Wireless Personal Communications (2022) 122:701–723
https://doi.org/10.1007/s11277-021-08920-3

# Performance Evaluation Using RYU SDN Controller in Software-Defined Networking Environment

**Shanu Bhardwaj**[1] · **S. N. Panda**[1]

Accepted: 8 August 2021 / Published online: 16 August 2021


**Abstract**
Software-defined networking (SDN) is a new approach that overcomes the obstacles which are faced by conventional networking architecture. The core idea of SDN is to separate the control plane from the data plane. This idea improves the network in many ways, such as efficient utilization of resources, better management of the network, reduced cost, innovation with new evolution, and many others. To manage all these changes, there is a great need for an efficient controller to improve the utilization of resources for the better performance of the network. The controller is also responsible for the analysis and monitoring of real-time data traffic. There is a great need for a high-performance controller in networking industries, data centres, academia, and research due to the tremendous growth of distributed processing-based real time applications. Therefore, it is crucial to investigate the performance of an open-source controller to provide efficient traffic routing, leading to improved utilization of resources for the enhanced performance metrics of the network. The paper presents an implementation of SDN architecture using an open-source RYU SDN controller for the network traffic analysis. The proposed work evaluates the performance of SDN architecture based custom network topology for a node to node performance parameters such as bandwidth, throughput and roundtrip time, etc. The simulation results exhibit an improved performance of the proposed work in comparison to the existing default network topology for SDN.

**Keywords** Software defined networking · Traffic network · SDN controller · RYU · Performance evaluation · Network parameters

## 1 Introduction

Nowadays, network bandwidth has become a major challenge in running various distributed computing applications by network administrators and Internet Service providers. There has been rapid growth and development of many new applications in the domain of Internet of Things (IoT), cloud computing which requires distributed processing in the

✉ Shanu Bhardwaj
shanubhardwaj1@gmail.com

[1] Chitkara University Institute of Engineering and Technology, Chitkara University, Rajpura, Punjab, India

network [1, 2]. Thus, the conventional network architecture is not sufficient to meet the requirement of these applications since a large amount of network capacity is occupied [2]. Most of the data traffic in the network is transmitted via transmission control protocol (TCP). TCP has versatile features such as congestion control, improves bandwidth, and provides reliability to the network during communication [3]. To meet the immense demand for network usage, it is essential to update the conventional architecture of the network [4].

In the conventional architecture, the problem is that both the data plane and control plane are integrated into the same appliance [5]. The design of the conventional network architecture for the data traffic exchange is combined in the network node in the form of a data plane, control plane, and end-to-end hosts [6]. There is no rule of abstraction for the control plane in the whole network. Hence, the conventional architecture cannot provide the global perspective of the network, and each device requires manual configuration [7]. This is the major reason due to which the conventional network has become convoluted and difficult to configure manually [8]. Furthermore, it has led to various constraints such as low scalability, security, manual configuration, reliability, and so on [9]. Therefore, a new paradigm is designed by researchers to prevail over the conventional network architecture is named as software-defined networking [10]. The new approach provides enhanced flexibility to build or configure the network easily and a more programmable distinction of the control plane from the data plane with a global perspective of the network [11]. The control plane is also termed the centralized plane because it maintains the whole network with the help of SDN controllers and also makes the network agile for the efficient usage of the data [12].

SDN controller relies on the control plane of the SDN environment. These controllers consist of network operating system (NOS), which provides software-based services and several network applications [13]. The communication between the control plane and the data plane takes place with the aid of a southbound application programming interface (API). The communication between the control plane and management plane is done with the northbound API. Several SDN controllers are a beacon, maestro, NOX/POX, OpenDayLight (ODL), open network operating system (ONOS), floodlight, RYU, etc. These SDN controllers worked as the brain of the SDN architecture [14]. Thus, for the new architecture, an efficient controller is required to improve the utilization of resources for better performance of the network in the SDN environment [15].

The SDN performs traffic engineering to optimize network resources and to achieve enhanced performance; however, still there exists a research gap to implement the structure of SDN with custom topology for traffic engineering and to provide an in depth illustration of performance measures needed for exploring the future of SDN [16, 17]. To the best of our knowledge, there has not been enough research in this domain for SDN architecture performance analysis using the RYU SDN controller [18]. Hence, it needs to be explored further to assess the performance parameters for the SDN environment.

The motivation behind this work is to implement the architecture of SDN using an open-source RYU SDN controller for the network traffic analysis because there is a great need for a high-performance controller in data centres, academia and networking industries. Therefore, it is crucial to investigate the performance of an open-source controller. The implementation of the SDN architecture is carried out in the Mininet emulator, including the RYU controller for a custom-designed topology which consists of a switching hub, network nodes considered under a single topology, and an OpenFlow switch. The proposed work aims to evaluate the in depth performance analysis of SDN architecture for various

parameters such as the number of packets transmitted, packets received, throughput, bandwidth, round trip time, etc.

### 1.1 Paper Organization

The rest of the paper is organized as mentioned graphically in Fig. 1. Section 2 elaborates the related work in SDN. Section 3 describes the background of the RYU SDN controller, followed by TCP 3-way handshake in SDN. Section 4 discusses the problem definition, which includes importance, the context of the research statement, and framework for illustrating results. Section 5 presents the methodology. Section 6 elaborates the proposed research and its implementation. Section 7 presents the results of implementation. Section 8 provides a discussion about the comparison of the proposed work with the existing state of the art. Finally, Sect. 9 concludes the paper.

## 2 Related Work

This section summarizes the existing literature in the field of SDN. The aim of reviewing the literature is to explore the research gaps for framing the objectives for performing research in the SDN environment. Islam et al. [18] investigated the performance of the

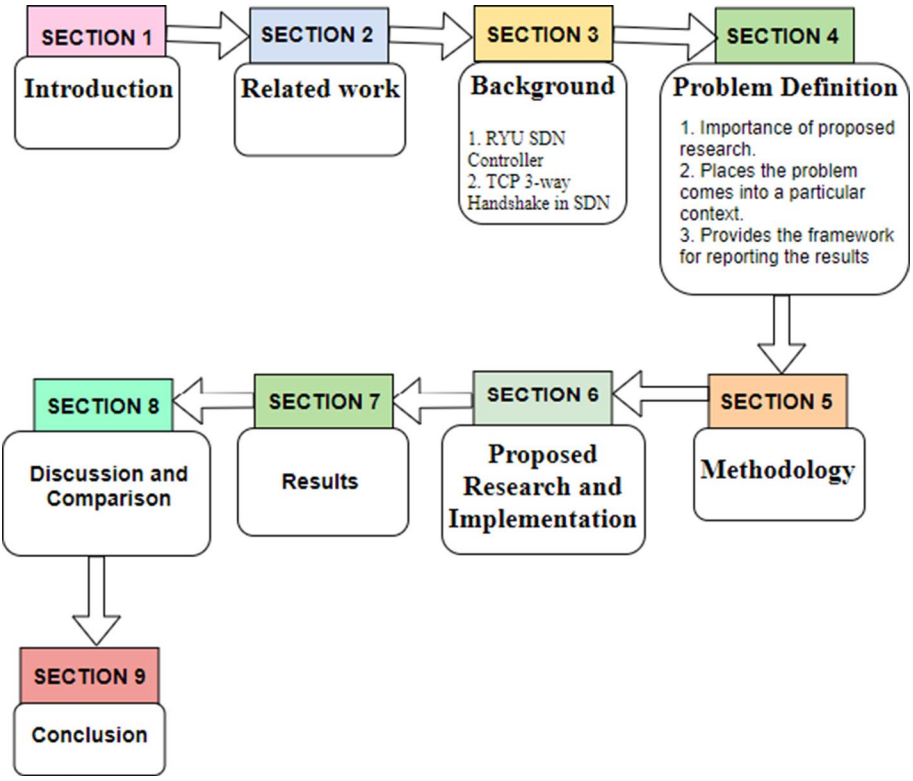


**Fig. 1** Organization of the paper

RYU controller in an SDN environment and evaluated different parameters based on data traffic such as throughput, packet loss, and bandwidth. The experimentation was performed in the Mininet emulator with one of the OpenFlow switches. The authors have explored the SDN environment performance for default topology. However, the performance evaluation of SDN environment for custom topology still needs to be explored further. Queiroz et al. [19] proposed an SDN approach for traffic monitoring and analysis based on Big Data. The authors determined the consumption of bandwidth of network resources in the SDN environment. The authors suggested that there still exists scope for further enhancements to increase the performance of SDN architecture. Badotra et al. [20] provided the comparative study of two open-source SDN controllers named as ODL and ONOS. The author claimed that the performance of the ODL controller was much better than the ONOS controller based on performance metrics such as throughput, bandwidth, and burst time. Priya et al. [21] compared the performance parameters of OpenFlow controllers named NOX, RYU, FloodLight, and POX. The comparison was made based on packet handling capacity on the Distributed Internet Traffic Generator (D-ITG) tool. The authors concluded that the FloodLight controller shows better performance in terms of throughput and delay as compared to other SDN controllers. The authors suggested that in the future, the work can be further extended by comparing the OpenDayLight and OpenContrail controller. Bhatia et al. [22] proposed an algorithm for the real-time data transmission supervised under SDN controller to evaluate the network performance parameters such as density and congestion prediction of the data traffic. The authors concluded that the proposed algorithm is useful in transmitting real-time data traffic in the SDN network with 97% accuracy. However, the evaluation of the proposed model for the vehicular density of congestion control needs to be explored further. Singh and Jha [23] compared the performance of different algorithms for SDN control planes. The authors measured the performance based on jitter, latency, Q-factor values and concluded that centralized architecture has better performance as compared to the other architectures. The authors suggested that future refinements to the SDN network can be made by proposing a multi-controller arrangement. Amin et al. [24] presented a comprehensive survey for hybrid SDN environments and identified the research gaps along with existing solutions in this domain. The authors mentioned future directions to implement the automated system for the management of the SDN network that works out in the management plane. Bholebawa et al. [25] provided the performance comparison of two SDN controllers: Floodlight and POX. The authors presented different SDN topologies for both controllers and compared the performance based on throughput and roundtrip time. The authors concluded that the Floodlight controller provided more effective results as compared to POX. Yu et al. [26] proposed SDN architecture which was capable of removing the vulnerabilities in the data traffic for end-to-end data communication and allowed the flexible assignment of IP as per the requirement. The authors concluded that future enhancements could be made to improve the security aspects of the SDN architecture. Wang et al. [27] proposed a solution for the reliable transmission of data packets from one host to another host in the SDN environment. The proposed framework provided a well-regulated usage of bandwidth as well as the guaranteed reliable packet transmission. Akyildiz et al. [28] investigated the SDN environment using traffic engineering, which includes flow management and traffic analysis. The authors also highlighted various key challenges related to SDN traffic engineering, such as availability and scalability. Tootoonchian et al. [29] provided the performance analysis of publically available OpenFlow controllers such as Beacon, NOX, Maestro, and NOX-AT. The authors also presented the design of a series of flow-based benchmarks for the SDN environment.

The literature review reveals that SDN allows an easy configuration of the network by providing the flexibility to separate the control plane from the data plane and hence optimizes the network resources by achieving better performance. RYU is considered one of the best SDN controllers for traffic engineering and is specially designed to make the network agile [18]. However, to the best of our knowledge, there is still a gap in implementing the SDN architecture using custom topology and providing an in-depth illustration of various network performance measures using the RYU controller [18]. Therefore, the proposed research focuses on the implementation of the SDN architecture using the RYU controller for a custom-designed topology in the Mininet emulator. The proposed work evaluates various node to node performances of the SDN network using throughput, bandwidth, round trip time, the transmission of OpenFlow packets, etc., on the RYU SDN controller.

## 3 Background

This section provides a brief description of the RYU SDN controller and its architecture [30].

### 3.1 RYU SDN Controller

RYU [31] is an open-source and component-based SDN controller that has been developed by a telecommunications company in Japan named Nippon Telegraph and Telephone (NTT). The controller is licensed under Apache 2.0. The word "RYU" is a Japanese word that means "flow". The controller directs the flow control to empower inventive networking. Several protocols are supported in the RYU controller for the management of the network, such as Netconf, OpenFlow, etc. [32]. In case of OpenFlow, all the versions from 1.0 to 1.5 are supported by the RYU controller. RYU controller is implemented in one of the most versatile Python programming languages [33].

Figure 2 provides the detailed architectural view of the RYU SDN controller. The architecture of the RYU SDN controller is categorized into three various planes: application layer, network layer, and physical layer [34]. The bottom layer is the physical layer, which consists of different physical and virtual devices connected via the internet to communicate with each other. This layer is also known as the infrastructure layer because this is the base layer that consists of various devices positioned on the same plane. The middle layer is the network layer which is also known as the control layer because this layer controls the flow of data traffic from one node to another node to maintain stability in the network without traffic overhead. The interface between the physical layer and the network layer is provided with the help of well-defined Application Programming Interfaces (APIs) known as southbound interfaces such as Netconf, OpenFlow, OF-config, etc. The topmost layer is the application layer which consists of the end-user applications such as network and business logic. The interface between the network layer and the application layer is provided with the help of APIs known as northbound interface such as REST, Quantum, user-defined and Restful management [35]. The main target of these above-mentioned applications is to collect the intelligence of the network at one centralized place named as the controller.

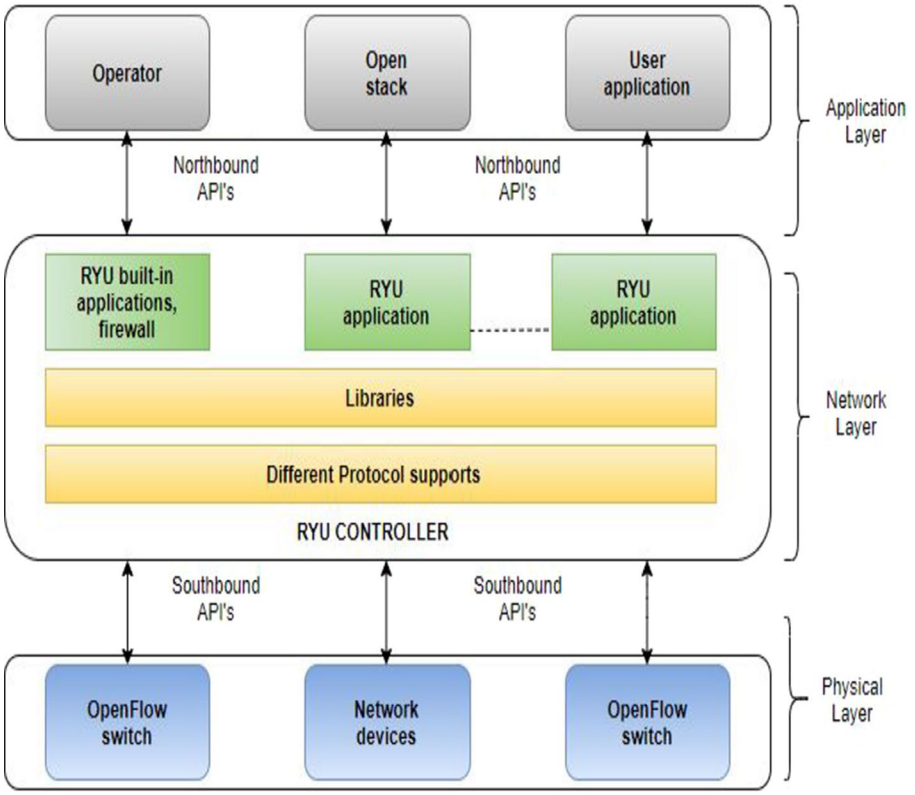


**Fig. 2** Architecture of RYU SDN controller

### 3.2 TCP 3- Way Handshake in SDN

Figure 3 shows the flow of TCP in the SDN environment. It depicts that the source node sends a request packet as TCP SYN to the destination node for the establishment of connection via the SDN switch. When a switch receives the request packet, it matches the request packet header with the entries in the flow table. If the match is found in the flow table, the switch will forward the packet to the destination node. Otherwise, the switch will forward the packet to the controller. Now, the centralized controller determines the right path for routing the packet from the source node to the destination node. After determining the route, the controller updates the switch by adding a new entry in the flow table and move the packet back to the switch [36].

When the destination node receives the SYN packet, it returns the SYN as well as the ACK packet to the source node. If the flow table is having an entry of packet SYN + ACK in the switch, then the packet will be sent to the source node [37]. Otherwise, the switch forwards the packet to the controller, and then the controller selects the best route for the packet. Thereafter,

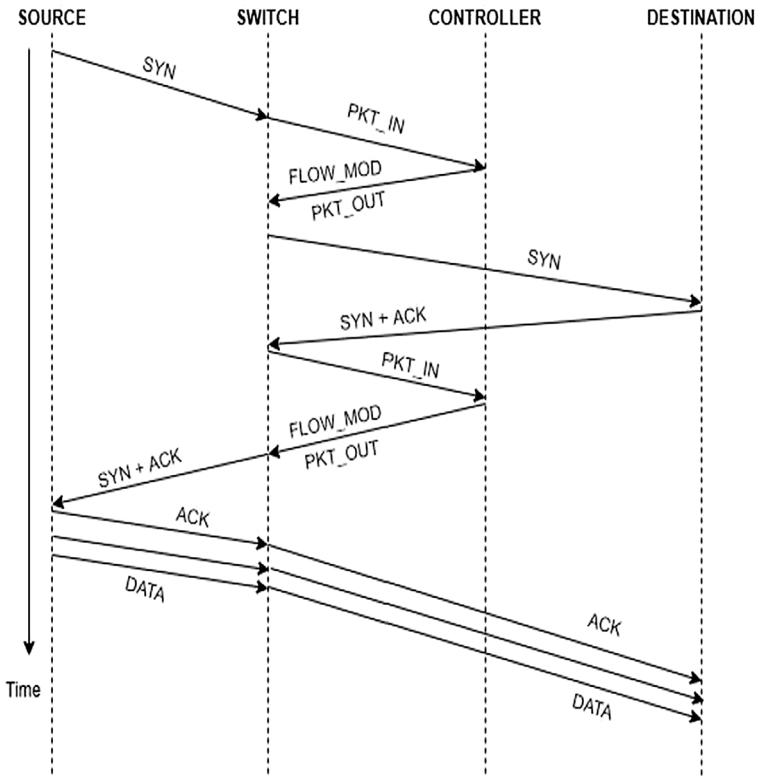


**Fig. 3** TCP 3-way handshake in SDN

the controller updates the switch to add a new entry in the flow table and send the packet back to the switchback.

Finally, when the SYN + ACK packets are received by the source node, it forwards the ACK to the destination node via the switch. In this way, a 3-way handshake is done using TCP, which establishes an end-to-end node connection between source and destination.

## 4 Problem Definition

The motivation behind this work is that there is a great need for a high performance controller in data centres for running varied number of applications in the distributed computing environment. However, there is a lack of sufficient research in this area of the traffic performance evaluation of the RYU SDN controller in the SDN environment. The existing approaches lag in providing the in-depth traffic performance evaluation of the RYU SDN controller [38]. The performance of the SDN controller provides a magnificent impact on amplifying the capabilities of the SDN network. Therefore, it is crucial to investigate the performance of an open-source SDN controller.

**Table 1** Problem definition

| | |
|---|---|
| Importance of proposed research | There is a great need for a high-performance controller in networking industries, academia, and data centres<br>The performance of the SDN controller provides a magnificent impact on amplifying the capabilities of the SDN network. Therefore, it is crucial to investigate the performance of an open-source SDN controller |
| Places the problem comes into a particular context | There is a lack of sufficient research in this area of traffic engineering for the SDN environment<br>The existing approaches lag in providing the in depth traffic performance evaluation of the RYU SDN controller [38] |
| Provides the framework for reporting the results | The proposed research evaluates the node to node performance metrics such as bandwidth, throughput, round trip time, etc., using the RYU SDN controller |

Table 1 highlights the problem definition of the research based on the following points: the importance of proposed research, places the problem comes into a particular context, and the framework for reporting the results.

## 5 Methodology

This section illustrates the methodology followed to perform the proposed research as depicted in Fig. 4. The first step of the methodology is to review the existing literature about network traffic analysis using SDN controllers in order to know the research gaps and objectives of our research. This has been discussed in Sect. 2. Thereafter, the Mininet tool is used to build up a custom network for the SDN environment. Later, the next important task is the implementation of the controller in the SDN. In the Mininet topology, the implementation of an open-source RYU SDN controller was done. Once the network is created, thereafter the data traffic is generated from source to destination node using the Wireshark tool. Lastly, the performance of the SDN network using the RYU controller is evaluated based on various metrics, and the proposed work is compared with the existing state of the art. Section 6 will provide the detailed elaboration of methodological steps for the implementation.

## 6 Proposed Research

This section elaborates on the proposed research and its implementation. The SDN architecture can be implemented using three constituents: network devices, controller, and application [39]. The controller is one of the primary entities of the SDN network, which receives the command from the application layer and forwards it to the network layer. The application layer communicates with the SDN controller with the help of different APIs [40].

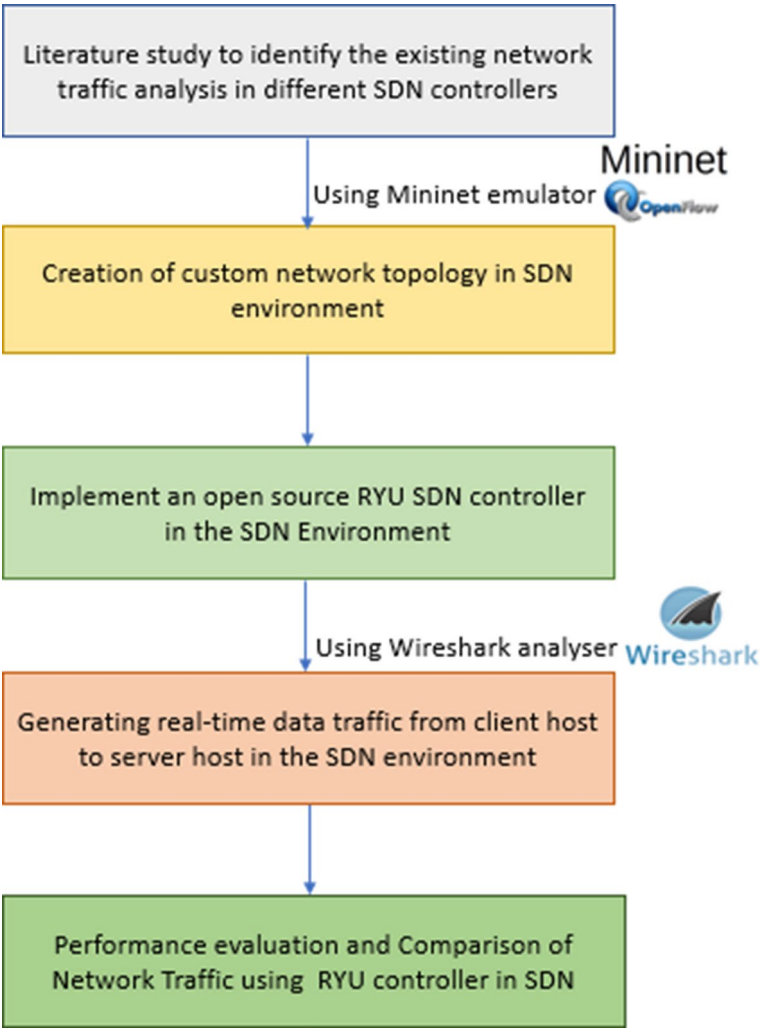


**Fig. 4** Methodology of the performance evaluation of SDN using RYU SDN controller

Implementing the physical framework of SDN architecture using real testbeds and hardware is very expensive. Therefore, in order to carry out the research in the SDN environment, a simulator or emulator is a good choice [41]. Some of the emulators are OFNet [42], NS3 [43], OMNET++ [44], Mininet. The experimentation for this research paper has been performed in the tool named Mininet. Mininet is an emulator which is an open-source network. It uses the command-line interface for communication. The SDN architecture has been created to transfer the data traffic from one node to a different node via a specific SDN controller in the SDN emulator. Figure 5 represents the abstract view of the proposed SDN architecture.

Figure 6 illustrates the detailed view of the designed custom SDN topology. The data plane consists of various networking devices such as OpenFlow switch, network nodes, forwarding engines, and processing functions. The connections h1-eth0, h2-eth0, h3-eth0,

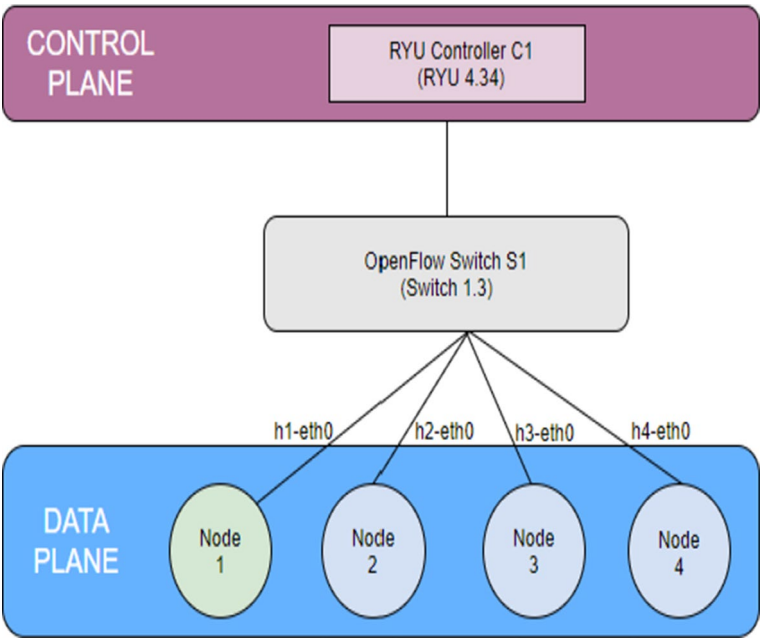


**Fig. 5** Abstract view of designed custom SDN topology

and h4-eth0 are the ports situated on networking nodes. The connection s1-eth1, s1-eth2, s1-eth3, and s1-eth4 are the ports situated on the OpenFlow switch. Each networking node has its forwarding engine and link to the processing function in order to proceed. The control plane consists of the NBI agents and the CDPI drivers, which makes communication possible between the data plane and control plane as well as the control plane and application plane. The RYU controller is implemented in the control plane having port number 6633 with IP address 127.0.0.1. In this proposed SDN custom topology, the application plane runs the network bandwidth application and the network topology viewer application for the analysis of the data traffic.

Figure 7 shows the screenshot of the custom SDN topology created in the Mininet tool having four nodes/hosts named h1, h2, h3, h4, and one OpenFlow switch. The links are created to make the connection among the hosts and switch such as (h1, s1), (h2, s1), (h3, s1), (h4, s1). After the physical switch creation, it would comprise with the capabilities of the corresponding controller (RYU), as shown in Fig. 8. By this step, the OpenFlow switch 1.3 was loaded to transmit the data packets from one host to another host. The data packets were generated in the network with the help of the Wireshark tool. These data packets were sent from the source host to the destination host.

To check the connections among the network devices, the "Links" command is used. The links of the OpenFlow switch with the respective hosts are mentioned as:

Figure 9 depicts the TCP window size, which is 85.3 Kb for this SDN topology. The total data packets transferred over the network with the RYU controller are 4.99 GB having a bandwidth of 4.29 Gbits/s.

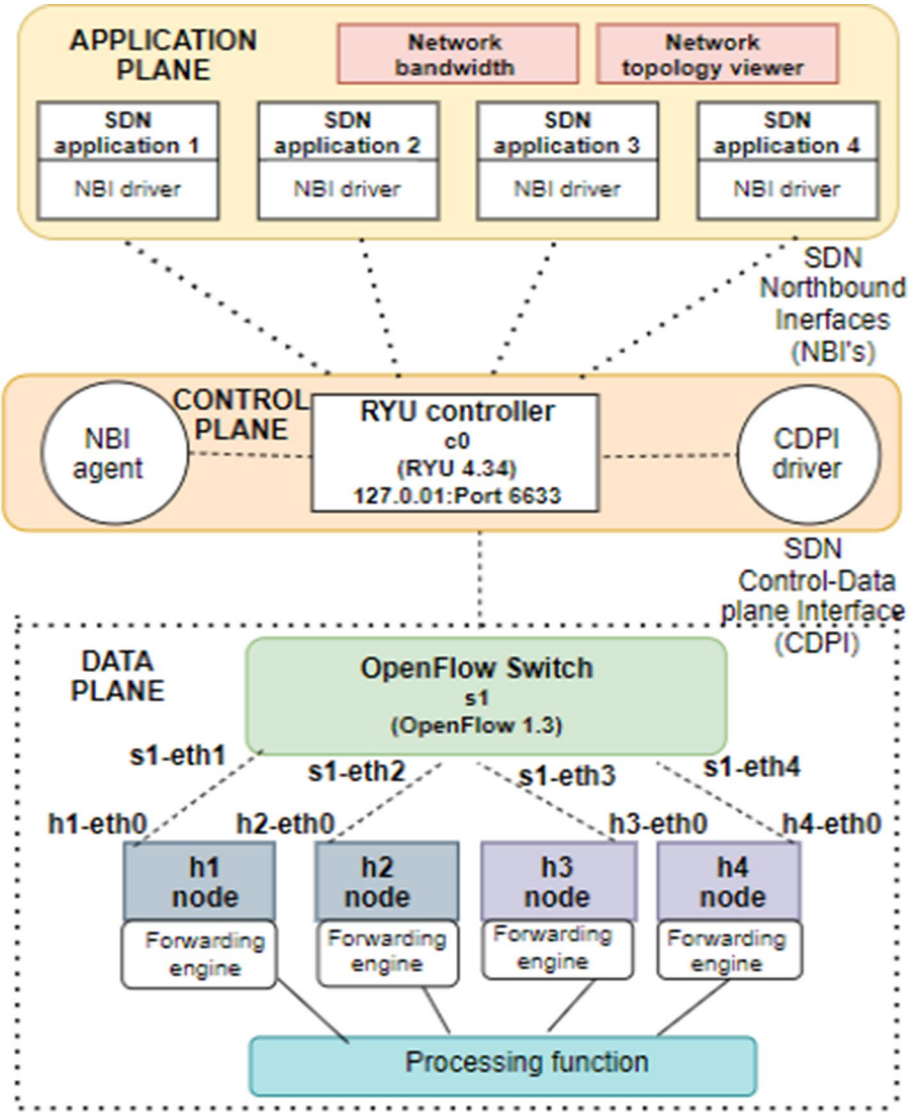


**Fig. 6** Detailed view of designed custom SDN topology

## 7 Result Analysis

This section reports the results obtained for the SDN network performance evaluation through the RYU SDN controller. The performance parameters such as bandwidth, throughput, round trip time have been evaluated under TCP, UDP data traffic using several benchmarking tools like ping as well as iperf.

Figure 10 depicts the connections among different ports. Here, the host is connected to the TCP port 5001 via IP 10.0.0.4, and the default size of the TCP window taken is

```
ubuntu@ubuntu-VirtualBox:~$ sudo mn --controller=remote,ip=127.0.0.1 --mac --sw
itch=ovsk,protocols=OpenFlow13 --topo=single,4
[sudo] password for ubuntu:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
```

**Fig. 7** Topology creation in SDN environment

782 Kb. The experimentation shows the interval between the client host and the server host, the transfer rate, and the bandwidth of the particular connection.

### 7.1 Bandwidth

RYU is an open-source SDN controller, and iperf benchmark tool is utilized to generate the TCP data traffic for the performance evaluation of bandwidth of the RYU SDN controller. In TCP traffic, the source node sends a request packet TCP SYN to the destination node for the establishment of connection via the SDN switch. TCP checks the number of packets sent to the destination host via a different number of processes. This test is regulated to measure the bandwidth of TCP traffic among the nodes. Here, the iperf command has been executed for 10 s.

Figure 11 shows the bandwidth of the proposed custom network topology between the nodes. The maximum data transferred between h1 to h2 is 26.7 Gbps, h1 to h3 is 25.2 Gbps, h1 to h4 is 25 Gbps, h2 to h3 is 24.8 Gbps, h2 to h4 is 23.8 Gbps, and h3 to h4 22.7 Gbps.

```
ubuntu@ubuntu-VirtualBox:~$ ryu-manager 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
packet in 1 00:00:00:00:00:01 33:33:00:00:00:02 1
packet in 1 00:00:00:00:00:02 33:33:00:00:00:02 2
packet in 1 00:00:00:00:00:04 33:33:00:00:00:02 4
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
packet in 1 00:00:00:00:00:02 00:00:00:00:00:01 2
packet in 1 00:00:00:00:00:01 00:00:00:00:00:02 1
packet in 1 00:00:00:00:00:03 33:33:00:00:00:02 3
packet in 1 00:00:00:00:00:02 33:33:00:00:00:02 2
packet in 1 00:00:00:00:00:01 33:33:00:00:00:02 1
packet in 1 00:00:00:00:00:04 33:33:00:00:00:02 4
packet in 1 00:00:00:00:00:03 33:33:00:00:00:02 3
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
packet in 1 00:00:00:00:00:04 00:00:00:00:00:01 4
packet in 1 00:00:00:00:00:01 00:00:00:00:00:04 1
packet in 1 00:00:00:00:00:02 33:33:00:00:00:02 2
packet in 1 00:00:00:00:00:01 33:33:00:00:00:02 1
packet in 1 00:00:00:00:00:04 33:33:00:00:00:02 4
packet in 1 00:00:00:00:00:03 33:33:00:00:00:02 3
packet in 1 00:00:00:00:00:02 33:33:00:00:00:02 2
packet in 1 00:00:00:00:00:04 33:33:00:00:00:02 4
packet in 1 00:00:00:00:00:01 33:33:00:00:00:02 1
packet in 1 00:00:00:00:00:03 33:33:00:00:00:02 3
```

**Fig. 8** Implementation of OpenFlow switch

### 7.2 Throughput

Throughput represents the maximum amount of data traffic processed via controller between two nodes of the network in a second. In TCP traffic, one of the host act as a client and another act as a server. iperf3 utility has been utilized to test the throughput of the controller. On the customer side, iperf3 has been implemented in 10 s, and the information about throughput has been gathered on another side of the network. The result of execution is shown in Table 2, and the same is shown graphically in Figs. 12 and 13.

Table 2 depicts the highest and lowest throughput in Gbps. It is 28.5 and 22.6 Gbps between h1 to h2, 28.5 and 20.9 Gbps between h1 to h3, 27.4 and 20.9 Gbps between h1 to h4, 29.9 and 22.1 Gbps between h2 to h3, 27.4 and 20.9 Gbps between h2 to h4, and 28.5 and 21.6 Gbps between h3 to h4 respectively.

```
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
h3-eth0<->s1-eth3 (OK OK)
h4-eth0<->s1-eth4 (OK OK)
mininet> ports
s1 lo:0 s1-eth1:1 s1-eth2:2 s1-eth3:3 s1-eth4:4
mininet> h4 iperf -s &
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
mininet> h1 iperf -c h4
------------------------------------------------------------
Client connecting to 10.0.0.4, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.1 port 53450 connected with 10.0.0.4 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  4.99 GBytes  4.29 Gbits/sec
```

**Fig. 9** TCP window size in proposed SDN topology

```
mininet> h1 iperf -c h4 -r
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
------------------------------------------------------------
Client connecting to 10.0.0.4, TCP port 5001
TCP window size:  782 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.1 port 53494 connected with 10.0.0.4 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  7.34 GBytes  6.31 Gbits/sec
[  5] local 10.0.0.1 port 5001 connected with 10.0.0.4 port 33080
[  5]  0.0-10.0 sec  4.32 GBytes  3.71 Gbits/sec
```

**Fig. 10** Implementation of OpenFlow switch

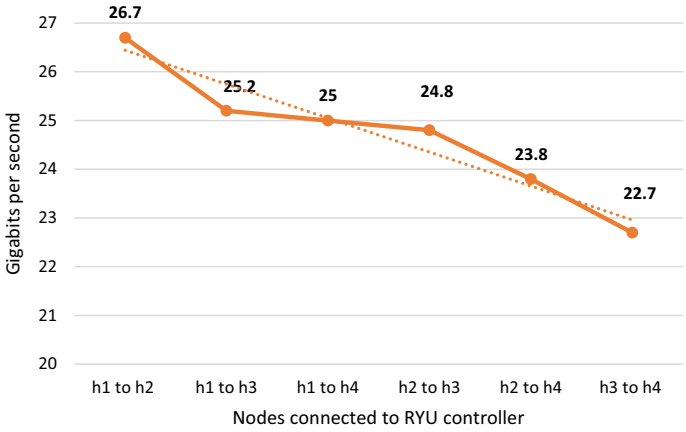


**Fig. 11** Comparison of TCP bandwidth between network nodes

**Table 2** TCP throughput for a node to node path of the network

| Time (s) | Nodes | | | | | |
|---|---|---|---|---|---|---|
| | h1 to h2 throughput (Gbps) | h1 to h3 throughput (Gbps) | h1 to h4 throughput (Gbps) | h2 to h3 throughput (Gbps) | h2 to h4 throughput (Gbps) | h3 to h4 throughput (Gbps) |
| 1 | 28.5 | 21.6 | 22.1 | 23.0 | 26.2 | 24.1 |
| 2 | 26.9 | 22.1 | 23.3 | 24.1 | 27.4 | 23.3 |
| 3 | 22.6 | 24.1 | 26.9 | 26.2 | 20.9 | 26.6 |
| 4 | 23.0 | 26.9 | 26.6 | 27.4 | 21.2 | 28.5 |
| 5 | 24.1 | 28.5 | 26.2 | 27 | 22.9 | 26.2 |
| 6 | 26.6 | 23.0 | 27.4 | 29.9 | 23.3 | 21.6 |
| 7 | 23.3 | 23.3 | 20.9 | 24.4 | 24.1 | 23.0 |
| 8 | 22.9 | 27.4 | 23 | 26.6 | 22.5 | 22.1 |
| 9 | 27.4 | 20.9 | 23.2 | 26.0 | 23.0 | 26.9 |
| 10 | 26.2 | 22.9 | 22.9 | 22.1 | 21.6 | 22.1 |

### 7.3 Round-Trip Time

Round trip time (RTT) is also called a ping test time. RTT is the total time taken to by the data packet to reach from a specific source host to the destination host and the acknowledgement packet to reach back to the source. Ping utility uses Internet Control Message Protocol (ICMP). Table 3 shows comparison of RTT for node to node path depicting minimum, maximum, and average values. The minimum RTTs of the proposed SDN topology

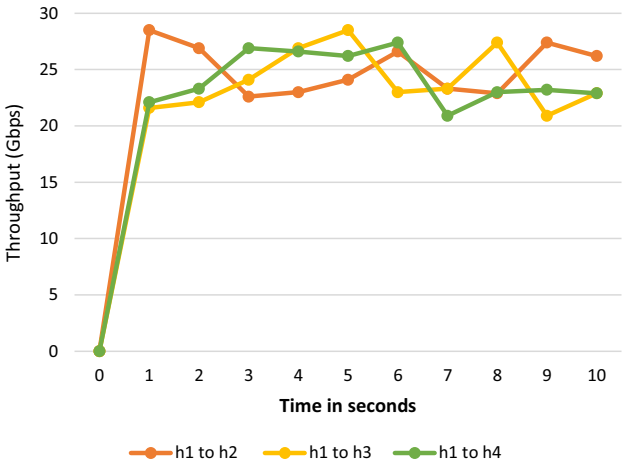


**Fig. 12** TCP throughput of the node to node path (from h1 to all hosts)

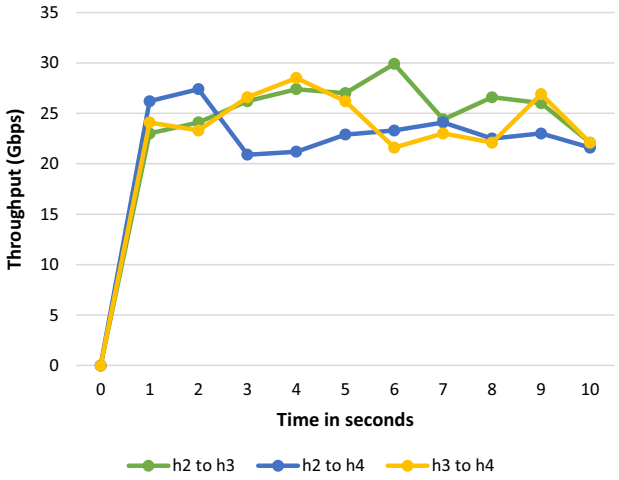


**Fig. 13** TCP throughput of node to node (from h2 and h3 to all hosts)

are measured as 0.054 ms, 0.036 ms, 0.057 ms, 0.057 ms, 0.011 ms, and 0.047 ms. The average RTTs of the proposed topology are measured as 1.002 ms, 1 ms, 0.999 ms, 0.888 ms, 1.05 ms, and 0.918 ms. The maximum RTTs of the proposed topology are 9.45 ms, 9.345 ms, 7.336 ms, 9.668 ms, and 8.149 ms as shown in Fig. 14.

### 7.4 Transmission of OpenFlow Packets

Figure 15 depicts the transmission of OpenFlow packets in the SDN network. It shows the number of bytes and the number of packets received for each connection. Table 4 shows that the connection s1-eth1 received 1640 bytes with 20 data packets, s1-eth2 received 1640 bytes

**Table 3** Comparative analysis of RTT for a node to a node path

| Nodes | Measures | Round trip time (in s) |
|---|---|---|
| h1 to h2 | Min | 0.054 |
| | Avg | 1.002 |
| | Max | 9.450 |
| h1 to h3 | Min | 0.036 |
| | Avg | 1 |
| | Max | 9.345 |
| h1 to h4 | Min | 0.057 |
| | Avg | 0.999 |
| | Max | 7.336 |
| h2 to h3 | Min | 0.057 |
| | Avg | 0.888 |
| | Max | 7.889 |
| h2 to h4 | Min | 0.011 |
| | Avg | 1.050 |
| | Max | 9.668 |
| h3 to h4 | Min | 0.047 |
| | Avg | 0.918 |
| | Max | 8.149 |

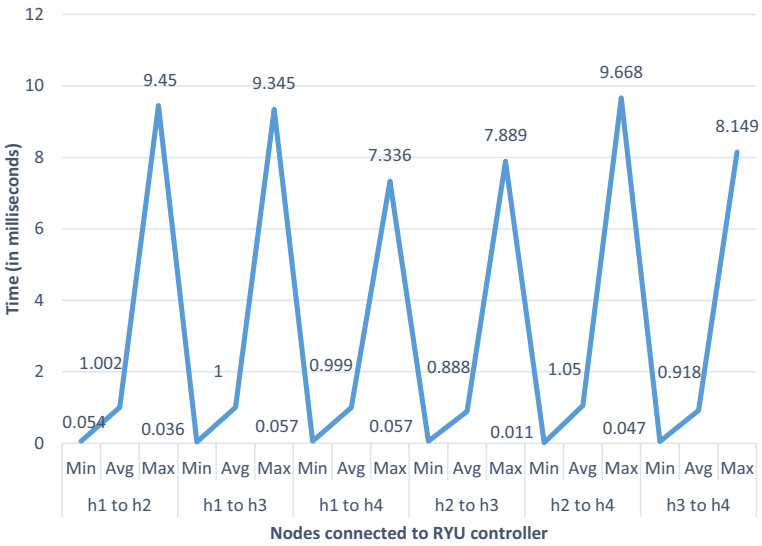


**Fig. 14** RTT of a node to node path

with 20 data packets, s1-eth3 received 1156 bytes of 14 data packets, and s1-eth4 received 1066 bytes of 13 data packets.

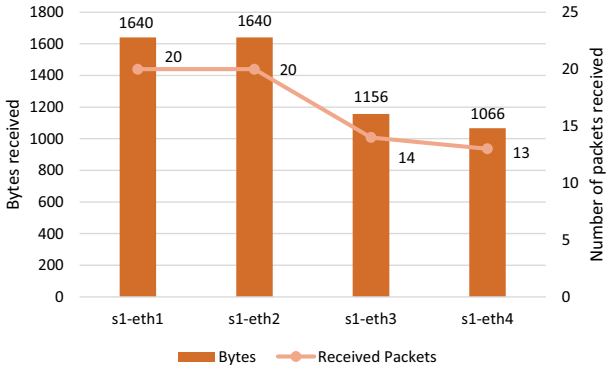


**Fig. 15** Transmission of Open Flow packets

**Table 4** Statistics of transmission of data packets

| Ports | Received packets | Received Bytes | Transmitted packets | Transmitted Bytes | Time (in a s) |
|---|---|---|---|---|---|
| s1-eth1 | 20 | 1640 | 49 | 5721 | 319.172 s |
| s1-eth2 | 20 | 1640 | 49 | 5721 | 319.172 s |
| s1-eth3 | 14 | 1156 | 43 | 5189 | 319.159 s |
| s1-eth4 | 13 | 1066 | 43 | 5189 | 319.171 s |

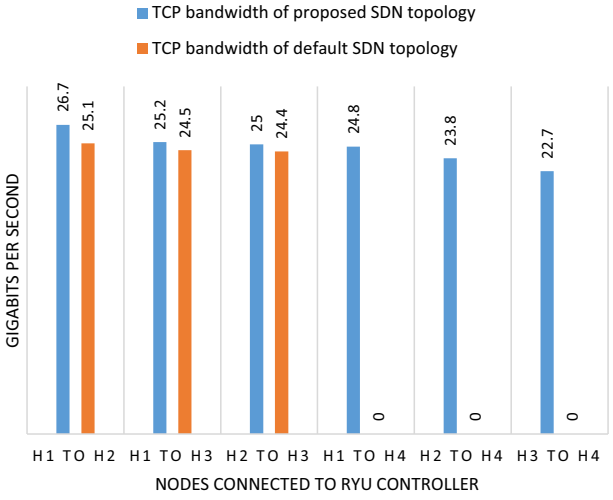


**Fig. 16** Comparison of the TCP bandwidth of proposed and default SDN topology

## 8 Discussion

This section provides a discussion on the comparison of proposed SDN topology and default SDN topology [18]. It demonstrated the efficiency of the proposed SDN topology with respect to the default SDN topology by analysing the result of performance metrics

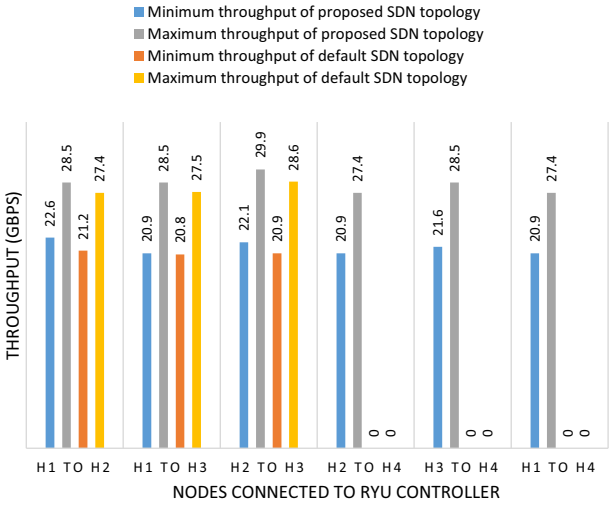


**Fig. 17** Comparison of throughput of proposed and default SDN topology

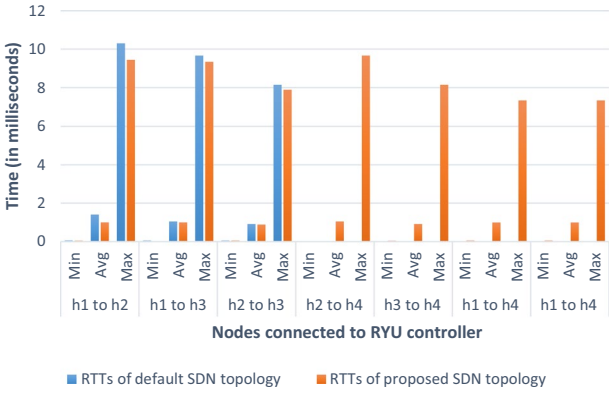


**Fig. 18** Comparison RTT's of proposed and default topology

such as bandwidth, throughput, and RTTs of nodes. The comparison has been evaluated by analysing the node to node performance of the metrics of both topologies.

The bandwidth of the network is the computation that indicates the maximal capacity of communication media links to transfer the data packets over a network in a particular interval of time. Figure 16 represents the comparative TCP bandwidth of the proposed SDN topology and the default SDN topology [18].

While investigating the bandwidth, using the Open Flow switch in the RYU SDN controller, the node to node performance metrics obtained represent that data packets transferred from host 1 (h1) to host 2 (h2) in the proposed topology is 26.7 Gbps and in the default topology is 25.1 Gbps. Similarly, the data packets transferred from h1 to h3 in the proposed custom topology is 25.2 Gbps, and in the default topology is 24.5 Gbps, between h2 to h3 in the proposed SDN topology is 25 Gbps, and in default, it is 24.4 Gbps. The total number of node combinations in default topology are three (h1

to h2, h1 to h3, and h2 to h3) whereas the proposed topology consists six node combinations (h1 to h2, h1 to h3, h1 to h4, h2 to h3, h2 to h4, and h3 to h4). Therefore, for three additional node combinations h1 to h4, h2 to h4, and h3 to h4, there are no values for the respective parameter in the default topology. The results of bandwidth depict a good improvement in the proposed custom topology as compared to the default SDN topology.

Figure 17 depicts the minimum and maximum throughput of proposed and default SDN topology in Gbps. The arrival of data packets is the key to achieve the high performance of the SDN network. The default SDN topology shows the minimum throughput from h1 to h2 is 21.2 Gbps, from h1 to h3 is 20.8 Gbps, and from h2 to h3 is 20.9 Gbps. However, the proposed topology shows 22.6 Gbps from h1 to h2, 20.9 Gbps from h1 to h3, 22.1 Gbps from h2 to h3, 21.6 Gbps from h2 to h4, and 20.9 Gbps from h1 to h4. The maximum throughput for default SDN topology is observed as 27.4 Gbps from h1 to h2 is 27.5Gbps from h1 to h3, and 28.6 Gbps from h2 to h3. However, the maximum node to node throughput for the proposed topology has been improved and measures 28.5 Gbps from h1 to h2, 28.5 Gbps from h1 to h3, 29.9 Gbps from h2 to h3, 27.4 Gbps from h2 to h4, and 27.4 Gbps from h1 to h4.

Figure 18 represents the comparison of minimum, average, and maximum RTT of the proposed and default SDN topology. The minimum RTTs of the default SDN topology are 0.061 ms, 1.05 ms, and 0.058 ms. However, the minimum RTTs of the proposed SDN topology are measured as 0.054 ms, 0.036 ms, 0.057 ms, 0.011 ms, 0.047 ms, and 0.057 ms which shows a good improvement.

The average RTTs of the default topology are 1.411 ms, 1.05 ms, and 0.918 ms, whereas the average RTTs of the proposed topology, are 1.002 ms, 1 ms, 0.888 ms, 1.05 ms, 0.918 ms, and 0.999 ms. This shows an improvement in the average RTT of proposed SDN topology in comparison to the default SDN topology.

The maximum RTTs of the default topology are 10.305 ms, 9.668 ms, and 8.149 ms. However, the maximum RTTs of the proposed topology are 9.45 ms, 9.345 ms, 7.889 ms, 9.668 ms, 8.149 ms, and 7.336 ms. The results show that the proposed topology has less RTT values as compared to that of the default topology.

## 9 Conclusion

In the recent past, there has been tremendous growth in technology, and traffic analysis is one of the crucial aspects of SDN's application domain. The controller is the brain of the network and is responsible for the analysis and monitoring of real-time data traffic. In any networking approach, monitoring and analysis of real-time data traffic is a significant mechanism to observe the transmission of data packets from the source host to the destination host. Therefore, the paper implements the architecture of SDN using an open-source RYU SDN controller for the network traffic analysis. This work can leverage better performance metrics for traffic routing in the SDN environment for enhanced usage of the network for various high end applications in the future. The proposed work provides the traffic analysis via performance evaluation of the RYU controller in the SDN environment to improve the utilization of resources for the better performance of the network, management of data traffic in the network, reduction of the cost for the innovations in the existing solutions, and many more.

RYU is one of the most recommended Open-source and widely used SDN controllers, and it is scripted in a python programming language. This presented work will be helpful for all the researchers working in the field of SDN and controller traffic evaluation. The experimentations mentioned above proved that the RYU SDN controller could also be considered one of the powerful controllers for traffic engineering. The research work proved the effective results for various performance metrics in the SDN environment using the RYU controller in comparison to the existing state of the art.

## References


1. Wan, J., Zou, C., Zhou, K., Lu, R., & Li, D. (2014). IoT sensing framework with inter-cloud computing capability in vehicular networking. *Electronic Commerce Research, 14*(3), 389–416.
2. Wan, J., Zhang, D., Sun, Y., Lin, K., Zou, C., & Cai, H. (2014). VCMIA: A novel architecture for integrating vehicular cyber-physical systems and mobile cloud computing. *Mobile Networks and Applications, 19*(2), 153–160.
3. Sharif, A., Li, J. P., & Sharif, M. I. (2019). Internet of Things network cognition and traffic management system. *Cluster Computing, 22*(6), 13209–13217.
4. Yan, Q., Yu, F. R., Gong, Q., & Li, J. (2015). Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges. *IEEE Communications Surveys and Tutorials, 18*(1), 602–622.
5. Farhady, H., Lee, H., & Nakao, A. (2015). Software-defined networking: A survey. *Computer Networks, 81*, 79–95.
6. Queiroz, W., Capretz, M. A., & Dantas, M. (2019). An approach for SDN traffic monitoring based on big data techniques. *Journal of Network and Computer Applications, 131*, 28–39.
7. Bhardwaj, S., & Panda, S. N. (2020). A study on noninvasive body wearable sensors. In *Intelligent communication, control and devices* (pp. 345–351). Springer, Singapore.
8. Akyildiz, I. F., Lee, A., Wang, P., Luo, M., & Chou, W. (2014). A roadmap for traffic engineering in SDN-OpenFlow networks. *Computer Networks, 71*, 1–30.
9. Phemius, K., Bouet, M., & Leguay, J. (2014). Disco: Distributed multi-domain sdn controllers. In *2014 IEEE network operations and management symposium (NOMS)* (pp. 1–4). IEEE.
10. Agarwal, S., Kodialam, M., & Lakshman, T. V. (2013). Traffic engineering in software defined networks. In *2013 Proceedings IEEE INFOCOM* (pp. 2211–2219). IEEE.
11. Tahaei, H., Salleh, R. B., Ab Razak, M. F., Ko, K., & Anuar, N. B. (2018). Cost effective network flow measurement for software defined networks: A distributed controller scenario. *IEEE Access, 6*, 5182–5198.
12. Bawany, N. Z., & Shamsi, J. A. (2019). SEAL: SDN based secure and agile framework for protecting smart city applications from DDoS attacks. *Journal of Network and Computer Applications, 145*, 102381.
13. Bevi, A. R., Shakthipriya, P., & Malarvizhi, S. (2019). Design of software defined networking gateway for the internet-of-things. *Wireless Personal Communications, 107*(2), 1273–1287.
14. Srivastava, V., & Pandey, R. S. (2020). A reward based formal model for distributed software defined networks. *Wireless Personal Communications, 116*, 691–707.
15. Freris, N. M. (2019). A software-defined architecture for control of IoT cyberphysical systems. *Cluster Computing, 22*(4), 1107–1122.
16. Suárez-Varela, J., & Barlet-Ros, P. (2018). Flow monitoring in Software-Defined Networks: Finding the accuracy/performance tradeoffs. *Computer Networks, 135*, 289–301.
17. Bhardwaj, S., & Panda, S. N. (2019). SDWSN: Software-defined wireless sensor networking. *Internal Journal of Innovative Technology and Exploring Engineering, 8*(12), 1064–1071.
18. Islam, M. T., Islam, N., & Al Refat, M. (2020). Node to node performance evaluation through RYU SDN controller. *Wireless Personal Communications, 1–16*, 15.
19. Queiroz, W., Capretz, M. A., & Dantas, M. (2019). An approach for SDN traffic monitoring based on big data techniques. *Journal of Network and Computer Applications, 131*(28–39), 16.
20. Badotra, S., & Panda, S. N. (2019). Evaluation and comparison of OpenDayLight and open networking operating system in software-defined networking. *Cluster Computing, 1–11*, 17.
21. Priya, A. V., & Radhika, N. (2019). Performance comparison of SDN OpenFlow controllers. *International Journal of Computer Aided Engineering and Technology, 11*(4–5), 467–479.

22. Bhatia, J., Dave, R., Bhayani, H., Tanwar, S., & Nayyar, A. (2020). Sdn-based real-time urban traffic analysis in vanet environment. *Computer Communications, 149*(162–175), 19.
23. Singh, S., & Jha, R. K. (2019). SDOWN: A novel algorithm and comparative performance analysis of underlying infrastructure in software defined heterogeneous network. *Fiber and Integrated Optics, 38*(1), 43–75.
24. Amin, R., Reisslein, M., & Shah, N. (2018). Hybrid SDN networks: A survey of existing approaches. *IEEE Communications Surveys and Tutorials, 20*(4), 3259–3306.
25. Bholebawa, I. Z., & Dalal, U. D. (2018). Performance analysis of SDN/OpenFlow controllers: POX versus foodlight. *Wireless Personal Communications, 98*(2), 1679–1699.
26. Yu, L., Wang, Q., Barrineau, G., Oakley, J., Brooks, R. R., & Wang, K. C. (2017). TARN: A SDN-based traffic analysis resistant network architecture. In *2017 12th international conference on malicious and unwanted software (MALWARE)* (pp. 91–98). IEEE.
27. Wang, M. H., Chen, L. W., Chi, P. W., & Lei, C. L. (2017). SDUDP: A reliable UDP-Based transmission protocol over SDN. *IEEE Access, 5*(5904–5916), 23.
28. Akyildiz, I. F., Lee, A., Wang, P., Luo, M., & Chou, W. (2016). Research challenges for traffic engineering in software defined networks. *IEEE Network, 30*(3), 52–58.
29. Tootoonchian, A., Gorbunov, S., Ganjali, Y., Casado, M., & Sherwood, R. (2012). On controller performance in software-defined networks. In *2nd {USENIX} Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE 12)*.
30. Khondoker, R., Zaalouk, A., Marx, R., & Bayarou, K. (2014). Feature based comparison and selection of software defned networking (SDN) controllers. In *2014 world congress on computer applications and information systems (WCCAIS)* (pp. 1–7). IEEE.
31. Silva, J. B., Silva, F. S. D., Neto, E. P., Lemos, M., & Neto, A. (2020). Benchmarking of mainstream SDN controllers over open off-the-shelf software-switches. *Internet Technology Letters, 3*(3), e152.
32. Meena, R. C., Bundele, M., & Nawal, M. (2020, February). RYU SDN controller testbed for performance testing of source address validation techniques. In *2020 3rd international conference on emerging technologies in computer engineering: Machine learning and internet of things (ICETCE)* (pp. 1–6). IEEE.
33. Ali, J., Lee, S., & Roh, B. H. (2018). Performance analysis of POX and Ryu with different SDN topologies. In *Proceedings of the 2018 international conference on information science and system* (pp. 244–249).
34. Asadollahi, S., Goswami, B., & Sameer, M. (2018). Ryu controller's scalability experiment on software defined networks. In 2018 *IEEE International conference on current trends in advanced computing (ICCTAC)* (pp. 1–5). IEEE.
35. Eftimie, A., & Borcoci, E. (2020). SDN controller implementation using OpenDaylight: experiments. In *2020 13th International Conference on Communications (COMM)* (pp. 477–481). IEEE.
36. Saeed, N. S. B., & Alenazi, M. J. (2020). Utilizing SDN to deliver maximum TCP flow for data centers. In *Proceedings of the 2020 the 3rd international conference on information science and system* (pp. 181–187).
37. Torres-Jr, P. R., García-Martínez, A., Bagnulo, M., & Ribeiro, E. P. (2020). Bartolomeu: An SDN rebalancing system across multiple interdomain paths. *Computer Networks*, *169*, 107117.
38. Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems, 29*(7), 1645–1660.
39. Bhardwaj, S., Panda, S. N., & Datta, P. (2020). Comparison and performance evaluation of software-defined networking controllers. In *2020 international conference on emerging smart computing and informatics (ESCI)* (pp. 276–281). IEEE.
40. 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.
41. Hou, X., Wu, M., & Zhao, M. (2019). An optimization routing algorithm based on segment routing in software-defined networks. *Sensors, 19*(1), 49.
42. "OFNet Quick User Guide." [Online]. http://SDNinsights.org/. Accessed 13 Sept 2020.
43. "NS-3." [Online]. https://www.nsnam.org/. Accessed 13 Sept 2020.
44. "OMNeT++ Discrete Event Simulator-Home." [Online]. https://www.omnetpp.org/. Accessed 13 Sept 2020.

**Shanu Bhardwaj** is currently pursuing M.Tech. in Computer Science and Engineering from Chitkara University, Punjab, India. She has completed her B.Tech. in Computer Science and Engineering from Kurukshetra University, Kurukshetra (India). Her research areas are Software-Defined Networking, Internet of Things (IoT), Wireless sensor network (WSN) and Network security. She has completed her internship on "Application of Big Data in Construction Law" from Tamkang University, Taiwan.

**S. N. Panda** has completed his Ph.D. (Computer Sci. & Appl.) from Kurukshetra University Kurukshetra, Haryana (India), M.Sc. (Computer Science) from Kurukshetra University, Kurukshetra, Haryana (India), B.Sc. (Hons) Distinction from Sambalpur University, Orissa (India), PGDEDP from National Institute of Electronics and IT formerly known as Regional Computer Centre, Chandigarh (India). He is currently working as Director (Research), Centre of Advanced Computing and Research, Chitkara University, Punjab Campus, and Rajpura Punjab, India. His domain of research are Technology Trends, Machine Vision, Communication, Security, Big Data, Bioinformatics, Cloud Computing, Communication, Communication Protocols, Computational Creativity, Computer Networks, Computer Vision, Cyber Security, Data Science, Databases, Decision Support Systems.