughput evaluation, the maximum amount of data that can be processed between two nodes in one second was measured. For testing purposes, a TCP node-to-node (TCP client-side to TCP server-side) connection was established and iperf3 utility was used. The result of execution was tabulated, which shows the throughput amounts for the POX controller compared to the RYU controller, and the third column (%) between each pair of SDN nodes in each line of the table indicates the throughput difference (RYU-POX\RYU) as a percentage. Figures 4 compares throughput values between h1 and h2 for POX and RYU controllers. Figures 5 and 6 shows throughput comparison between POX and RYU controllers for node pairs h1-h3 and h2-h3, respectively.

Table 3. Maximum TCP throughput for three node-to-node paths with both controllers.

Throughput (Gbps)									
Time (sec)	h1 to h2		%	h1 to h3		%	h2 to h3		%
	POX	RYU		POX	RYU		POX	RYU	
0-1	24.55	25.3	2.96	21.58	20.8	-3.75	23.33	20.9	-11.63
1-2	23.38	23.5	0.51	23.37	21.6	-8.19	24.55	26.6	7.7
2-3	22.80	22.8	0	23.01	23.0	-0.04	24.21	23.3	-3.9
3-4	22.44	21.2	-5.85	24.24	26.2	7.48	24.50	24.4	-0.4
4-5	23.38	26.4	11.43	23.08	26.8	13.88	23.36	26.2	10.83
5-6	24.55	25.7	4.47	22.01	21.6	-1.9	24.94	27.1	7.97
6-7	24.84	27.3	9.01	21.8	21.5	-1.4	24.75	28.6	13.46
7-8	22.65	23.6	4	24.52	27.0	9.18	22.65	21.8	-3.9
8-9	23.68	26.5	10.64	23.24	27.5	15.49	24.21	24.1	-0.45
9-10	24.80	27.4	9.48	24.16	28.5	15.22	23.97	22.1	-8.46

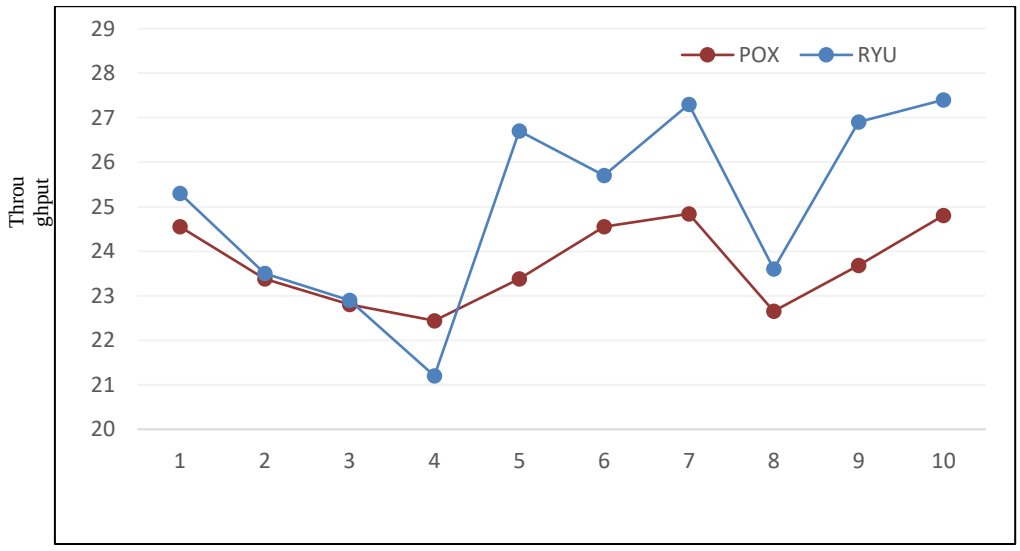


Figure 4. h1 to h2 throughput comparison between POX and RYU controllers.

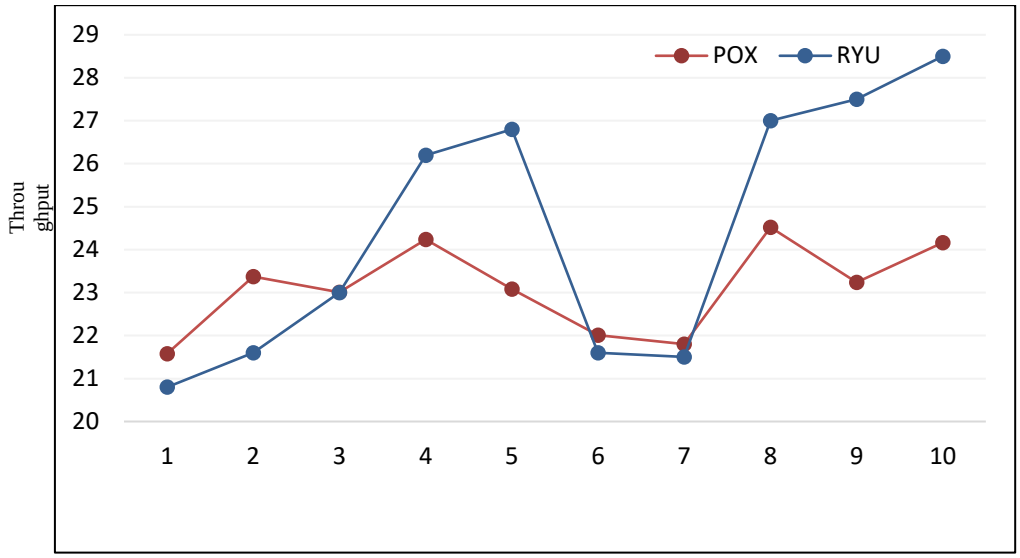


Figure 5. h1 to h3 throughput comparison between POX and RYU controllers.

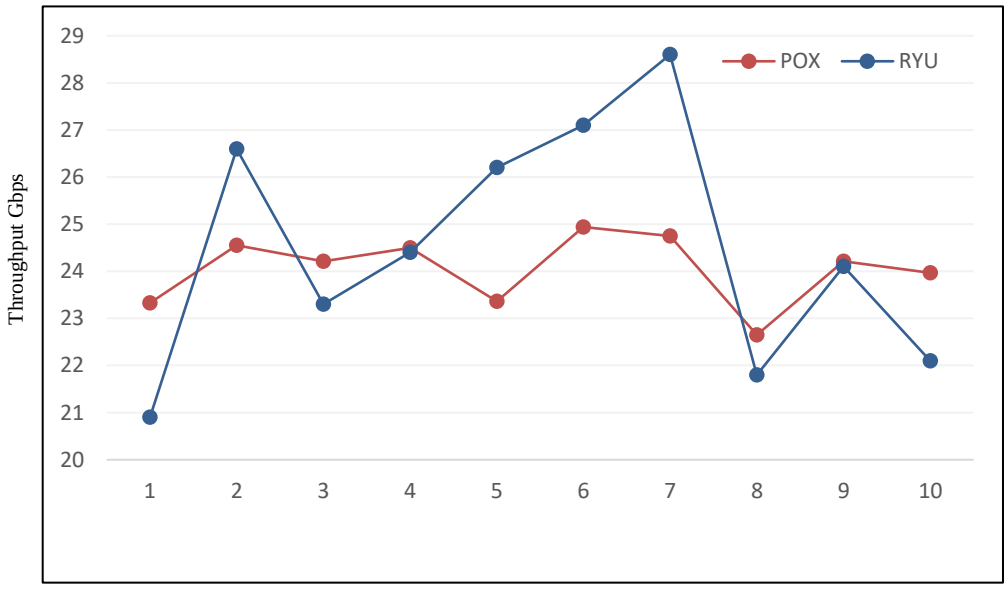


Figure 6. h2 to h3 throughput comparison between POX and RYU controllers.

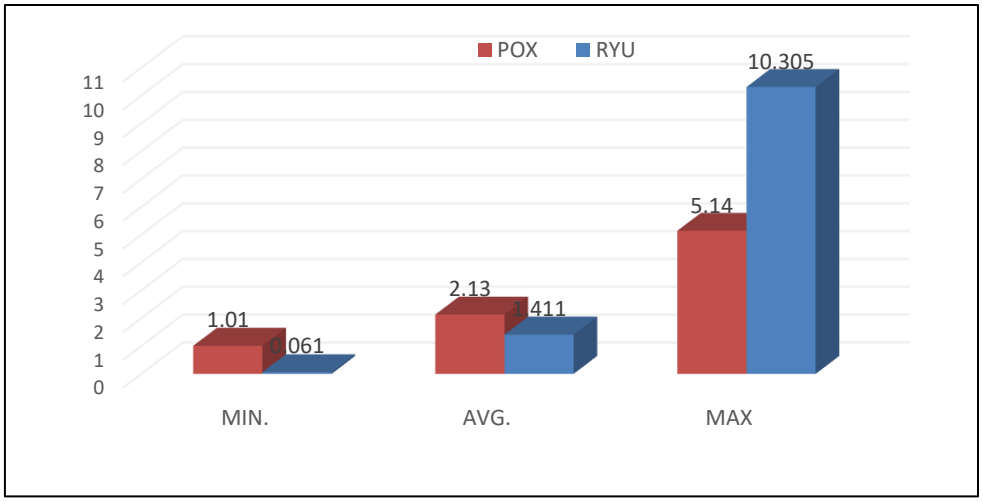
4.3 Round Trip Time (RTT) Metric

For the RTT test, a ping utility test was used, which provides message packets that report errors and other information via the ICMP protocol. RTT is defined as the time it takes for a sent packet to arrive at a specific destination and for a response to be received. The simulation was performed by executing the ping command on the client-side and sending an ICMP echo request to the server node and waiting until the echo reply packet arrives. RTT was measured for each packet between h1 and h2, h1 and h3, and h2 and h3 to determine the minimum, maximum, and average RTT values for the POX and RYU controllers.

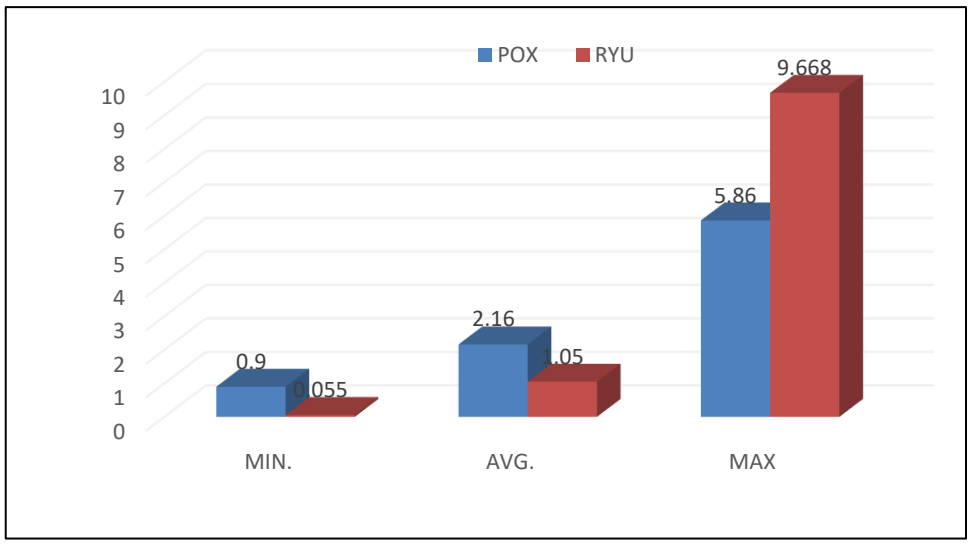
Table 4. Round Trip Time (RTT) for three node-to-node paths with both controllers.

RTT (ms)						
	h1to h2		h1to h3		h2 to h3	
	POX	RYU	POX	RYU	POX	RYU
MIN.	1.01	0.061	0.90	0.055	1.06	0.058
AVG.	2.13	1.411	2.16	1.050	2.03	0.918
MAX	5.14	10.305	5.86	9.668	5.14	8.149

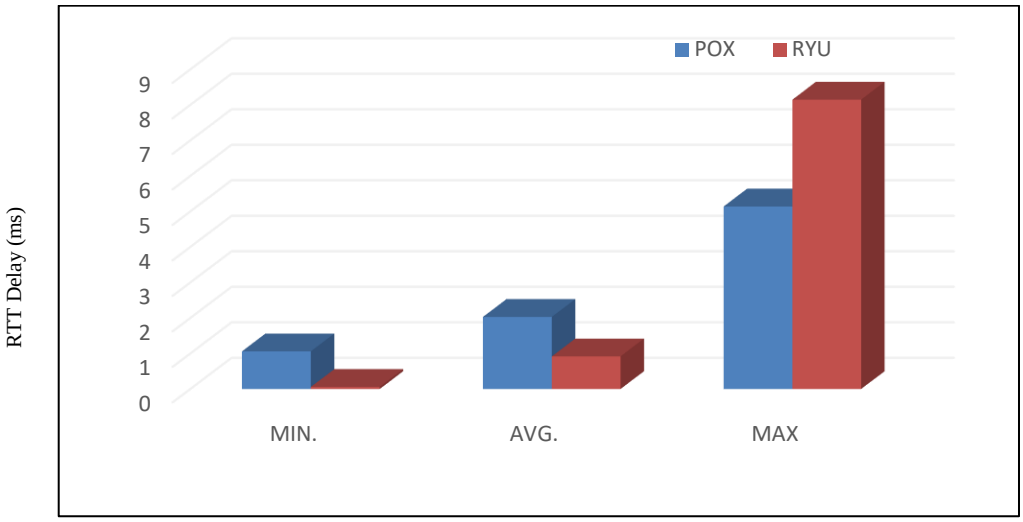
Figure 7a shows the RTT value between h1 and h2 for both POX and RYU controllers. Figure 7b,c shows the RTT comparison between POX and RYU controllers for SDN node pairs h1 and h3, and h2 and h3. On average, the RYU controller had 0.5 to 1 ms less delay when compared to the POX controller.



(a) RTT delay between h1 and h2



(b) RTT delay between h1 and h3



(c) RTT delay between h1 and h3

Figure 7. RTT delay comparison for three node-to-node paths between the POX and RYU controllers.

4.4 Jitter

Jitter is defined as the latency (response time or RTT) variation that does not necessarily depend on the latency itself. In this test, the variation in the delay of packets was evaluated for both POX and RYU controllers. This test was conducted using UDP packets and the iperf3 utility. The results of the test and a comparison with the RYU controller are shown in Table 5.

Table 5. Jitter for the three node-to-node paths with both controllers.

Jitter (ms)						
Nodes	h1 to h2		h1 to h3		h2 to h3	
Controller	POX	RYU	POX	RYU	POX	RYU
Jitter	0.036	0.016	0.042	0.014	0.035	0.014

Figure 8 shows a comparison in jitter value between different SDN node pairs for POX and RYU controllers. As is indicated in the figure, the jitter value does not change for different node pairs; however, results show higher jitter values when POX was used as a controller compared to the RYU controller.

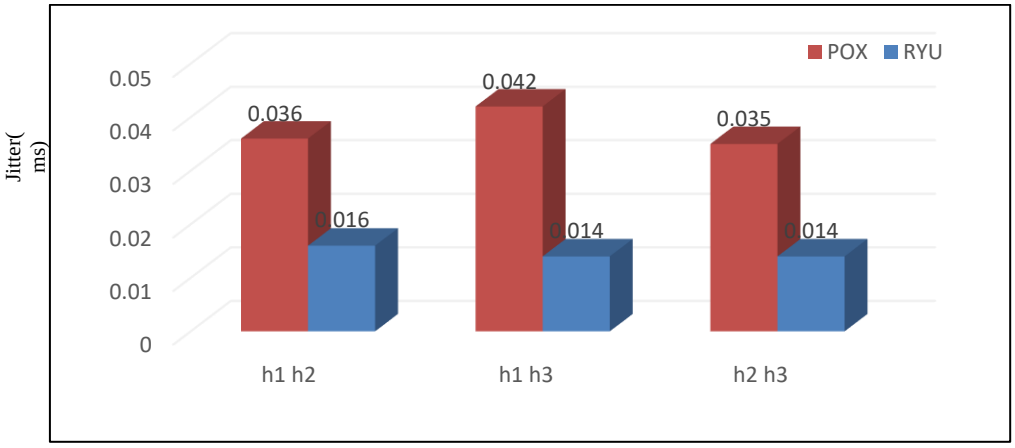


Figure 8. Jitter for three node-to-node paths with both controllers.

5. Results

For emulation purpose, Mininet, iperf3, ping, and the tested controllers POX and RYU were ran on the emulation machine that comes with the following specifications; Intel ® Core™ i5-6200 U CPU @ 2.30 GHz × 2 (2 cores) and 4 GB RAM with Ubuntu 14.04 LTS-64-bit operating system. An evaluation for both scenarios, POX and RYU, was conducted to invetiage the performance in term of QoS parameters, namely Throughput, Round-Trip Time, and Jitter using TCP, UDP, and ICMP traffic. In term of throughput, the results show that the RYU controller performed better in comparison to the POX controller; the RYU controller recorded 1.24% to 5.35% higher thoughput. Regarding the delay parameter, RYU performed better with 0.5 to 1 ms less delay, and around 0.02 ms less jitter. Therefore; the RYU controller may be the best suitable controller for applications that require higher throughput and less delay and jitter when there is no restriction on the platform supported by the controller.

6. Conclusions

Due to the diversity of SDN applications and the use of controllers, the choice of the best-fitted controller has become an application-dependent process, therefore, this study evaluates different SDN controllers in trem of their effect on the SDN QoS performance. A comparison between the POX and RYU, which are widely used controllers, was conducted. POX and RYU share similar characteristics: both are developed in Python and both are used to create and develop new SDN applications. Mininet was used as an emulation tool because of its reliable results, and different QoS parameters were taken into consideration when conducting the comparison, such as delay, jitter, and throughput. The emulation environment was implemented on a system using an Intel ® Core™ i5-6200 U CPU @ 2.30 GHz × 2 (2 cores), and 4 GB RAM with Ubuntu 14.04 LTS-64-bit operating system. Emulation results show a superiority of the RYU controller over the POX controller in terms of throughput, delay, and jitter.

The obtained results show that RYU controller performed better compared to POX controller, whereas, the RYU controller recorded 1.24% to 5.35% higher average throughput. In addition, RYU controller obtained 0.5 to 1 ms less delay and around 0.02 ms less jitter values than its counterpart POX controller. Therefore, the RYU controller may be the best fit controller when it comes to choosing a controller for applications that require high throughput and less delay and jitter values if there is no restriction on the platform they support. The reason is that although both controllers share some similarities, they have some major differences, which may impact their usage in specific circumstances. For instance, the RYU controller only supports the Linux platform, whereas the POX controller supports Linux, Windows, and Mac platforms; therefore, it always depends on the requirements of the application when it comes to choosing a suitable SDN controller. As a future work, more controllers are to be investigated to have an insight into their impact on the SDN performance.

References

Campanella A, Yan, B, Casellas, R, Giorgetti, A (2020). Reliable Optical Networks With ODTN. Resiliency and Fail-Over in Data and Control Planes, Journal of Lightwave Technology, 38, pp. 2755-2764.

Chaurasia, A, Mishra, S. N., Chinara, S. (2019). Performance Evaluation of Software-Defined Wireless Networks in IT-SDN and Mininet-WiFi, International Conference on Advances in Information Technology (India), pp. 315-319.

Giorgetti, A, Sgambelluri, A, Casellas, R, Morro, R, Campanella, A, Castoldi, P (2020). Control of open and disaggregated transport networks using the Open Network Operating System, IEEE/OSA Journal of Optical Communications and Networking, 12 A171-A181.

Mondal, A, Misra, S, Maity, I (2020). Performance Analysis of OpenFlow Systems in Software-Defined Networks. IEEE Systems Journal, 14, pp.124-131.

Aghdam, Z. N., Rahmani, A. M., Hosseinzadeh, M. J. C. M., Biomedicine, P. I. (2021). The Role of the Internet of Things in Healthcare: Future Trends and Challenges. Computer Methods and Programs in Biomedicine, 199, 105903.

Al Majeed, S., Askar, S., Fleury, M. (2014). H.265 Codec over 4G Networks for Telemedicine System Application. UKSim-AMSS 16th International Conference on Computer Modelling and Simulation (UK), Cambridge (pp. 292-297), doi: 10.1109/UKSim.2014.59.

Askar S., Zervas, G., Hunter, D. K., & Simeonidou, D. (2011). Evaluation of Classified Cloning Scheme with self-similar traffic. 3rd Computer Science and Electronic Engineering Conference (CEEC), Colchester, 2011, pp. 23-28, doi: 10.1109/CEEC.2011.5995819.

Askar, S. (2016). Adaptive Load Balancing Scheme For Data Center Networks Using Software Defined Network. Journal of University of Zakho, Vol. 4(A), No.2, Pp 275-286,

Askar, S. (2017). SDN-Based Load Balancing Scheme for Fat-Tree Data Center Networks. Al-Nahrain Journal for Engineering Sciences (NJES), Vol.20, No.5, pp.1047-1056

Askar, S., Zervas, G., Hunter, D. K., & Simeonidou, D. (2011). Service differentiation for video applications over OBS networks. 1*6th European Conference on Networks and Optical Communications*, Newcastle-Upon-Tyne, pp. 200-203.

Askar, S., Zervas, G., Hunter, D. K., & Simeonidou, D. (2011). A novel ingress node design for video streaming over optical burst switching networks. Optics Express, Vol. 19 (26), pp. 191-194

Askar, S., Zervas, G., Hunter, D. K., & Simeonidou, D. (2011). Adaptive Classified Cloning and Aggregation Technique for Delay and Loss sensitive Applications in OBS Networks. in Optical Fiber Communication Conference/National Fiber Optic Engineers Conference 2011, OSA Technical Digest (CD) (Optical Society of America, 2011), paper OThR4.

Atiqur, R., Liton, A., Wu, G. (2020). Content Caching Strategy at Small Base Station in 5G Networks with Mobile Edge Computing. International Journal of Science and Business. Vol. 4 (4). Pp:104-112.

Aziz, M. N., Islam, A. (2020). Reviewing Data Mining as an enabling technology for BI. International Journal of Science and Business. Vol. 4 (7). Pp:46-51.

Baydaa Hassan Husain & Shavan Askar (2021). Survey on Edge Computing Security. *International Journal of Science and Business, 5*(3), 52-60.

Chnar Mustafa Mohammed & Shavan Askar (2021). Machine Learning for IoT HealthCare Applications: A Review. International Journal of Science and Business, 5(3), 42-51.

Loren, E. P., James, B. M. (2020). The Promise of Interactive Shared Augmented Reality, Computer, 53, pp45- 52.

Sakic, E., Kellerer, W. (2018). Response Time and Availability Study of RAFT Consensus in Distributed SDN Control Plane. IEEE Transactions on Network and Service Management, 15, 304-318.

Torres, E., Reale, R., Sampaio, L, Martins, P. (2020). A SDN/OpenFlow Framework for Dynamic Resource Allocation based on Bandwidth Allocation Model, IEEE Latin America Transactions. 18, pp.853-860.

Eti, I., Bari, M. (2020). Digital Marketing Makes Consumer Closer: An Internet Giant Creating Challenges at Present: International Journal of Science and Business. Vol. 4 (10), Pp:64-76.

Baskoro, F., Hidayat, R., Wibowo, S. B. (2019). LACP Experiment using Multiple Flow Table in Ryu SDN Controller, International Conference on Applied Information Technology and Innovation (Indonesia), pp. 51-55.

Fares, N., Askar, S. (2016). A Novel Semi-Symmetric Encryption Algorithm for Internet Applications. Journal of University of Duhok, Vol. 19, No. 1, pp. 1-9

Fizi, F., & Askar, S. (2016). A novel load balancing algorithm for software defined network based datacenters, International Conference on Broadband Communications for Next Generation Networks and Multimedia Applications (CoBCom), Graz, 2016, pp. 1-6, doi: 10.1109/COBCOM.2016.7593506.

Glena Aziz & Shavan Askar(2021). Software Defined Network Based VANET. International Journal of Science and Business, 5(3), 83-91.

Tao, H., Richard, Y. F., Chen, Z. (2017). A Survey on Large-Scale Software Defined Networking (SDN) Testbeds: Approaches and Challenges, IEEE Communications Surveys & Tutorials. 19 891-917.

Tariqul, I., Nazrul, I. (2020). Node to Node Performance Evaluation through RYU SDN Controller, Wireless Pers. Commun, 555–570.

Ibrahim Shamal Abdulkhaleq & Shavan Askar (2021). Evaluating the Impact of Network Latency on the Safety of Blockchain Transactions. International Journal of Science and Business, 5(3), 71-82.

Blank, J., Deb, K. P. (2020). Multi-Objective Optimization in Python, IEEE Access. 8 89497-89509.

Vladimir, J., Yousef, S. (2018). Latency Measurement in an SDN Network Using a POX Controller, IEEE Canadian Conference on Electrical & Computer Engineering (Canada), Quebec City, pp. 1-5.

Xia, J., Cai, Z., Hu, G., Xu, M. (2019). An Active Defense Solution for ARP Spoofing in OpenFlow Network, Chinese Journal of Electronics. 28, 172-178.

Hyojoon, K., Nick, F. (2013). Improving network management with software defined networking, IEEE Commun. Mag. 51, 114–119.

Taehong, K., Jungho, M., Seong-Eun, Y. (2019). Load Balancing of Distributed Datastore in OpenDaylight Controller Cluster, IEEE Transactions on Network and Service Management, 16, 72-83.

Keti, F., Askar, S. (2015). Emulation of Software Defined Networks Using Mininet in Different Simulation Environments. *6th International Conference on Intelligent Systems, Modelling and Simulation*, Kuala Lumpur, 2015, pp. 205-210, doi: 10.1109/ISMS.2015.46.

Kosrat Dlshad Ahmed, Shavan Askar (2021). Deep Learning Models for Cyber Security in IoT Networks: A Review. International Journal of Science and Business, 5(3), 61-70

Kurdistan Ali & Shavan Askar (2021). Security Issues and Vulnerabilities of IoT Devices. International Journal of Science and Business, 5(3), 101-115.

Sequeira, L., de la Cruz, J. L., Ruiz-Mas, J., Saldana, J. Fernandez-Navajas, J. Almodovar (2017). Building an SDN Enterprise WLAN Based on Virtual APs, IEEE Communications Letters, 21, 374-377.

Alsaeedi, M., Mohamad, M., Al-Roubaiey, A. (2019). Toward Adaptive and Scalable OpenFlow-SDN Flow Control: A Survey, IEEE Access, 7, 107346-107379.

Nick, M., Tom, A., Hari, B. (2008). OpenFlow: Enabling innovation in campus networks, SIGCOMM Comput. Commun. Rev. 38, 69–74.

Mohammad, J. (2020). A framework synthesis by Ad-HOC based Cyber-Physical System for Performance Measure into Peak and off-Peak hours. International Journal of Science and Business, 4 (11). Pp:33-39.

Otoom, M., Otoum, N., Alzubaidi, M. A., Etoom, Y., Banihani, R. J. B. S. P., & Control. (2020). An IoT-based framework for early identification and monitoring of COVID-19 cases. 62, 102149.

Li, Q., Zou, X., Huang, Q., Zheng, J., Lee, P. C. (2019) Dynamic Packet Forwarding Verification in SDN. IEEE Transactions on Dependable and Secure Computing, 16, 915-929.

Amin, R., Reisslein, M., Shah, N. (2018) Hybrid SDN Networks . A Survey of Existing Approaches, IEEE Communications Surveys & Tutorials. 20 , 3259-3306.

Rakhmatulin, I. (2020). Review of EEG feature selection by neural networks. International Journal of Science and Business. Vol. 4 (9). Pp:101-112.

Samann, Fady E. F., Zeebaree, S. RM, Askar, S. IoT Provisioning QoS based on Cloud and Fog Computing, Journal of Applied Science and Technology Trends, Vol. 2, No. 1, pp. 29-40.

Papavassiliou, S. (2020). Software Defined Networking (SDN) and Network Function Virtualization (NFV), Future Internet. 12, 1-3.

Akhilesh, S. T., Anu, M., Michael, P. M. (2016). Software Defined Optical Networks (SDONs): A Comprehensive Survey, IEEE Communications Surveys & Tutorials, 18, 2738-2786.

Selvaraj, S., & Sundaravaradhan, S. J. S. A. S. (2020). Challenges and opportunities in IoT healthcare systems: a systematic review. 2(1), 139.

Sulaiman, S., Askar, S. (2015). Invetigation of the Impact of DDoS Attack on Network Efficiency of the University of Zakho. Journal University of Zakho, Vol. 3(A) , No.2, Pp 275-280.

Xue, X. (2020). SDN-Controlled and Orchestrated OPSquare DCN Enabling Automatic Network Slicing With Differentiated QoS Provisioning, Journal of Lightwave Technology, 38, 1103-1112.

Jiaying, Y., Zhigeng, H., Muhammad, S., Liangmin, W. (2019). Robust Security Architecture for SDN-Based 5G Networks, Future Internet. 11, 1-14.

Yu, Y. (2019) Fault Management in Software-Defined Networking: A Survey, IEEE Communications Surveys & Tutorials, 21, 349-392.

Zhala Jameel Hamad & Shavan Askar (2021). Machine Learning Powered IoT for Smart Applications. *International Journal of Science and Business, 5*(3), 92-100.

Khalid, Z. M., Askar, S. (2021). Resistant Blockchain Cryptography to Quantum Computing Attacks. *International Journal of Science and Business, 5*(3), 116-125

Cite this article:

Shavan Askar & Faris Keti (2021). Performance Evaluation of Different SDN Controllers. *International Journal of Science and Business, 5*(6), 67-80. doi: https://doi.org/10.5281/zenodo.11164550

Retrieved from http://ijsab.com/wp-content/uploads/747.pdf

Published by

IJSAB International