PAPER • OPEN ACCESS

Performance Evaluation of QoS metrics in Software Defined Networking using Ryu Controller

To cite this article: Himanshi Babbar and Shalli Rani 2021 *IOP Conf. Ser.: Mater. Sci. Eng.* **1022** 012024

PRIME
PACIFIC RIM MEETING
ON ELECTROCHEMICAL AND SOLID STATE SCIENCE
HONOLULU, HI
Oct 6–11, 2024
Abstract submission deadline:
April 12, 2024
Learn more and submit!
Joint Meeting of
The Electrochemical Society
•
The Electrochemical Society of Japan
•
Korea Electrochemical Society

Performance Evaluation of QoS metrics in Software Defined Networking using Ryu Controller

Himanshi Babbar[1], Shalli Rani[2]
[1,2] Chitkara University Institute of Engineering and Technology, Chitkara University, Punjab, India
E-mail: himanshi.babbar@chitkara.edu.in[1], shalli.rani@chitkara.edu.in[2]

Abstract. With assistance from Software Defined Networking (SDN), networks have become more creative to build and maintain over the last few years. The inflexibility of modern network architecture is presenting researchers with a tough achievement. SDN replaces existing inescapable and complicated networks with a creative way of separating the control plane from the data plane and fixing those constraints, namely configurations done manually, monitoring, protection, usability, and functionality. SDN controllers (e.g., POX, RYU, ONOS, OpenDaylight, Floodlight, etc.) have therefore developed as a core for the management of these networks. The performance evaluation of the SDN controller has an outstanding effect on improving the flexibilities and abilities of a network topology of SDN infrastructures. This research summarizes the SDN controller performance tests using criteria (e.g., Bandwidth, Round-Trip Time). This study employs a Mininet emulator to enforce an SDN architecture that comprises an RYU controller with a switching part, one OpenFlow switch, and multiple nodes. The goal is to determine performance QoS (Quality of Service) including Bandwidth, Throughput using the iperf test.

Keywords: Software Defined Networking (SDN), OpenFlow Protocol, SDN Controllers, Iperf, Wireshark, Complexity Analysis.

1. Introduction

All the traditional routers are replaced with SDN. Defined as “SDN is a network framework which involves separating a network control functions from its data forwarding functions centralizing its intelligence and abstracting its underlying architecture from applications and services [1].” The three-layered architecture of SDN comprises of the Application layer, Control layer, and Infrastructure layer. The application layer in SDN architecture is writing our applications running on physical or virtual hosts like firewalls, packet gateway, load balancer, custom forwarding engine, etc [2]. The Control layer works as the brain of the entire network consists of multiple hardware components and all these hardware components are connected to the central controller, therefore, this controller contains the control layer as these hardware’s are connected to the network because these are the blanket hardware which doesn’t have the software inbuilt known as blind or open hardware. There are two interfaces: Northbound API (talks with the application layer) and Southbound API (talks with the infrastructure layer) [3].

The task of NBI is written inside the application, compile the application to the corresponding network protocol, and map it to one of the modules that are there inside the controller. Therefore, users can utilize this module to write their programs. Example: users can write the load balancing application where the usage of layer-2 and layer-3 module forwards the packets to the final destination [4]. The role of SBI is the conversion of the control layer to the rules. From this individual network, functionality users have to convert it to the rule whenever the new packet arrives the rule will be executed. Inside a router, there are two levels of abstraction one is the control plane and the other is the data plane. The control plane (CP) implements the routing functionalities and construction of the routing table and its management. The data plane (DP) task is to forward the packets by looking into the destination IP (Internet Protocol) header and making the match with the routing table. The local copy of the routing table inside the interfaces that is forwarding the information and then forward the packets to the outgoing interfaces [5].

Control and Data functionalities traditionally are implemented in a single router whenever we are implementing the control and data plane functionalities in a single router then the complexity of control functionality becomes higher because we have multiple routers with their control planes [6]. These control planes need to coordinate with each other to generate or manage the routing table and this control needs to be performed in a distributed way, because of its architectural limitations. The idea of SDN is to separate all the control plane's from the routers and make a centralized control plane in [7] therefore we take out the brains from the individual routers because it makes the decisions and TCAM hardware just do the process of forwarding and place it in a centralized place which is our inter route controller.

(i) **Control Plane:** "The module which takes all the decisions taken by the application designer or network support team."

(ii) **Data Plane:** "The module which carries out all the tasks given by the control plane. The main task is the forwarding of packets. This plane will come from the vendors therefore, now vendors will deliver a dumb switch which has the TCAM hardware along with the forwarding logic."

As shown in above Figure 1, to interact between Network Operating System (NOS) and corresponding packet forwarding hardware, users require an open interface to the hardware so that users don't depend on the corresponding vendors to program our network they require an open interface to the hardware. Facilitates an open API for the application development so that any developer can develop an application [8]. Users require an extensible operating system to convert the program to rules, therefore this application is not just the programs, from these programs' users need to map it to the corresponding rules which are executed at TCAM hardware of the packet forwarding engine which is there inside the switches. The entire paper is organized as section-2 studies the literature of the previous years, section-3 explains the OpenFlow protocol and SDN Controllers in brief, section-4 discusses the methodology, section-5 demonstrates the implementation and result in an analysis of the methodology discussed, section-6 throws light on the complexity of topology and finally section-7 concludes the paper.

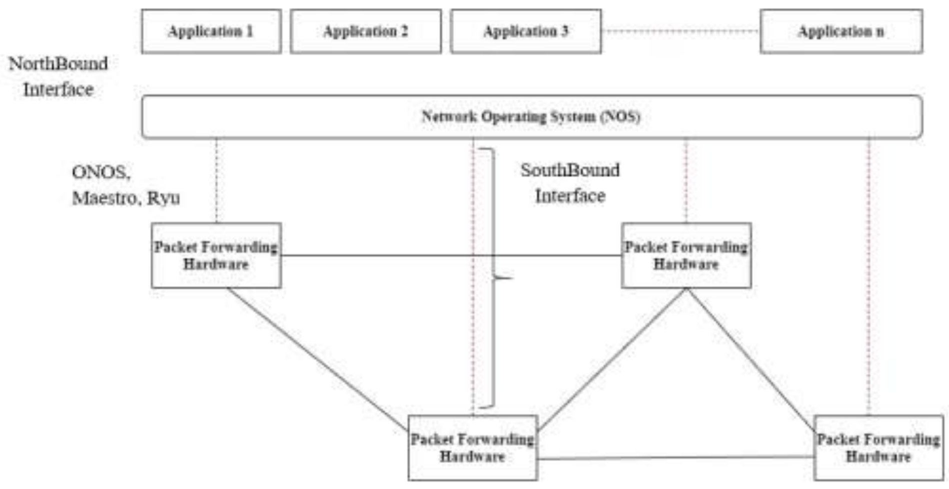

Application 1
Application 2
Application 3
Application n
NorthBound Interface
Network Operating System (NOS)
ONOS, Maestro, Ryu
SouthBound Interface
Packet Forwarding Hardware
Packet Forwarding Hardware
Packet Forwarding Hardware
Packet Forwarding Hardware

Figure 1. SDN based approach to open the innovation

2. Related Work

With the growing trend in SDN, many recent works in load balancing have been taken into consideration for research and experimental purposes. The short description of recent works is shown below and in Table 1. The authors of [9] have proposed the fat-tree topology for load balancing in SDN using the two SDN controllers and are executed in a mininet emulator. The analysis shows that 50% of improvement happened in finding the average load and 41% of improvement is predicted in average delay in the QoS metrics using ping, response time, etc. [10] explained the analysis performed on the controller of SDN that is based on the Throughput, Response time, etc. For execution authors used a mininet emulator which contains a Ryu controller to evaluate the performance based on parameters including RTT, jitter, delay, etc. Demonstration of various load balancing techniques in [11] was performed on a single switch topology that includes Round Robin, Weighted Round Robin, Least Connections, weighted Least Connections, and Random load balancing techniques are compared. In this mininet and Pox, the controller is used for experimentation. In [12] the authors have compared the methods of active and passive network state collection where Load balancing in SDN is running at the controller. A comparison is done through mathematical operations and emulation setup. Authors of [13] proposed the OpenFlow architecture based on the small campus network for the customized model of the network is accomplished by implementing the test analysis based on various parameters includes bandwidth, response time, etc. and code for the creation of network topology is done by the use of python script.

Table 1. Literature Studied

Ref. No.	Algorithm Used	Pros	Cons
[14]	Multipath routing algorithm	Bandwidth utilization improved, achieve better traffic distribution.	Routing overhead
[15]	Load balancing mechanism in the multi-controller.	an improvement over traffic distribution	Resources inefficient.
[16]	Load balancing technique	Improved balancing of load, better QoS, improved traffic segregation	Overhead, Delay in the network.
[17]	Simple path algorithm	Exchange of packets, packet loss minimized, load balancing improved	Throughput and delay.
[18]	Heuristic approach optimization	Minimized response time, maximize throughput	Shortage of Reliability, scalability.
[19]	Honey Bee Foraging algorithm	Performance of the system improved	Overhead.
[20]	QoS aware load balancing	Overload minimized, improved performance of a network	Delay and jitter.

3. Technologies and Controllers

3.1. OpenFlow Protocol

OpenFlow is a protocol used to control the forwarding actions of ethernet switches within an SDN. Initially published through Stanford University's Clean Slate program [21]. The real fact is nowadays inter-networking architecture is becoming open and they are moving from a closed networking innovation to an open innovation networking community where every vendor joins altogether so the vendors are building their hardware and the community is building the opensource operating system and the interface to interact with the corresponding hardware [22]. It is useful in 2 ways:

- It makes the innovation specific or rapid because this inter software is open to the community and users can design their own network protocol on the open-source operating system and paste on the hardware for these users don't require to search for the hardware [23].

- Network administrators don't bother about reading 1000's manuals from 3 different vendors. They just concentrate on a specific operating system and write their own rules on the top of a specific network operating system.

3.1.1 OpenFlow client i.e. software layer and hardware layer. From the hardware layer, source MAC is started i.e. it is a wildcard character that means the client can accept any source MAC field or any destination MAC field, as explained in Figure 2 source IP or destination IP has to be 128.9.1.10, and the source and destination port can be anything. If this is the case then corresponding action forwards the packets to the eth3 [24]-[25]. The entire packet forwarding behavior users can write it as match and action pair, therefore, users have the rule that has the component of match entries in a table. If there is a match with the rule, users can execute the corresponding action and forward it to the destination field which means this particular route where users want to forward the packet, most of the internet networking protocols can implement in the form of match action pair as shown in below Figure 3.

Rule and action are defined as a rule specify what us an interesting value of the specific field inside the packet header like switch port, VLAN, MAC source and destination, ethernet type, etc. if there is a match in the rule means certain fails that users are specifying that there can be set of actions [14]. Actions mean forwarding the packets to zero and more ports in the switch [25]. It encapsulates and forwards the packets, therefore modifies certain fields in the packet header and then forwards the packet. Lastly, drop the packets if users want to implement the firewall loop then our extension can be added.

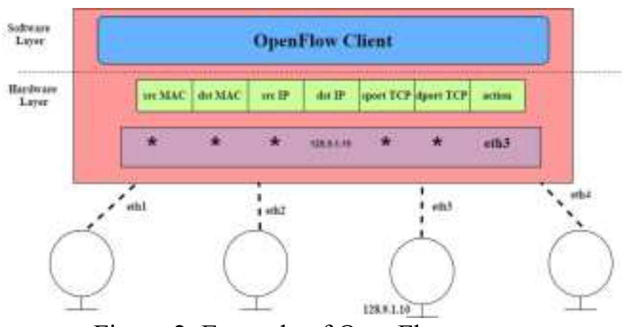


Figure 2. Example of OpenFlow

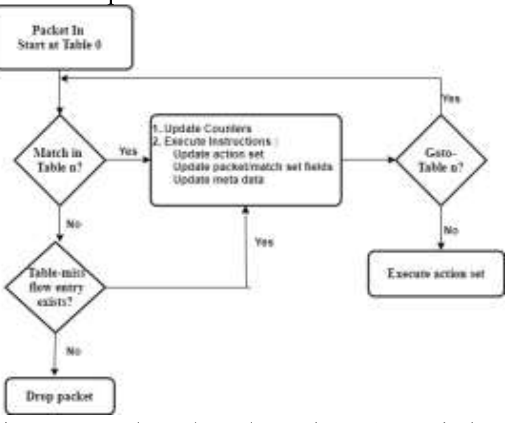


Figure 3. Packet Flow through an OF switch

3.1.2 Flow tables entries are classified by match fields and priorities, i.e. match fields and priorities are bound together to define a specific flow entry in a particular flow table. Flow tables include:
• **Switching**: It includes the MAC destination. If there is a match with the MAC destination field then the packet is forwarded to eth2. It behaves like a normal L2 mechanism.
• **Firewall**: If the user wants to execute the firewall, first look into the TCP destination port. If the TCP destination port is 22 then just drop the packet.
• **Forwarding**: If the user wants to execute the forwarding of the packets, firstly look into the IP destination. If IP belongs to 202.2.*.* then forward the packet to eth2.
• **Flow Switching**: With the help of flow tables users can make the convergence between the packet and circuit switching network. Therefore, the idea of packet switching is to choose a specific path for a specific flow.

• **Source Routing**: If the packet is coming from source 16.2.3.* and if the destination is 202.2.*.* the action is forward the packet to eth2.
• **VLAN Switching**: Given a set of packets, if the user wants to send the packets to a specific destination then packets can be forwarded to multiple ports of the switch which constructs VLAN. If VLAN ID=2 then the packet is forwarded to eth2 and eth3.

3.1.3 Group Tables with the normal OpenFlow rule users can achieve many of the operations like load balancing. For the same destination, there is a need to forward the packets on two different paths which are not possible in load balancing existing flow tables. Group tables comprise of two components: Group type and Bucket. **Group type** defines how the buckets will be handled and discuss the behavior of the group, for example, ALL, SELECT, etc. **Bucket** explains that each bucket will have a separate list of actions and parameters, for example, Bucket 1 output to port 1, bucket 2 output to port 2, etc.

3.2 SDN Controllers
Controllers in SDN are coined as the "brain" of the controller network and it is considered as the NOS (Network Operating System). It coordinates and manages the flow of control to the routers or switches below (via Southbound Interfaces) and the business logic applications above (via Northbound Interface) to under vent the intelligent networks [25]. The control plane in SDN Architecture is known as the SDN controller. It lies between two devices of the network, at one device it's a network device and at the other end it's applications. Their direct communication must pass through the controller. There are various variety of controllers available in SDN: NOX, POX, Ryu, OpenDaylight, Floodlight, and there are various other controllers other than the listed above include Beacon, NOX, Maestro, FlowVisor, RouteFlow, and many others. **POX** is a Python-based open-source platform SDN control application, that includes OpenFlow SDN Controllers. It enables high pace development and prototyping. **NOX** is an open-source C++ based platform SDN control applications. POX is a Python-based platform and is becoming more common as compared to NOX. **RYU** is defined as the software-defined framework of the network that is component-based. It gives the software components that have well-described API's. It is a Python-based platform and in Apache 2.0 license each code is freely available. **OpenDayLight** is a platform (open source) that increases the SDN by giving us the industry-supported framework for the SDN OpenDaylight platform. It is available free which includes all the users and customers. **FloodLight** is an open-source and Java-based platform to create the applications. It follows the REST API's interface that makes it so easy to interface the program with the specified product.

4. Methodology
For the creation of network topology users can emulate a network in a single machine. There is a platform that is an SDN based tool [26]. It is an emulated network in a computer which is independent of the existing network and can be set up as required. Users can emulate these **components** using **mininet**:
(i) A virtual namespace for the legacy network.
(ii) OpenVSwitch, a tool that provides switch implementation in an open-source platform.
(iii) Virtual Namespace, an instance of the protocol stack which works like the individual host.
(iv) Virtual link

Various commands of mininet have been defined for creating the network topology in Figure 4. Python scripts are the language used at the backend for the mininet topology using the Ryu controller [8]. Using this mininet emulation tool the custom networking topology was designed with the depth=2 and fanout=3 means the three-layer of switches are created. The total number of 9 hosts and 4 switches connected to the controller with each switch comprises of three hosts. The parameters used for the execution environment in Table 2 which has a built-in basic structure:

Table 2. Simulation Parameters

Parameter	Value	Description
topo	Tree,depth=2,fanout=3	The topology of 9 hosts and 4 switches with two layers of switches and each switch connected with 3 hosts.
Mac	Nill	Sets the Host's MAC address periodically.
Switch	OVSK	Uses OpenVSwitch.
Controller	Remote	Uses Ryu OpenFlow Controller.

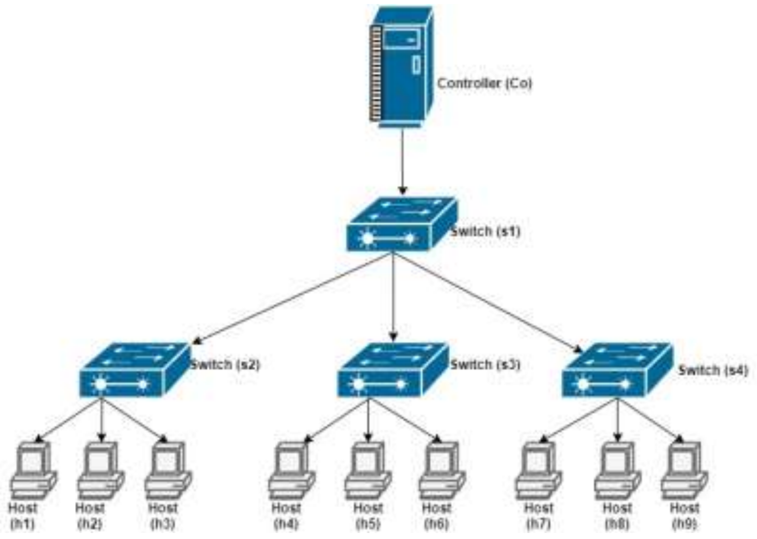
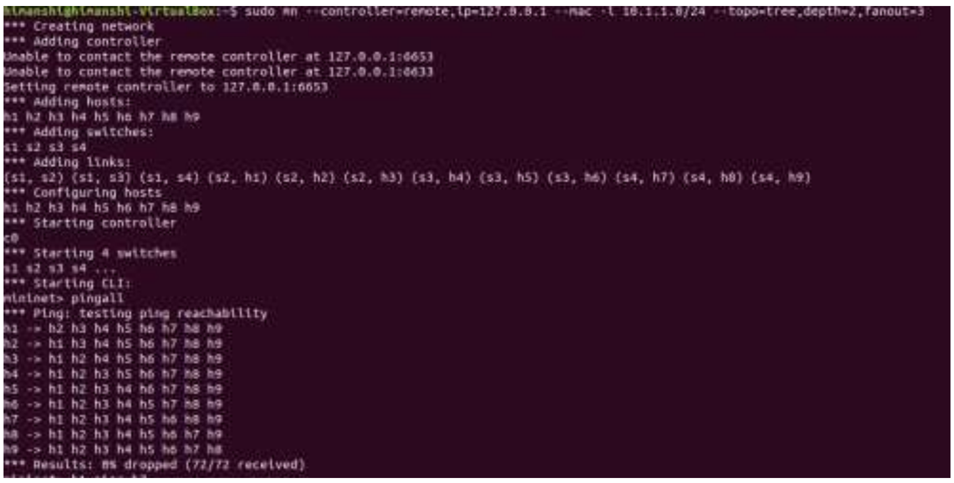


Figure 4. Customized tree topology for SDN

TCP tests are used to measure or test the bandwidth between the links of hosts. Latency, Jitter, Packet loss is measured with the use of UDP tests. In this paper, the researcher needs to measure the bandwidth link from h1 to h4, which states how much traffic needs to upload. For testing, the researcher needs to send the traffic from h1 and to be received by h4. The creation of topology is given in Figure 5 below:



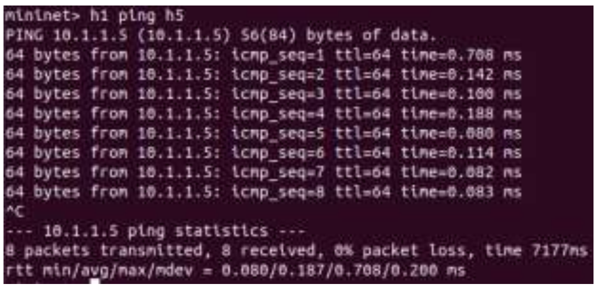
```
himanshi@himanshi-VirtualBox:~$ sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24 --topo=tree,depth=2,fanout=3
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(s1, s2) (s1, s3) (s1, s4) (s2, h1) (s2, h2) (s2, h3) (s3, h4) (s3, h5) (s3, h6) (s4, h7) (s4, h8) (s4, h9)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9
h2 -> h1 h3 h4 h5 h6 h7 h8 h9
h3 -> h1 h2 h4 h5 h6 h7 h8 h9
h4 -> h1 h2 h3 h5 h6 h7 h8 h9
h5 -> h1 h2 h3 h4 h6 h7 h8 h9
h6 -> h1 h2 h3 h4 h5 h7 h8 h9
h7 -> h1 h2 h3 h4 h5 h6 h8 h9
h8 -> h1 h2 h3 h4 h5 h6 h7 h9
h9 -> h1 h2 h3 h4 h5 h6 h7 h8
*** Results: 0% dropped (72/72 received)
```

Figure 5. SDN tree topology creation in mininet

The analysis of the network using the ping command is by analyzing the number of bytes and ICMP sequence packets sent in how much time. Ping command is used to test the reachability between the two nodes, as given in the below Figure 6 h1 pings h5 i.e. 10.1.1.5 IP address and send 64 bytes of data with the time 0.708 ms. In the below-given Figure 8 ovs-ofctl has been connected and handshake has been performed, the table-miss entry has been added and the switching hub is in the status waiting for Packet-In. ICMP echo reply has returned normally. Check the flow table in s1 the table miss entries of the flow table s1 priority 1 is identified:

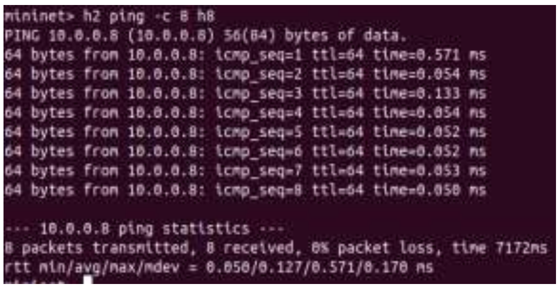
(i) Receive (in-port): “s1-eth1”, target MAC address (dl tar): host 4 → Actions (actions)=output: “transfer to ethernet port 2”.

(ii) Receive (in-port): “s1-eth2”, target MAC address (dl tar): host 1 → Actions (actions)=output: “transfer to ethernet port 1”.



```
mininet> h1 ping h5
PING 10.1.1.5 (10.1.1.5) 56(84) bytes of data.
64 bytes from 10.1.1.5: icmp_seq=1 ttl=64 time=0.708 ms
64 bytes from 10.1.1.5: icmp_seq=2 ttl=64 time=0.142 ms
64 bytes from 10.1.1.5: icmp_seq=3 ttl=64 time=0.100 ms
64 bytes from 10.1.1.5: icmp_seq=4 ttl=64 time=0.188 ms
64 bytes from 10.1.1.5: icmp_seq=5 ttl=64 time=0.080 ms
64 bytes from 10.1.1.5: icmp_seq=6 ttl=64 time=0.114 ms
64 bytes from 10.1.1.5: icmp_seq=7 ttl=64 time=0.082 ms
64 bytes from 10.1.1.5: icmp_seq=8 ttl=64 time=0.083 ms
^C
--- 10.1.1.5 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7177ms
rtt min/avg/max/mdev = 0.080/0.187/0.708/0.200 ms
```

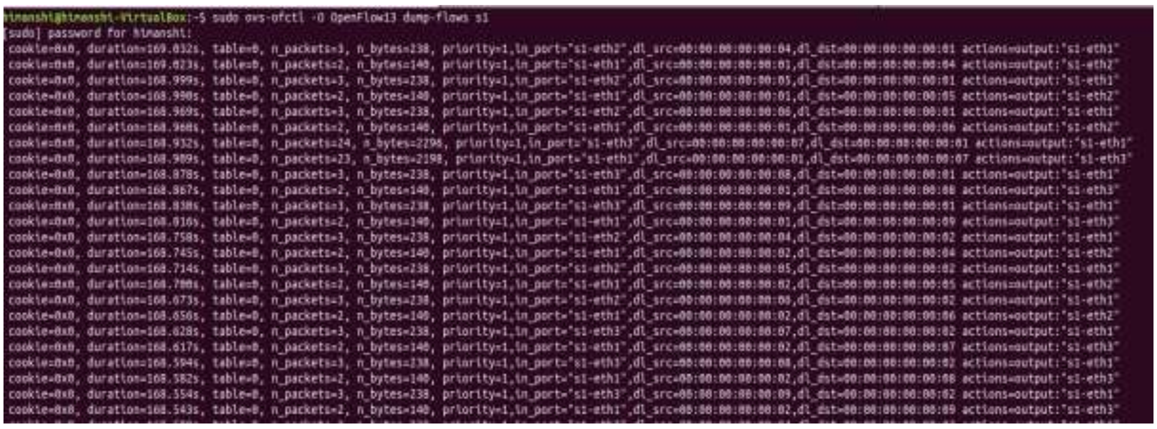
Figure 6. Reachability between h1 and h5



```
mininet> h2 ping -c 8 h8
PING 10.0.0.8 (10.0.0.8) 56(84) bytes of data.
64 bytes from 10.0.0.8: icmp_seq=1 ttl=64 time=0.571 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=0.054 ms
64 bytes from 10.0.0.8: icmp_seq=3 ttl=64 time=0.133 ms
64 bytes from 10.0.0.8: icmp_seq=4 ttl=64 time=0.054 ms
64 bytes from 10.0.0.8: icmp_seq=5 ttl=64 time=0.052 ms
64 bytes from 10.0.0.8: icmp_seq=6 ttl=64 time=0.052 ms
64 bytes from 10.0.0.8: icmp_seq=7 ttl=64 time=0.053 ms
64 bytes from 10.0.0.8: icmp_seq=8 ttl=64 time=0.050 ms

--- 10.0.0.8 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7172ms
rtt min/avg/max/mdev = 0.050/0.127/0.571/0.170 ms
```

Figure 7. Echo request between h2 and h8

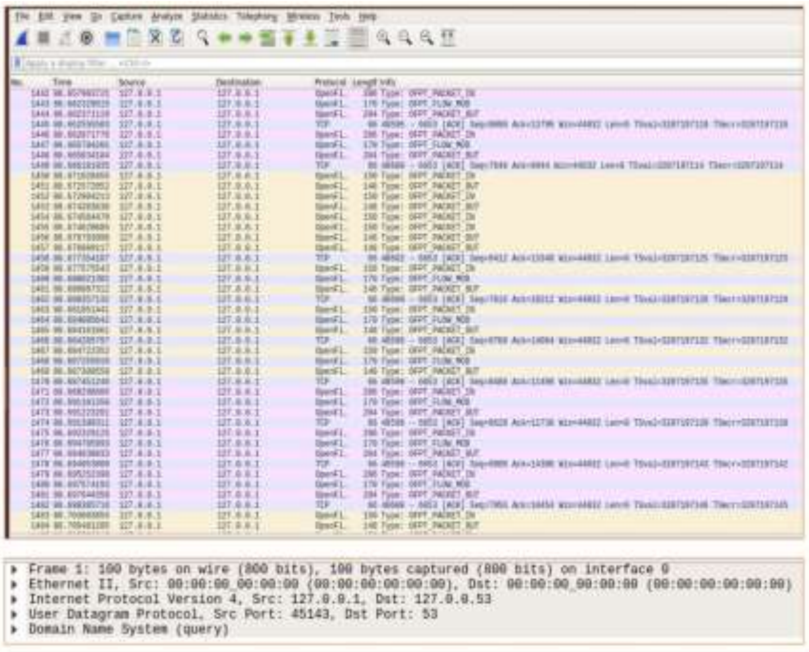


```
himanshi@himanshi-VirtualBox:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
[sudo] password for himanshi:
 cookie=0x0, duration=169.032s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=169.023s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:04 actions=output:"s1-eth2"
 cookie=0x0, duration=168.999s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=168.990s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:05 actions=output:"s1-eth2"
 cookie=0x0, duration=168.969s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=168.960s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:06 actions=output:"s1-eth2"
 cookie=0x0, duration=168.932s, table=0, n_packets=24, n_bytes=2296, priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:07,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=168.909s, table=0, n_packets=23, n_bytes=2198, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:07 actions=output:"s1-eth3"
 cookie=0x0, duration=168.878s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:08,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=168.867s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:08 actions=output:"s1-eth3"
 cookie=0x0, duration=168.838s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:09,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=168.816s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:09 actions=output:"s1-eth3"
 cookie=0x0, duration=168.758s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth1"
 cookie=0x0, duration=168.745s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:04 actions=output:"s1-eth2"
 cookie=0x0, duration=168.714s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth1"
 cookie=0x0, duration=168.700s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:05 actions=output:"s1-eth2"
 cookie=0x0, duration=168.673s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth1"
 cookie=0x0, duration=168.656s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:06 actions=output:"s1-eth2"
 cookie=0x0, duration=168.628s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:07,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth1"
 cookie=0x0, duration=168.617s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:07 actions=output:"s1-eth3"
 cookie=0x0, duration=168.594s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:08,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth1"
 cookie=0x0, duration=168.582s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:08 actions=output:"s1-eth3"
 cookie=0x0, duration=168.554s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:09,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth1"
 cookie=0x0, duration=168.543s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:09 actions=output:"s1-eth3"
```

Figure 8. Check flows of s1

4.1. *Wireshark*

Wireshark is defined as the network packet analyzer. Wireshark is a method for the sniffing and review of packets. It gathers local internet traffic and stores the information for offline review. Wireshark manages to capture Ethernet, Bluetooth, Wireless network traffic (IEEE.802.11), Token Ring, packet switching connections. Wireshark is used for capturing the live data packets from the interface of the network as shown in Figure 9 and is available open-source. There are open files that contain the captured packets of data with the tcpdump. The packets received are OFPT PACKET IN, OFPT PACKET OUT, OFPT FLOW MOD.



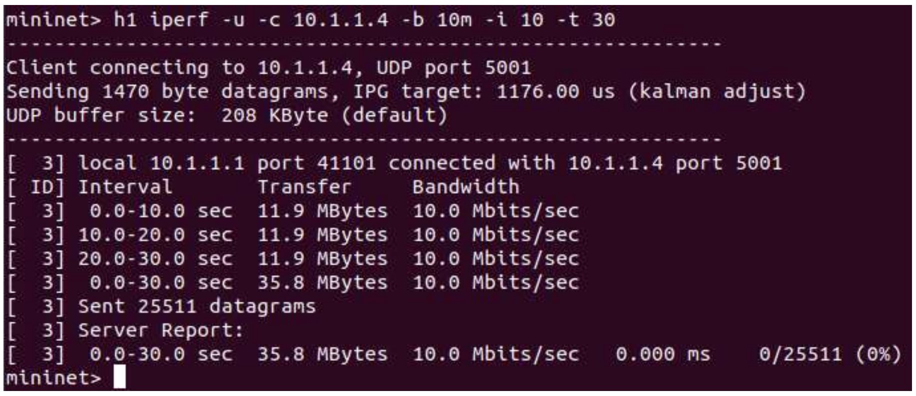
```
▶ Frame 1: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface 0
▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.53
▶ User Datagram Protocol, Src Port: 45143, Dst Port: 53
▶ Domain Name System (query)
```

Figure 9. RYU Capture packets of s1 tree topology in Wireshark

5. Implementation and Result analysis

The main objective is to generate TCP traffic from h1 to h4 by measuring the bandwidth, therefore here h1 is the sender and h4 is the receiver.

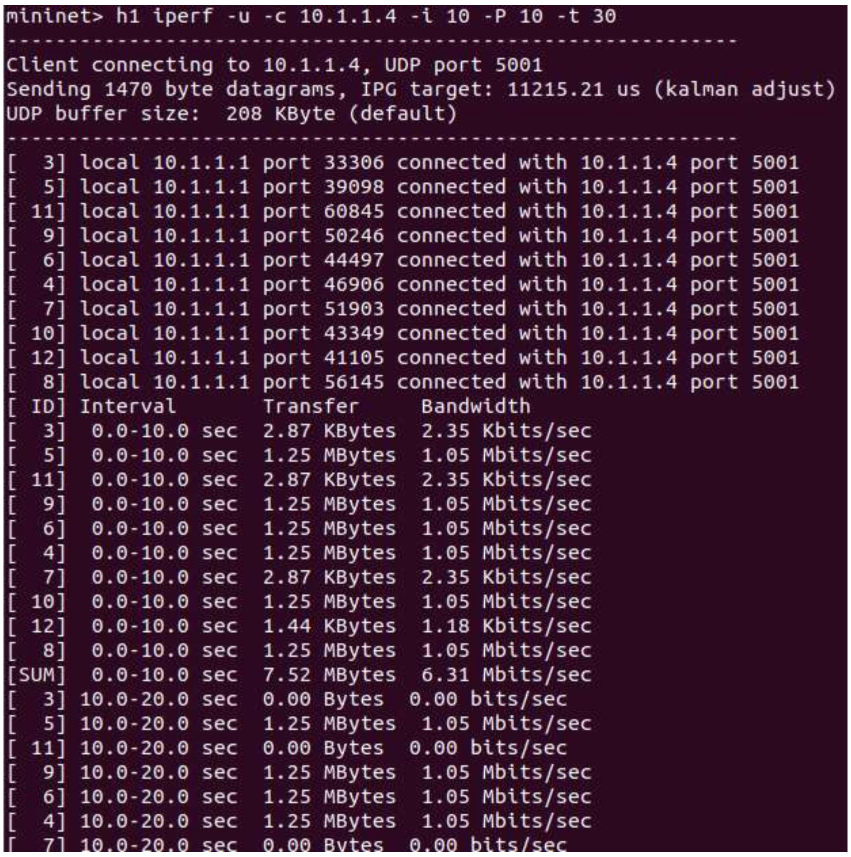
Run Iperf TCP server and client in h4 and h1 respectively Iperf is the most widely used and generates testing the traffic and performance testing in the TCP or UDP network. -c is the TCP client, 10.1.1.4 IP address of h4, -i 10 is the total reporting time, -t 30 is the total test duration. Iperf server and client are starting respectively. The h4 iperf TCP server in Figure 10 is listening to the port 5001 and the size of the window is 85.3 KByte. The h1 iperf TCP client in Figure 10 is connecting to 10.1.1.4 (h4) with the TCP port 5001. ID=3 is taking the total interval time 10 seconds while transferring 22.1 GBytes of data and the amount of data transmitted(Bandwidth) is 19.0 Gbits/sec, the next 10 seconds client is transferring 22.6 GBytes of data in 19.4 Gbits/sec and so on till it completes 30 seconds where the total number of bytes are 71040926664 which is divided by 1024 to get in KB, MB and GB which gets out to be 66.1 GBytes of data which is transferred by the client, the bandwidth is calculated as the number of bytes*8 i.e 71040926664*8 which gives the number of bits per second to get into KB, MB, GB divide by 1000 and final convert into bytes for 30 sec divide the GB by 30 which gives the bandwidth 18.9 Gbits/sec. For the traffic acknowledgment according to the dump-flows, h1 is sending 18.9 Gbps to h4 and h4 is sending 0.08Gbps to h1. According to the perspective of dump-ports, RX pkts=1610356, bytes=71040951139, and TX pkts=1308467, bytes=86373245 h1 is generating higher bandwidth traffic 71040951139 number of bytes port 1 receives, the port bandwidth that h1 has to send to h4 is 66.1 GBytes.



```
mininet> h1 iperf -u -c 10.1.1.4 -b 10m -i 10 -t 30
------------------------------------------------------------
Client connecting to 10.1.1.4, UDP port 5001
Sending 1470 byte datagrams, IPG target: 1176.00 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.1.1.1 port 41101 connected with 10.1.1.4 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  11.9 MBytes  10.0 Mbits/sec
[  3] 10.0-20.0 sec  11.9 MBytes  10.0 Mbits/sec
[  3] 20.0-30.0 sec  11.9 MBytes  10.0 Mbits/sec
[  3]  0.0-30.0 sec  35.8 MBytes  10.0 Mbits/sec
[  3] Sent 25511 datagrams
[  3] Server Report:
[  3]  0.0-30.0 sec  35.8 MBytes  10.0 Mbits/sec   0.000 ms    0/25511 (0%)
mininet>
```

Figure 10. Iperf TCP server in h4 and client in h1

Run iperf UDP server in h4 iperf server in Figure 12 is listening on the UDP port 5001 receiving 1470 bytes of datagrams with the buffer size of 208 Kbyte. Run iperf UDP client in h1 -u: UDP, -c: Client, 10.1.1.4: IP address of h4, -b 10m: bandwidth internal time and transfer 10 Mbps data. If bandwidth is not specified it is by default taken as 1mbps. -P: creates 10 parallel connections and each connection will send 1mbps default value. Iperf client in Figure 11 is connecting to 10.1.1.4 (h4) UDP port 5001 while sending 1470 bytes of datagrams with the buffer size of 208 Kbyte.



```
mininet> h1 iperf -u -c 10.1.1.4 -i 10 -P 10 -t 30
------------------------------------------------------------
Client connecting to 10.1.1.4, UDP port 5001
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.1.1.1 port 33306 connected with 10.1.1.4 port 5001
[  5] local 10.1.1.1 port 39098 connected with 10.1.1.4 port 5001
[ 11] local 10.1.1.1 port 60845 connected with 10.1.1.4 port 5001
[  9] local 10.1.1.1 port 50246 connected with 10.1.1.4 port 5001
[  6] local 10.1.1.1 port 44497 connected with 10.1.1.4 port 5001
[  4] local 10.1.1.1 port 46906 connected with 10.1.1.4 port 5001
[  7] local 10.1.1.1 port 51903 connected with 10.1.1.4 port 5001
[ 10] local 10.1.1.1 port 43349 connected with 10.1.1.4 port 5001
[ 12] local 10.1.1.1 port 41105 connected with 10.1.1.4 port 5001
[  8] local 10.1.1.1 port 56145 connected with 10.1.1.4 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  2.87 KBytes  2.35 Kbits/sec
[  5]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[ 11]  0.0-10.0 sec  2.87 KBytes  2.35 Kbits/sec
[  9]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  6]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  4]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  7]  0.0-10.0 sec  2.87 KBytes  2.35 Kbits/sec
[ 10]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[ 12]  0.0-10.0 sec  1.44 KBytes  1.18 Kbits/sec
[  8]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[SUM]  0.0-10.0 sec  7.52 MBytes  6.31 Mbits/sec
[  3] 10.0-20.0 sec  0.00 Bytes  0.00 bits/sec
[  5] 10.0-20.0 sec  1.25 MBytes  1.05 Mbits/sec
[ 11] 10.0-20.0 sec  0.00 Bytes  0.00 bits/sec
[  9] 10.0-20.0 sec  1.25 MBytes  1.05 Mbits/sec
[  6] 10.0-20.0 sec  1.25 MBytes  1.05 Mbits/sec
[  4] 10.0-20.0 sec  1.25 MBytes  1.05 Mbits/sec
[  7] 10.0-20.0 sec  0.00 Bytes  0.00 bits/sec
```

Figure 11. Iperf UDP client in h1

6. Complexity of tree topology

The time complexity of tree topology is by the use of data structure and its implementation. If the adjacent list of the algorithm defined as R(m) = R(n) + R(m – n – 1) + b, where n is the number of hosts on one side of root and m-n-1 on the other side. Let's analyze the boundary conditions:

Case 1: Skewed tree (One of the subtrees is empty and another subtree is nonempty) n is 0 in this case. R(m) = R(0) + R(m-1) + b R(m) = 2R(0) + R(m-2) + 2b R(m) = 3R(0) + R(m-3) + 3b R(m) = 4R(0) + R(m-4) + 4b . . . . R(m) = (m-1)R(0) + R(1) + (m-1)b R(m) = mR(0) + (m)b, The value of R(0) will be some constant say c. (traversing an empty tree will take some constants time) R(m) = m(b+c) R(m) = (m) (Theta of n).

Case 2: Both left and right subtrees have an equal number of nodes, R(m) = 2R(—m/2—) + b. The complexity of • searching worst-case complexity of O(m), • Insertion worst-case complexity of O(m), • Deletion worst-case complexity of O(m).

7. Conclusion

This paper focuses on the evaluation of network performance through the SDN Controller i.e. Ryu taking into consideration 9 hosts and 4 switches. Results are obtained after the implementation is evaluated by various parameters that are bandwidth and Round-Trip Time (RTT). iperf is used for testing the TCP/UDP server or client for host h1 and host h4 to estimate the performance of the Ryu controller in mininet using the tree topology. In the future, researchers would be focussing on different network topologies like a single, linear, ring, etc. using the different SDN Controllers.

8. References

[1] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for SDN? Implementation challenges for software-defined networks," IEEE Communications Magazine, vol. 51, no. 7, pp. 36–43, 2013.
[2] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, "Software-defined networking: State of the art and research challenges," Computer Networks, vol. 72, pp. 74–98, 2014.
[3] C. Chen-Xiao and X. Ya-Bin, "Research on load balance method in SDN," International Journal of Grid and Distributed Computing, vol. 9, no. 1, pp. 25–36, 2016.
[4] K. Govindarajan and K. C. Meng, ". . Kannan Govindarajan , Kong Chee Meng , Hong Ong!" pp. 293–299, 2013.
[5] D. S. Rana, S. A. Dhondiyal, and S. K. Chamoli, "International Journal of Computer Sciences and Engineering Open Access Software-Defined Networking (SDN) Challenges, issues and Solution," no. February 2019.
[6] H. Babbar and S. Rani, "Emerging prospects and trends in software-defined networking," Journal of Computational and Theoretical Nanoscience, vol. 16, no. 10, pp. 4236–4241, 2019.
[7] A. A. Abdelaziz, E. Ahmed, A. T. Fong, A. Gani, and M. Imran, "SDN-Based load balancing service for cloud servers," IEEE Communications Magazine, vol. 56, no. 8, pp. 106–111, 2018.
[8] Y. Zhou, Y. Wang, J. Yu, J. Ba, and S. Zhang, "Load Balancing for Multiple Controllers in SDN Based on Switches Group," pp. 227–230, 2017.
[9] R. Chaudhary and N. Kumar, "LOADS: Load Optimization and Anomaly Detection Scheme for Software-Defined Networks," IEEE Transactions on Vehicular Technology, vol. 68, no. 12, pp. 12 329–12 344, 2019.
[10] T. Hu, Z. Guo, P. Yi, T. Baker, and J. Lan, "Multi-controller Based Software-Defined Networking: A Survey," IEEE Access, vol. 6, no. c, pp. 15 980–15 996, 2018.
[11] Y. Zhou, K. Zheng, W. Ni, and R. P. Liu, "Elastic switch migration for control plane load balancing in SDN," IEEE Access, vol. 6, pp. 3909–3919, 2018.
[12] J. Cui, Q. Lu, H. Zhong, M. Tian, and L. Liu, "A Load-Balancing Mechanism for Distributed SDN Control Plane Using Response Time," IEEE Transactions on Network and Service Management, vol. 15, no. 4, pp. 1197–1206, 2018.
[13] F. AL-Tam and N. Correia, "Fractional switch migration in multi-controller software-defined networking," Computer Networks, vol. 157, pp. 1–10, 2019.

[14] B. A. He, "Multipath Routing with Load Balancing and Admission Control in SDN," vol. 4, no. c, pp. 4–9, 2016.
[15] Y.-w. M. J.-l. C. Y.-h. Tsai, "Load-Balancing Multiple Controllers Mechanism for Software-Defined Networking," Wireless Personal Communications, 2016.
[16] M. M. Mulla, M. M. Raikar, M. K. Meghana, N. S. Shetti, and R. K. Madhu, Load Balancing for Software-Defined Networks. Springer Singapore, 2019. [Online]. Available: http://dx.doi.org/10.1007/978-981-13-5802- 9 22
[17] V. D. Chakravarthy and B. Amutha, "A novel software-defined networking approach for load balancing in data center networks," International Journal of Communication Systems, no. June, pp. 1–16, 2019.
[18] M. Farhoudi, P. Habibi, and M. Sabaei, "Server Load Balancing in Software-Defined Networks," 9th International Symposium on Telecommunication: With Emphasis on Information and Communication Technology, IST 2018, pp. 435–441, 2019.
[19] B. Kang and H. Choo, "An SDN-enhanced load-balancing technique in the cloud system," The Journal of Supercomputing, 2016.
[20] B. R. Tamma, "PT US CR," Computer Communications, 2016.
[21] T. Hu, P. Yi, J. Zhang, and J. Lan, "Reliable and load balance-aware multi-controller deployment in SDN," China Communications, vol. 15, no. 11, pp. 184–198, 2018.
[22] L. Nkenyereye, L. Nkenyereye, S. M. Riazul Islam, Y. H. Choi, M. Bilal, and J. W. Jang, "Software-defined network-based vehicular networks: A position paper on their modeling and implementation," Sensors (Switzerland), vol. 19, no. 17, pp. 1–14, 2019.
[23] A. Ikram, S. Arif, N. Ayub, and W. Arif, "Load Balancing In Software Defined Networking (SDN)," MAGNT Research Report, vol. 5, no. 1, pp. 298–305, 2018.
[24] H. Zhong, Y. Fang, and J. Cui, "LBBSRT: An efficient SDN load balancing scheme based on server response time," Future Generation Computer Systems, vol. 68, pp. 183–190, 2017.
[25] H. Babbar and S. Rani, "Emerging prospects and trends in software-defined networking," *Journal of Computational and Theoretical Nanoscience*, Vol.16, pp. 4236–4241, 2019.
[26] H. Babbar and S. Rani, "Software-defined networking framework securing internet of things," *Integration of WSN and IoT for Smart Cities, EAI/Springer Innovations in Communication and Computing,* pp. 1-14, 2020.

Acknowledgments
The authors would like to give special thanks to the almighty for his support and blessings and would like to thanks my Respected Teachers, Supervisor, Chitkara University for their valuable guidance and helps me in providing the conducive environment and who helped me so much in the area of research and have learned a lot many things.