Node to Node Performance Evaluation through RYU SDN Controller

Article *in* Wireless Personal Communications · May 2020

DOI: 10.1007/s11277-020-07060-4

3 authors, including:

Md Tariqul Islam
University of Campinas
6 PUBLICATIONS **48** CITATIONS
SEE PROFILE

Nazrul Islam
Mawlana Bhashani Science and Technology University
48 PUBLICATIONS **259** CITATIONS
SEE PROFILE

Wireless Personal Communications
https://doi.org/10.1007/s11277-020-07060-4

Node to Node Performance Evaluation through RYU SDN Controller

Md. Tariqul Islam[1] · Nazrul Islam[1] · Md. Al Refat[1]

Abstract
Over the last few years, networks became more innovative for constructing and managing with the support of Software Defined Networking (SDN). The inflexibility of typical network architecture gives a great challenge to researchers. SDN is replacing current inflexible and complex networks with an innovative way where the control plane is decoupled from the data plane and fixing some limitations, including manual configuration, debugging, security, scalability, and mobility. Therefore, SDN controllers (e.g., NOX, POX, ONOS, OpenDaylight, Floodlight, Beacon, RYU, etc.) have emerged as a brain to manage such networks. SDN controller's performance study provides an excellent impact on enhancing the flexibilities and capabilities of an underlying SDN infrastructure network. This paper contextualizes the performance test analysis on the SDN controller is carried out through criterion (e.g., Throughput, Round-Trip Time, etc.). To implement an SDN architecture, this work uses a Mininet emulator containing a controller RYU with switching hub component, one OpenFlow switch, and three nodes. The aim is to evaluate the performance parameters such as Bandwidth, Throughput, Round-Trip Time, Jitter, and Packet loss between nodes using the RYU controller.

Keywords Software defined networking · SDN controller · RYU · Performance evaluation

1 Introduction

Nowadays, the world acts as a global village where all the different parts of the world connected by telecommunication, especially the Internet. The revolution has spread to the telecommunication and networking world like other aspects of human civilization. Very surprising to imagine that from the sixties, how far the people are moving

✉ Nazrul Islam
nazrul.islam@ieee.org

Md. Tariqul Islam
mtarislam@gmail.com

Md. Al Refat
refatict10@gmail.com

[1] Department of Information and Communication Technology, Mawlana Bhashani Science and Technology University, Tangail, Bangladesh

Published online: 20 January 2020

forward with the blessing of the Internet. The number of internet users today worldwide is more than three billion. To meet the enormous network usage demand, it is important to change the current network architecture. The tendency is to interconnect everything through the concept and implementation of cloud computing and the Internet of Things.

In the traditional network architecture, the network designed for exchanging information between end-hosts and the control and data planes combined in a network node. The control plane is responsible for the configuration of the node and programming the paths to be used for data flows. Once these paths have determined, they are pushed down to the data plane and data forwarding at the hardware level based on this control information. The traditional network system uses a distributed method to manage the network because there is no control plane abstraction of the whole network. Therefore, networks have become complicated and hard to configure and manage when something does not work. The limitations of the current network system are manual configuration, debugging, security, scalability, and mobility [1–3]. To overcome these challenges, two underlying concepts and technologies named Software Defined Networking (SDN) [1–5] and Network Function Virtualization (NFV) [6–9] has emerged. These technologies are giving us new ways to design, build, and operate networks where SDN centralizes network control in software, freeing it from proprietary hardware appliances, and NFV turns standard network functions into virtualized network functions.

Current 4G and coming 5G can provide an underlying network and the final connection between the enterprise and the customer. SDN and NFV make the best use of that underlying network by optimizing, securing, and providing other services to that network. SDN and NFV are the core of 5G. The use of SDN and NFV with 5G enables more excellent application responsiveness and better network programmability [9]. This new dimension makes SDN and NFV a key to building the modern network system. This research focuses on SDN that refers to the clear separation of control and data plane where network intelligence and state are logically centralized. In this architecture, the central control plane is maintaining the entire network.

The control plane consists of an SDN controller, which has a software-based service called network operating system (NOS) and can contain several applications. The NOS's southbound interface programs the communication between the data and the control planes through different management/control protocols such as OpenFlow, OVSDB, NETCONF, SNMP, etc. These protocols help to data plan how to forward data. Through Application Programming Interfaces (APIs), the NOS's northbound interface bare the mentioned services to the specific control applications. There are several SDN controllers such as NOX, POX, ONOS, OpenDaylight, RYU, Floodlight, Beacon, Trema, etc. Each one has different features, depending on the implemented programming language and functionality. At present most of the SDN controller based on the OpenFlow protocol because it is the most widely deployed SDN communications standard protocols to communicate between the controller and other networking devices. The RYU SDN controller presents a new SDN controller architecture, which is a component based SDN framework. It supports not only the OpenFlow protocol but also Netconf, OF-config, etc. SDN controllers deployed in large-scale networks where performance is a crucial matter. A couple of studies have performed regarding benchmarking of commonly available SDN controllers. To the best of our knowledge, no such research is available which covers the RYU SDN Controller. This paper presents the results of SDN network performance parameters such as Bandwidth, Throughput, Round-Trip Time, Jitter, and Packet loss using RYU Controller.

The rest of the paper is structured as follows. In Sect. 2, we present some background and related works. Section 3 briefly discusses the RYU SDN controller. Section 4 presents the research methodology. Section 5 shows the experimental evaluation, Sect. 6 shows the performance and result analysis, and Sect. 7 concludes the paper.

2 Background and Related Work

With the increasing interest in SDN, in recent many OpenFlow controllers have been developed and released for research and commercial use. A brief description of common OpenFlow SDN controllers listed in Table 1.

NOX [10] was the first OpenFlow controller released in early 2008. NOX firstly released in a single-threaded manner. The new version of NOX, C++ based controller, written on top of Boost library with a multithreaded learning switch application, was released in 2011. Many other OpenFlow controllers have released since NOX. Beacon a Java-based high performance and multithreaded OpenFlow controller released in early 2010. This controller can start and stop existing and new applications at runtime [11]. The floodlight is the leading open-source SDN controller. The specialty of this controller supports high availability in the way of its proprietary sibling and Big Switch's Big Network Controller via a hot standby system [12]. POX is a Python-based open-source controller for developing an SDN application that supports OpenFlow [13]. POX used for faster development and prototyping of new network applications. POX controller can be used to convert cheap, dumb merchant silicon devices into a hub, switch, router, or middleboxes such as firewall, load balancer. The drawback of POX is, it is written in Python, the performance of the network is slower than other programming languages, such as C++ or Java, and it does not work in a distributed manner. Open Network Operating System (ONOS) is another SDN controller that offers an open-source, distributed network operating system [14]. It is written in Java; thus, it requires more time when learning to develop applications for it, unlike POX. OpenDaylight (ODL) is an open-source SDN controller project implemented in Java [15]. In recent, the sixth version of the ODL SDN controller has released that supports more Internet of Things (IoT) deployments. In an earlier research study, Ref. [16] has provided the comparative performance analysis of four publicly available OpenFlow

Table 1 List of available SDN controllers

Controller name	Programming language	Developed by	License provider	Platform support
NOX	C++	Nicira Network	GPL	Most supported on Linux
POX	Python	Nicira Network	Apache	Linux, MAC OS, and Windows
Becon	Java	Stanford University	GPLv2	Linux, MAC OS and Windows
Mastero	Java	Rice University	LGPL	Linux, MAC OS and Windows
Floodlight	Java	Big Switch Network	Apache	Linux, MAC OS and Windows
Trema	C, Ruby	NEC	GPLv2	Linux only
OpenDay light	Java	Cisco and OpenDaylight	–	Linux
Ryu	Python	NTT	Apache	Most supported on Linux

controllers, including NOX, NOX-MT, Beacon, and Maestro. Afterward, a lot of comparative study and comparison among controllers has done [17, 18]. Reference [19] has evaluated and analyzed several controllers-NOX, POX, Beacon, Floodlight, Mul, etc. comparatively concerning their architecture. The analyzed indexes include performance, scalability, reliability, and security. Reference [18] shows differences between SDN controllers (including POX, RYU, ONOS, and OpenDaylight) by comparing their performances using the Mininet simulation environment. In another study, the authors claim that the Java-based Beacon controller has the best performance among several controllers (NOX, POX, Maestro, Floodlight, Ryu) because it supports multithreading, and it is a cross-platform language [20]. Whereas Python presents issues with multithreading on the performance level and C, C++ has problems with memory management. Reference [21] performed the performance analysis of ODL and found that the benchmarking of this controller is not good enough and has a memory leakage problem. The authors of Ref. [22] have shown the performance comparison performed between the Python-based POX controller and Java-based Floodlight controller over different network topologies by analyzing network throughput and round-trip delay using Mininet simulator. In an advanced study, POX, RYU, Trema, FloodLight, and OpenDaylight, these five controllers have analyzed to collect their properties like available interfaces, modularity, and productivity. Considering the requirements and the features of those topmost five controllers, the Authors concluded that that "RYU" is the best one [23]. Yet, to our best knowledge, there is no specific study only on the RYU controller that provides its performance characteristics. This paper presents the performance evaluation through the RYU controller on the Mininet environment using iperf3 [24] and ping [25] benchmarking tools.

3 RYU Overview

RYU is an open-source component-based software-defined networking framework project developed by Nippon Telegraph and Telephone (NTT). RYU is an SDN controller fully implemented in Python. RYU name comes from a Japanese word meaning "flow," which is a Japanese dragon, one of the water gods. It manages "flow" control to enable intelligent networking. The RYU Controller provides software components with well-defined application program interfaces (APIs) that make it easy for developers to create new network management and control applications. RYU supports various protocols for managing network devices, such as OpenFlow, Netconf, OF-config, etc. About OpenFlow, RYU supports fully 1.0, 1.2, 1.3, 1.4, 1.5 and Nicira Extensions [26, 27]. The architecture of the RYU SDN Controller shown in Fig. 1 [26, 27].

RYU SDN Controller has three layers. The top layer consists of business and network logic applications known as the application layer. The middle layer consists of network services known as the control layer or SDN framework, and the bottom layer consists of physical and virtual devices know as an infrastructure layer. The middle layer hosts northbound APIs and southbound APIs. The controller exposes open northbound APIs such as a Restful management API, REST, API for Quantum, User-defined API via REST or RPC, which are used by applications. RYU provides a group of components such as OpenStack Quantum, Firewall, OFREST, etc. useful for SDN applications. The objective of these applications is to gathered network intelligence by using a controller, performed analytics by running algorithms, and then orchestrated the new rules by

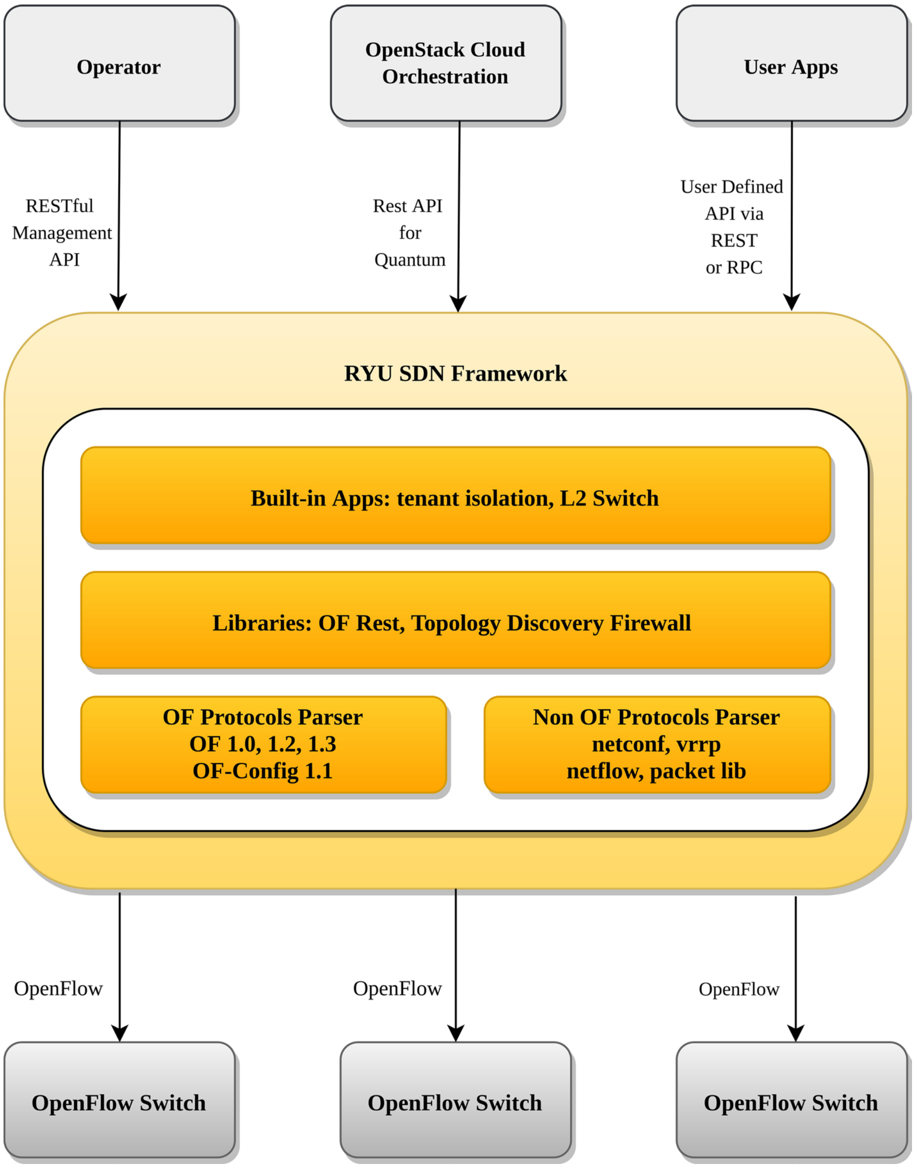

Operator
OpenStack Cloud Orchestration
User Apps
RESTful Management API
Rest API for Quantum
User Defined API via REST or RPC
RYU SDN Framework
Built-in Apps: tenant isolation, L2 Switch
Libraries: OF Rest, Topology Discovery Firewall
OF Protocols Parser OF 1.0, 1.2, 1.3 OF-Config 1.1
Non OF Protocols Parser netconf, vrrp netflow, packet lib
OpenFlow
OpenFlow
OpenFlow
OpenFlow Switch
OpenFlow Switch
OpenFlow Switch

Fig. 1 RYU SDN controller architecture

using the controller. The southbound interface is capable of supporting multiple protocols such as OpenFlow, Netconf, OF-config, etc. RYU uses OpenFlow to interact with the forwarding plane (switches and routers) to modify how the network will handle traffic flows. It has been tested and certified to work with several OpenFlow switches, including OpenvSwitch and offerings from Centec, Hewlett Packard, IBM, and NEC.

4 Methodology

Three components needed to implement SDN architecture. The first one is, SDN Applications are programs that communicate behaviors and needed resources with the SDN Controller via an application programming interface (APIs). The second one is, the SDN Controller (RYU) is a logical entity that receives instructions or requirements from the SDN Application layer and relays them to the networking components. The third one is SDN Networking Devices, control the forwarding and data processing capabilities of the network. This component includes the forwarding and processing of the data path. For a researcher, to create a physical infrastructure of SDN architecture is very costly and tough tasks. To solve this problem, one can use a specific simulator/emulator to create SDN architecture. In Software-Defined Networking, there are several tools used as a simulator such as OMNET++ [28], EstiNet [29], OFNet [30], MaxiNet [31], NS-3 [32] and Mininet [33, 34]. Among them, Mininet seems to work well. This research has performed a performance evaluation through the RYU OpenFlow controller using iperf3 and ping tools.

Mininet is developed to support research and education in SDN [33, 34]. It is a unique open-source network emulator. Mininet is very friendly to use because it uses the command-line interface. The reason behind this is that Mininet uses network namespaces as its virtualization technology; it can support many virtual nodes or hosts without interrupt or slowing down the simulation. Mininet generates a realistic virtual network, running real kernel, switch, and application code, on a single machine (VM, cloud or native). Virtual Software Defined Networking can be designed using Mininet, which consists of an SDN controller. Mininet supports several built-in basic network topologies like Minimal topology, Single topology, Linear topology, Tree topology, etc. [35]. Mininet command-line interface (CLI) is used to implement these network topologies by executing a single line command.

The minimal topology is known as default topology, which represents one OpenFlow switch connected to two hosts and the SDN OpenFlow reference controller. A default topology is designed in Mininet CLI using '**sudo mn'** command. The Single topology design with a single OpenFlow switch connected to define the number of hosts. The switch, in turn connected to the SDN OpenFlow remote controller. A single topology having 'x' (where x is a user-defined number such as 1, 2, 3, etc.) numbers of hosts are designed in Mininet CLI using **'sudo mn – topo = single, x'** command. In linear topology, each host connects with its switch, and the switches are connected linearly. That means, if there are 'n' hosts in a network, then 'n' numbers of switches are required linearly. All the OpenFlow switches, in turn, get connected with a common SND controller. A linear topology having 'y' (where y is a user-defined number such as 1, 2, 3 etc.) numbers of hosts are designed in Mininet using **'sudo mn –topo = direct, y'** CLI command. In a tree topology, all the OpenFlow switches and hosts connected in a tree fashion. A tree topology is designed in Mininet CLI using **'sudo mn –topo = tree, depth = m, fan out = n'** command. Where 'm' and 'n' is a user-defined number such as 1, 2, 3, etc. Depth represents the number of levels of switches to connected starting from the remote SDN OpenFlow controller, and fans out represent the number of output gates available to connect switches or hosts.

Iperf3 [24] is one of the most popular and powerful benchmarking tools used for network performance measurement. It measures end to end obtainable bandwidth using both User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) traffic. Iperf3 is open source software and is available on Linux, Unix, and Windows platforms. It has a dynamic server (client/server parameter exchange) functionality. In SDN, it can create data traffic to measure the throughput between the two nodes in one or both directions. In general, for each test, iperf3 report parameters as bandwidth, throughput, jitter, and packet loss.

Ping [25] is a system administrator's tool. It uses Internet Control Message Protocol (ICMP) echo request packet to a target network address at periodic intervals, and then it returns the packet to the source. It measures the round- trip time and gives the total packets sent, received, and lost, as well as the minimum, maximum, and the average time that it took to receive a response.

Mininet, iperf3, ping, and the tested controller run on the same machine. The machine used to perform the experiments has 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.

5 Experimental Evaluation

To evaluate the performance of the SDN controller, Mininet installed over Ubuntu. Then, two terminals opened, one for the Mininet and another for the RYU, for creating network topology and installing the RYU controller. Following instruction executed in the terminal for downloading RYU controller code from the RYU repository on 'GitHub.'

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

Status of available RYU manager version after the RYU installation package shown in Fig. 2.

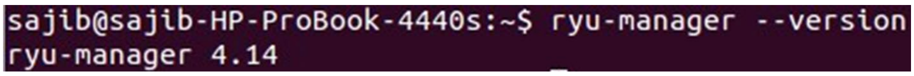
The designed network topology contains a single OpenFlow switch (S1) connected to three nodes (h1, h2, h3), and a RYU controller shown in Fig. 3. In the Mininet terminal following instruction executed for Fig. 3.

```
$ sudo mn –topo single,3 –mac –switch ovsk – controller remote –x
```

Where, the parameters **mn** start the CLI Mininet, **–topo single,3** create a topology of one switch and three nodes, **–mac** sets the MAC to address of the node automatically, **–switch ovsk** uses OpenvSwitch, **–controller remote** use external OpenFlow controller and **-x** starts xterm.

All the nodes have assigned a unique IP address and MAC address. The IP address and MAC address for node h1 are '10.0.0.1' and '00:00:00:00:00:01', for node h2 are '10.0.0.2' and '00:00:00:00:00:02' and for node h3 are '10.0.0.3' and '00:00:00:00:00:03'. After creating virtual SDN network topology, five xterm started to open on the desktop PC. Each xterm corresponds to nodes 1 to 3, the switch and the controller. The xterm window titled "switch: s1 (root)" for the switch. Following instruction executed to set 1.3 for the OpenFlow version.

```
# ovs-vsctl set Bridge s1 protocols = OpenFlow13
```



```
sajib@sajib-HP-ProBook-4440s:~$ ryu-manager --version
ryu-manager 4.14
```

Fig. 2 Status of RYU manager version

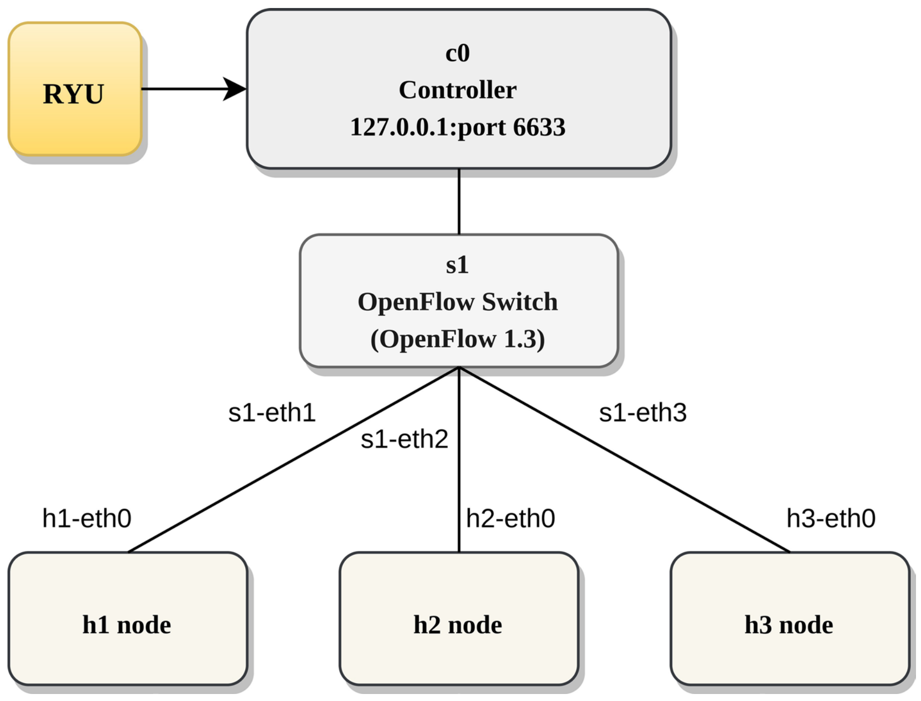


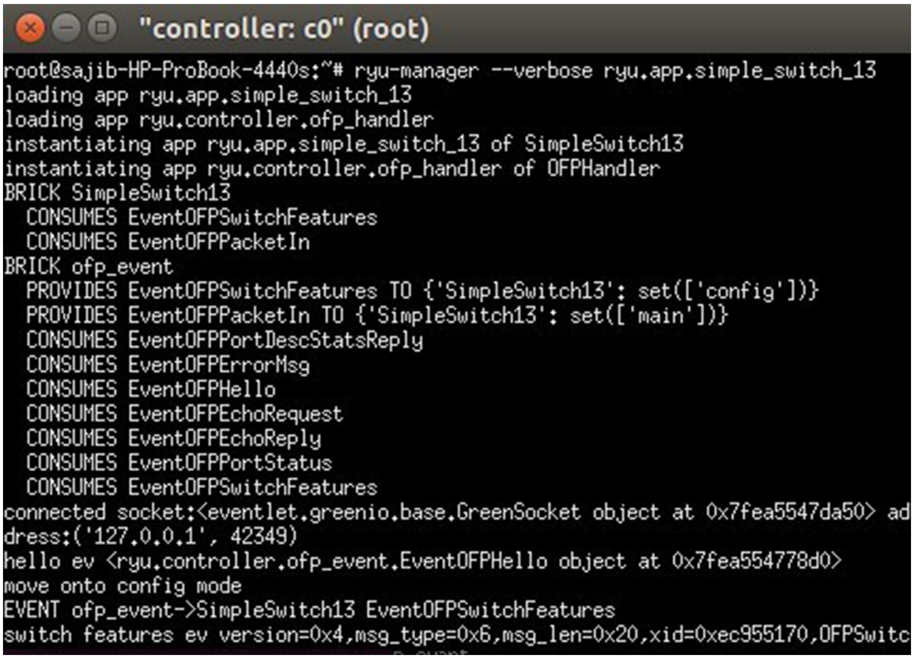
Fig. 3 Designed network topology

To implement the RYU switching hub component, the directory of RYU had accessed and executed the Simple Switch13 application which supported OpenFlow 1.3 in the xterm window titled as "controller c0 (root)". The file located in the folder/RYU/app named simpleswitch13.py It required some time to connect to OpenvSwitch (OVS). After that RYU controller able to activate on remote IP address (127.0.0.1) to host machine Mininet.

Status of a RYU controller connecting OpenFlow enabled switch of Simple Switch 13 application shown in Fig. 4. After the study of the experiment different network analysis graph has plotted, which shows the expected results.

6 Result Analysis

From the implemented network design to evaluate the performance through the RYU controller, the primary performance parameters like Bandwidth, Throughput, Round-Trip Time, Jitter, and Packet loss has assessed under TCP, ICMP, and UDP traffic using iperf3 and ping benchmarking tools. The performance assessments have executed to collect data about the SDN network with the RYU controller. The purpose of these evaluates the performance behavior through the RYU controller. The subsequent subsections show and explain the performance parameters and their results.



```
root@sajib-HP-ProBook-4440s:~# ryu-manager --verbose ryu.app.simple_switch_13
loading app ryu.app.simple_switch_13
loading app ryu.controller.ofp_handler
instantiating app ryu.app.simple_switch_13 of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK SimpleSwitch13
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPPacketIn
BRICK ofp_event
  PROVIDES EventOFPSwitchFeatures TO {'SimpleSwitch13': set(['config'])}
  PROVIDES EventOFPPacketIn TO {'SimpleSwitch13': set(['main'])}
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPErrorMsg
  CONSUMES EventOFPHello
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPEchoReply
  CONSUMES EventOFPPortStatus
  CONSUMES EventOFPSwitchFeatures
connected socket:<eventlet.greenio.base.GreenSocket object at 0x7fea5547da50> ad
dress:('127.0.0.1', 42349)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x7fea554778d0>
move onto config mode
EVENT ofp_event->SimpleSwitch13 EventOFPSwitchFeatures
switch features ev version=0x4,msg_type=0x6,msg_len=0x20,xid=0xec955170,OFPSwitc
```

Fig. 4 Status of RYU controller connecting switches of simple Switch13 application

6.1 Bandwidth

To examine the RYU controller bandwidth performance, iperf3 benchmark utility used to generates TCP traffic. TCP use processes to check that the packets correctly sent to the

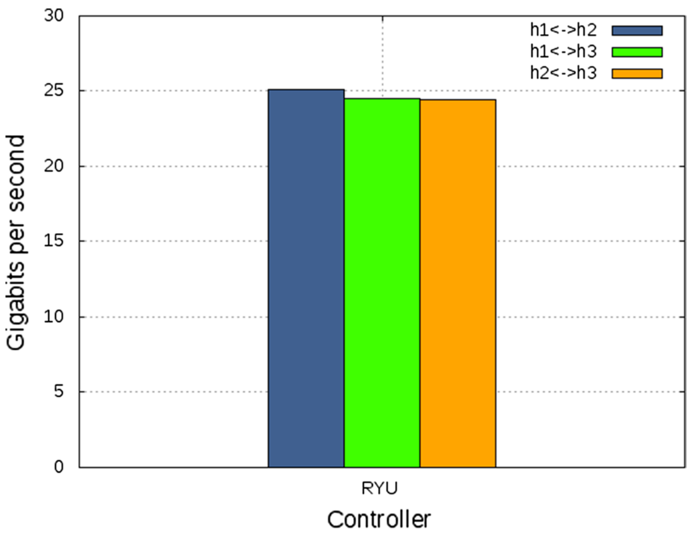


Fig. 5 Comparative TCP bandwidth between network nodes

receiver. This test conducted measuring the TCP bandwidth between the nodes from topology, executing iperf3 command for 10 s. The obtained bandwidth result is shown in Fig. 5.

Analyzing the bandwidth Fig. 5 using switch component in RYU controller the maximum amount of data transferred between h1 and h2 network node is 3,137,500 KB at a rate of 25.1 gigabits per second, between h1 and h3 network node is 3,062,500 KB at a rate of 24.5 gigabits per second and between h2 and h3 network node is 3,050,000 KB at a rate of 24.4 gigabits per second.

6.2 Throughput

In the throughput test, the controller evaluated for the maximum amount of data it can process in a second between two nodes, which measured in bits per second or data packets per second. Making a TCP node to node connection where one node act as client-side and other act as a server-side, iperf3 utility has been used to test the controller throughput performance. To measure TCP throughput, iperf3 has executed in 10 s on the client-side, and data have been collected every 1 s on the server-side. The result of execution is tabulated in Table 2 and is shown graphically in Fig. 6. The throughput graph helps in understanding end-to-end performance.

Figure 6 depicts highest and lowest throughput in gigabits approximately 27.4 gigabits and 21.2 gigabits between h1 and h2 node, 28.5 gigabits, and 20.8 gigabits between h1 and h3 node, and 28.6 gigabits and 20.9 gigabits between h2 and h3 node.

6.3 Round-Trip Time

In the third test, the controller evaluated for measuring Round-Trip Time (RTT), also known as ping time, that tells the time required to send a packet towards a specific destination and receive a response. This time delay consists of the propagation times between the two points of a signal. Ping utility use ICMP that provides message packets to report errors and other information. In this simulation, the ping command has executed on the

Table 2 TCP throughput for three node to node path

Time (s)	h1 to h2 throughput (Gbps)	h1 to h3 throughput (Gbps)	h2 to h3 throughput (Gbps)
1	25.3	20.8	20.9
2	23.5	21.6	26.6
3	22.9	23.0	23.3
4	21.2	26.2	24.4
5	26.7	26.8	26.2
6	25.7	21.6	27.1
7	27.3	21.5	28.6
8	23.6	27.0	21.8
9	26.9	27.5	24.1
10	27.4	28.5	22.1

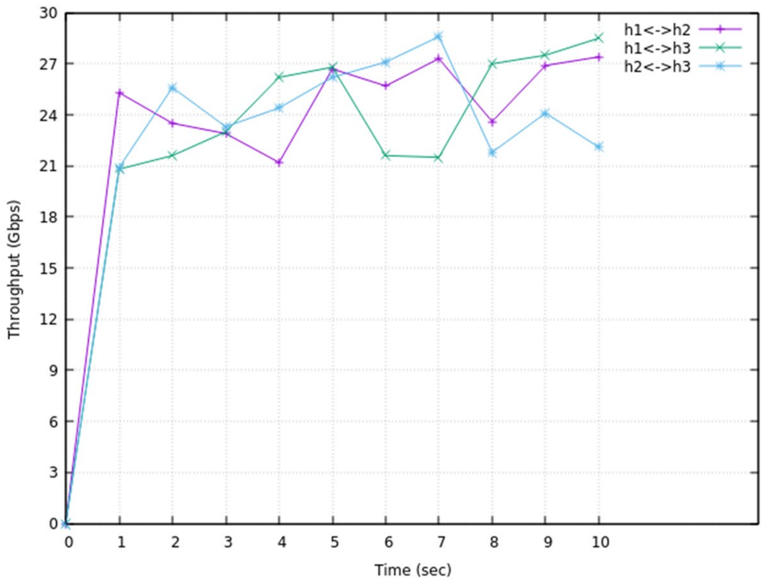


Fig. 6 TCP throughput for three node to node path

client-side to measure RTT where the client node sends an ICMP echo request to another server node and then waits for the return packet echo reply. For the ping test, ten ICMP packets have transmitted from node h1 to node h2, node h1 to node h3, and node h2 to

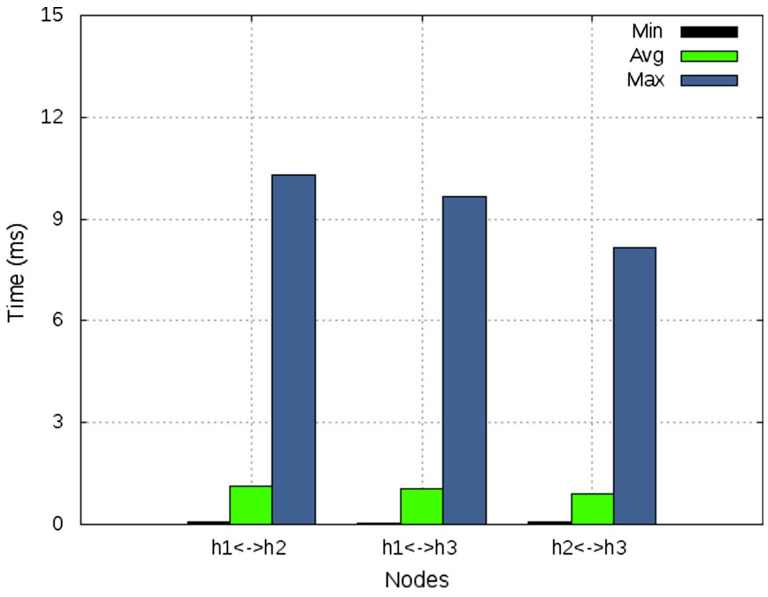


Fig. 7 RTT for three node to node path

Table 3 Jitter for three node to node path

Bandwidth (Mbits/sec)	h1 to h2 jitter (ms)	h1 to h3 jitter (ms)	h2 to h3 jitter (ms)
10	0.016	0.014	0.014
20	0.008	0.001	0.005
30	0.008	0.003	0.002
40	0.004	0.006	0.001
50	0.002	0.004	0.005

Table 4 Packet loss for three node to node path

Bandwidth (Mbits/sec)	h1 to h2 packet loss (%)	h1 to h3 packet loss (%)	h2 to h3 packet loss (%)
10	0.012	0.39	0.012
20	0.0059	0.0018	0.0059
30	0.0039	0.0078	0.0039
40	0.0059	0.0088	0.0088
50	0.0024	0.0094	0.019

node h3. RTT has measured for each packet, and the minimum, maximum, and average values determined. Three different scenarios on RTT is shown in Fig. 7.

Figure 7 depicts three different scenarios where minimum RTTs are 0.061 ms, 0.055 ms, and 0.058 ms; average RTTs are 1.411 ms, 1.050 ms, and 0.918 ms, and maximum RTTs are10.305 ms, 9.668 ms, and 8.149 ms.

6.4 Jitter and Packet Loss

In the last test, the controller evaluated for variation in the delay of packets reaching its destination known as jitter and the number of packet loss during transmission of packer from source (node) to the target (node). The jitter is the latency (response time or RTT) variation and does not depend on the latency (response time or RTT).

This test has used UDP packets where packets are sent to the receivers without any checks. To measure jitter and packet loss, iperf3 utility has been executed on five different bandwidths under UDP node to node connection. The result of executions is tabulated in Tables 3 and 4 and are shown in graphically in Figs. 8 and 9.

Figure 8 shows the jitter result for three scenarios does not take much time. The result does not vary considerably, which refers to a reliable connection. The jitter value is particularly essential on network links because a high jitter can break the connection.

Figure 9 shows the packet loss result for three scenarios does not exceed 1% packet loss. To keep a good network link quality, the packet loss should not go over 1% because a high packet loss rate will generate a lot of TCP segment retransmissions, which will affect the bandwidth.

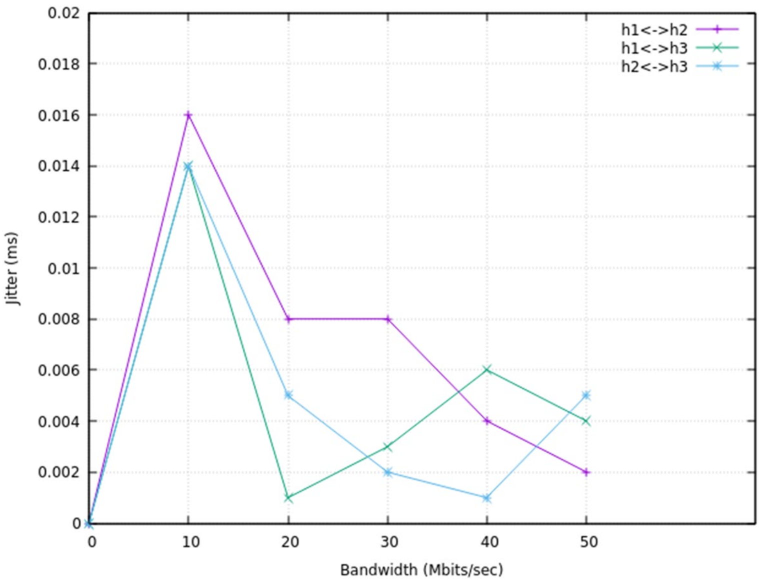


Fig. 8 Jitter for three node to node path

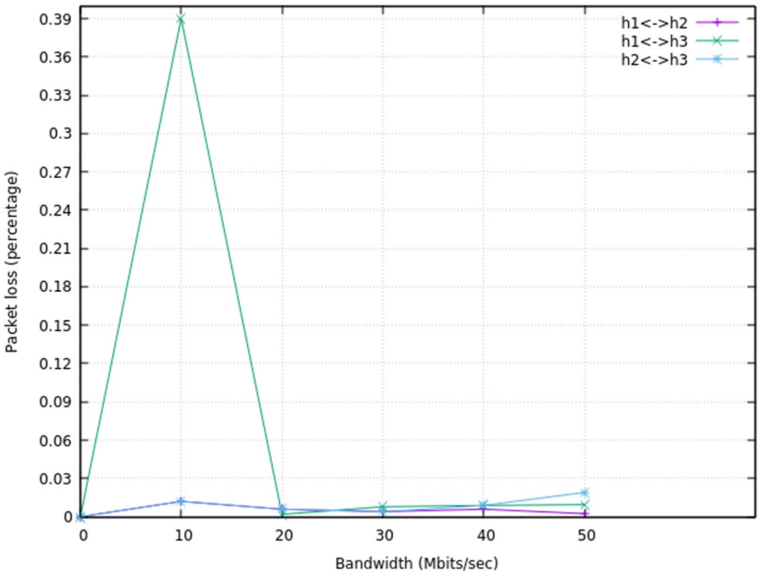


Fig. 9 Packet loss for three node to node path

7 Conclusion

This paper analyzes the performance evaluation through the RYU controller, considering one OpenFlow switch and three nodes. For three different node to node path, we obtained result after extensive simulation study, which has evaluated by bandwidth, throughput, round-trip time, jitter, and packet loss parameters.

In the future, it would be fascinating to assess the performance of RYU controllers by considering different topologies like a single, linear, tree, and custom topology in Mininet emulator and compare its performance corresponding to another OpenFlow SDN controller to choose a suitable controller for implementing the efficient control plane application. Being Python coded, RYU is an excellent choice for small business and research applications, and it presents facilities for applications and modules development.

References

1. Hakiri, A., Gokhale, A., Berthou, P., Schmidt, D. C., & Gayraud, T. (2014). Software-defined networking: Challenges and research opportunities for future internet. *Computer Networks, 75*, 453–471.
2. Masoudi, R., & Ghaffari, A. (2016). Software defined networks: A survey. *Journal of Network and Computer Applications, 67*, 1–25.
3. Kreutz, D., Ramos, F. M. V., Verıssimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE, 103*(1), 14–76.
4. Nunes, B. A. A., Mendonca, M., Nguyen, X. N., Obraczka, K., & Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials, 16*(3), 1617–1634.
5. Jammal, M., Singh, T., Shami, A., Asal, R., & Li, Y. (2014). Software defined networking: State of the art and research challenges. *Computer Networks, 72*, 74–98.
6. Mijumbi, R., Serrat, J., Gorricho, J. L., Bouten, N., Turck, F. D., & Boutaba, R. (2015). Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials, 18*(1), 236–262.
7. Han, B., Gopalakrishnan, V., Ji, L., & Lee, S. (2015). Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine, 53*(2), 90–97.
8. Yang, M., Li, Y., Jin, D., Zeng, L., Wu, X., & Vasilakos, A. V. (2015). Software-defined and virtualized future mobile and wireless networks: A Survey. *Mobile Networks and Applications, 20*(1), 4–18.
9. Akyildiz, I. F., Lin, S. C., & Wang, P. (2015). Wireless software-defined networks (W-SDNs) and network function virtualization (NFV) for 5G cellular systems: An overview and qualitative evaluation. *Computer Networks, 93*, 66–79.
10. "noxrepo/nox: The NOX Controller." [Online]. https://github.com/noxrepo/nox. Accessed 10 Oct 2018.
11. Erickson, D. (2013). The beacon openflow controller. In *Proceedings of the second ACM SIGCOMM workshop on hot topics in software defined networking* (pp. 13–18). ACM.
12. "floodlight/floodlight: Floodlight SDN OpenFlow Controller." [Online]. https://github.com/floodlight/floodlight. Accessed 12 Oct 2018.
13. "noxrepo/POX: The POX Controller." [Online]. https://github.com/noxrepo/POX. Accessed 12 Oct 2018.
14. Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., et al. (2014). ONOS: Towards an open, distributed SDN OS. In *Proceedings of the third workshop on hot topics in software defined networking* (pp. 1–6). ACM.
15. Medved, J., Varga, R., Tkacik, A., & Gray, K. (2014). OpenDaylight: Towards a model-driven SDN controller architecture. In *Proceeding of IEEE international symposium on a world of wireless, mobile and multimedia networks 2014* (pp. 1–6). IEEE.
16. Tootoonchian, A., Gorbunov, S., Ganjali, Y., Casado, M., & Sherwood, R. (2012). On controller performance in software-defined networks. In *Presented as part of the 2nd USENIX workshop on hot topics in management of internet, cloud, and enterprise networks and services*.
17. Salman, O., Elhajj, I. H., Kayssi, A., & Chehab, A. (2016). SDN controllers: A comparative study. In *2016 18th mediterranean electrotechnical conference (MELECON)* (pp. 1–6). IEEE.

18. Stancu, A. L., Halunga, S., Vulpe, A., Suciu G., Fratu, O., & Popovici, E. C. (2015). A comparison between several software defined networking controllers. In *2015 12th international conference on telecommunication in modern satellite, cable and broadcasting services (TELSIKS)* (pp. 223–226). IEEE.
19. Shalimov, A., Zuikov, D., Zimarina, D., Pashkov, V., & Smeliansky, R. (2013) Advanced study of SDN/OpenFlow controllers. In *Proceedings of the 9th central & eastern European software engineering conference in Russia* (p. 1). ACM.
20. Shah, S. A., Faiz, J., Farooq, M., Shafi, A., & Mehdi, S. A. (2013). An architectural evaluation of SDN controllers. In *2013 IEEE international conference on communications (ICC)* (pp. 3504–3508). IEEE.
21. Khattak, Z. K., Awais, M., & Iqbal, A. (2014). Performance evaluation of OpenDaylight SDN controller. In *2014 20th IEEE international conference on parallel and distributed systems (ICPADS)* (pp. 671–676). IEEE.
22. Bholebawa, I. Z., & Dalal, U. D. (2018). Performance analysis of SDN/OpenFlow controllers: POX versus floodlight. *Wireless Personal Communications, 98*(2), 1679–1699.
23. Khondoker, R., Zaalouk, A., Marx, R., & Bayarou, K. (2014). Feature based comparison and selection of software defined networking (SDN) controllers. In *2014 world congress on computer applications and information systems (WCCAIS)* (pp. 1–7). IEEE.
24. Dugan, J., Elliott, S., Mah, B. A., Poskanzer, J., & Prabhu, K. (2016). iPerf-The ultimate speed test tool for TCP, UDP and SCTP. línea]. https://iperf.fr. [Último acceso: 23 Mayo 2018].
25. "PingTool." [Online]. https://pingtool.org. Accessed 20 Oct 2018.
26. "Ryu SDN Framework." [Online]. http://osrg.github.io/ryu/. Accessed 25 Oct 2018.
27. "Ryu.pdf on WikiStart Attachment NZNOG SDN Tutorial." [Online]. https://nsrc.org/workshops/2014/nznog-SDN/attachment/wiki/WikiStart/Ryu.pdf. Accessed 25 Oct 2018.
28. "OMNeT++ Discrete Event Simulator - Home." [Online]. https://omnetpp.org/. Accessed 27 Sept 2018.
29. "EstiNet 9.0.". [Online]. Accessed 27 Sept 2018.
30. "OFNet Quick User Guide." [Online]. http://SDNinsights.org/. Accessed 28 Sept 2018.
31. "MaxiNet: Distributed Network Emulation." https://maxinet.github.io/. Accessed 10 Oct 2018.
32. "NS-3." [Online]. https://www.nsnam.org/. Accessed 11 Oct 2018.
33. Keti, F., & Askar, S. (2015). Emulation of software defined networks using mininet in different simulation environments. In *2015 6th international conference on in intelligent systems, modeling and simulation* (pp. 205–210). IEEE.
34. de Oliveira, R. L. S., Schweitzer, C. M., Shinoda, A. A., & Prete, L. R. (2014). Using Mininet for emulation and prototyping software-defined networks. In *2014 IEEE Colombian conference on communications and computing (COLCOM)* (pp. 1–6). IEEE.
35. Bholebawa, I. Z., & Dalal, U. D. (2016). Design and performance analysis of OpenFlow-enabled network topologies using mininet. *International Journal of Computer and Communication Engineering, 5*(6), 419.

Md. Tariqul Islam received the B.Sc. (Engineering) degree in Information and Communication Technology from Mawlana Bhashani Science and Technology University, Tangail, Bangladesh. His research interests are Software Defined Networking (SDN), Quality of Service (QoS), Quality of Experience (QoE), Network Function Virtualization (NFV), Internet of Things (IoT) and Multimedia Networking.

Nazrul Islam holding M.Sc. degree in Electrical Engineering with emphasis on Telecommunication Systems from Blekinge Institute of Technology, Karlskrona, Sweden. However, currently he is working as an Assistant Professor in the Department of Information and Communication Technology at Mawlana Bhashani Science and Technology University, Tangail, Bangladesh. His current research interests in the fields related to Communication Networks, Software Defined Networking and its applications, mainly modeling and analysis with respect to Quality of Service (QoS) and Quality of Experience (QoE).

Md. Al Refat received the B.Sc. (Engineering) degree in Information and Communication Technology from Mawlana Bhashani Science and Technology University, Tangail, Bangladesh.