Performance Analysis of proposed OpenFlow network using pox controller vs. traditional network

Jyoti Kumari[1], Samir Srivastava[2]

[1]Computer Science and Engineering Department, Kamla Nehru Institute of Technology, Sultanpur

[2]Computer Science and Engineering Department, Kamla Nehru Institute of Technology, Sultanpur

Email: [1]jyotiit4@gmail.com, [2]samir@knit.ac.in

Abstract - Management of the networking system is more complex because of the dynamic nature of current networking system in which control plane is fully distributed in the network. Programmable networking system SDN makes it easy to manage and configure the network by removing the controlling function of the networking elements and placing it at logically centralized control plane. The aim of this research paper is to analyze the performance of SDN for a small network using POX controller and compare it with the conventional network. mininet tool is used to create and analyze the performance of the conventional and programmable network. Our performance analysis is based on latency, throughput, delay, and jitter.

Keyword**-** *SDN; OpenFlow; mininet; POX; Throughput; latency.*

I. INTRODUCTION

In traditional networking system, control-plane is fully distributed. Networking device (e.g. router, switch) has its own control-plane and data-plane as shown in fig. 1. Control-plane have the forwarding policies based on which data-plane forward the packets. When packets arrive at a networking device the embedded firmware tells the hardware, where to forward the packet. Any adjustment in the forwarding policy required to reconfigure the nodes that have their specific interface and network administrator has to manually perform low-level configuration on these vendor-specific networking devices. There is lack of open standard interface which also limits the researchers to easily develop and test their applications. Horizontal scaling of the network is also a very difficult task in conventional networking paradigm. Hence, management of such dynamic and state changing network is a very complex task.

SDN provides a better way to configure and manage the network by decoupling the control-plane from data-plane. It removes the control logic from the networking devices to make it simple data forwarding element (e.g. OpenFlow switch). Control logic of the network is placed at the logically centralized controller (Network Operating System), which provide the abstract view of the network. SDN concept is based on separation between the definition of network policies, implementation of these policies in the hardware devices and traffic forwarding [1]. Fig. 2 explains the simplified architecture of software-defined networking system. SDN has three open API: (1) southbound interface, (2) northbound interface and (3) east-westbound interface (e.g., Flow visor) to handle the communication protocol between logically centralized controllers. Objectives of these interfaces are explained in [2].

OpenFlow protocol is one of the most popular standard protocol and the most commonly deployed SDN technology. It was originally proposed by Stanford University. OpenFlow is defined as the first standard communication protocol between data-plane and control-plane in SDN architecture by open network foundation (ONF) [3].

This paper is organized by discussing the OpenFlow based SDN architecture in section II. Environment setup using the simulation tool and also a brief introduction of the tool is discussed in section III. Section IV describes the proposed OpenFlow network model and traditional network model and their comparative performance analysis is discussed in section V. finally, a conclusion of the topic based on the result obtained and suggestions for the future work is discussed in section VI.

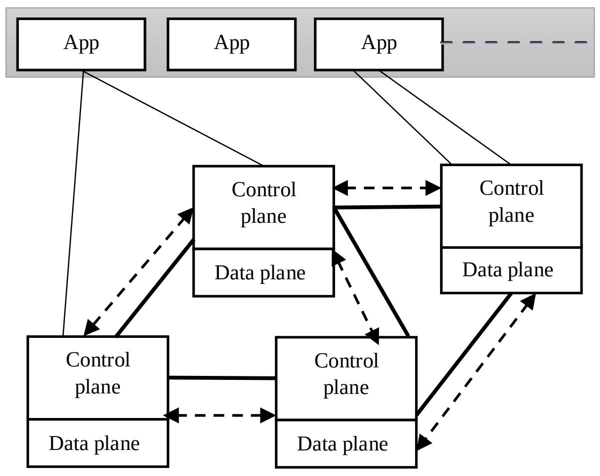


Fig. 1. Simplified Architecture of Traditional Network

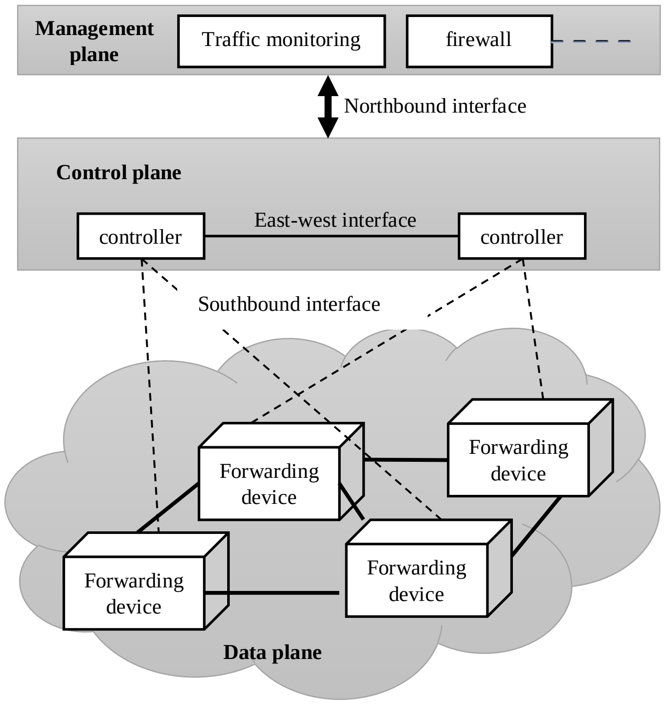


Fig. 2. Simplified Architecture of SDN

II. OPENFLOW BASED SOFTWARE DEFINED NETWORK ARCHITECTURE

OpenFlow is a flow-based protocol that supports to implement SDN concept in hardware and software [4]. Fig. 3 illustrates the OpenFlow-based network architecture. OpenFlow network typically includes three important components: (a) OpenFlow switches, (b) OpenFlow Controller, and (c) OpenFlow protocol.

A. OpenFlow Switch

Switches use the flow table to handle the incoming packets. Flow table contains flow entries that are stored in decreasing order of priority. Each flow entry contains (a) header field, to match against incoming packets, (b) Action (set of zero or more actions), to apply on the packet when header field matched, (c) counter, to keep statistics of the packet. Header file of the incoming packet is compared with header field of each flow entry of the flow table starting from the first. If no match found, the packet is sent to the controller using Packet-in message. Local traffic (traffic to and from the secure channel) is not checked against flow table. OpenvSwitch [5] is most widely used software based OpenFlow switch.

B. Controller

Control plane consists of a central controller or multiple physically distributed but logically centralized controllers. East-West bound interface defines the communication protocol between these logically centralized controllers.

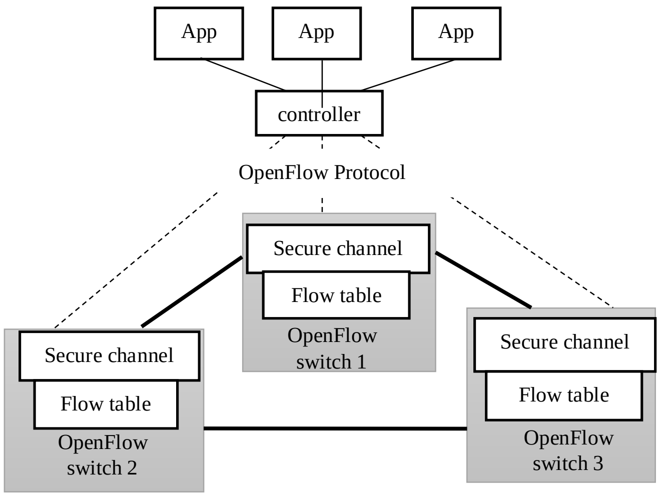


Fig. 3. OpenFlow based network architecture

It gives the abstract view of the application layer. NOX [6] was the first OpenFlow controller, written in C++ and python. POX [7], typically termed as NOX's younger sibling, Ryu [8], floodlight [9], OpenDaylight [10] are some examples of OpenFlow controllers.

C. OpenFlow Protocol

OpenFlow protocol uses the secure channel (TLS/TCP) to establish the connection between the controller and switches. Controller manages, configure and communicate with the switches through this secure channel. OpenFlow protocol support: (1) Controller-to-switch messages, are initiated by the controller to configure, manage or to get the state of switches, (2) Asynchronous messages, are initiated by the switches and sent to the controller when switch state change, error or no flow entry for incoming packet, (3) Symmetric message, are initiated by either controller or switch and sent without any solicitation.

III. SIMULATION TOOL AND EXPERIMENT SETUP

In this section, we discuss a brief introduction to a simulation tool required for designing the layer 2 OpenFlow network and traditional network. Experiment with the physical testbeds is very expansive for researchers. To create a physical testbed for the research of OpenFlow applications, there are many required components (e.g., OpenFlow switches, controller machine, physical infrastructure). Mininet is an open source network emulator which helps to create and run realistic software-based networks, on a single computer system. Mininet uses lightweight virtualization to run multiple switches and hosts on a single operating system (LINUX) kernel. The code we develop and test on mininet, is easily movable to the real network with minimal changes, for deployment, testing, and performance analysis.

The easiest way of mininet installation on the non-Linux operating system, is installing mininet virtual machine in a virtualization software. We experimented with mininet vm of mininet version 2.2.2 on Ubuntu 14.04 LTS, running in Oracle vm VirtualBox, on Windows operating system. Installation steps and setup notes are available in [12]. Putty is used along with the Xming server to make an X11 forwarding enabled ssh connection with mininet vm. Xming server is running on Windows operating system which enables to use X11 applications (e.g., gedit, xterm, Wireshark). To make ssh connection with mininet vm, it is required to get the IP address of mininet vm using the command-

```
$ sudo dhclient eth1
$ sudo ifconfig eth1
```

IV. PROPOSED NETWORK MODEL

In this section, we create an OpenFlow network and a traditional network using python language. To write python code for network designing, it is required to use one editor. In this paper, gedit (a graphical text editor) is installed in mininet using command-

```
$ sudo apt-get install gedit
```

A. *OpenFlow Network:*

We create an OpenFlow network using 10 OpenvSwitch (s1 to s10), 28 virtual hosts (h1 to h28) and an OpenFlow controller pox as shown in figure 4. Each host has a unique IP address. Hosts and switches are connected with virtual ethernet cable of 1000Mbps bandwidth. In OpenFlow network initially flow table of each OpenvSwitch is empty. Remote OpenFlow controller POX (carp branch) controls all 10 switches. It is required to create a component class for the controller that allows the hosts to communicate with each other by implementing learning switch logic.

According to learning switch logic, if a switch receives a packet and switch have no flow rule for the packet then the packet will be sent to the controller. The Controller will store the MAC address of the sender and switch port at which received the packet is received. The packet will be flooded by the controller to get the MAC address and port of the recipient. After that controller will install the flow rule for the sender and the receiver in the flow table of the switch. Component for layer 2 learning switch named learning_sw.py is saved in /pox/ext folder that makes OpenvSwitches act as a type of layer 2 learning switch. To run pox controller, it is required to run the following in new xterm window.

```
/home/mininet/pox/pox.py log.level –DEBUG
learning_sw
```

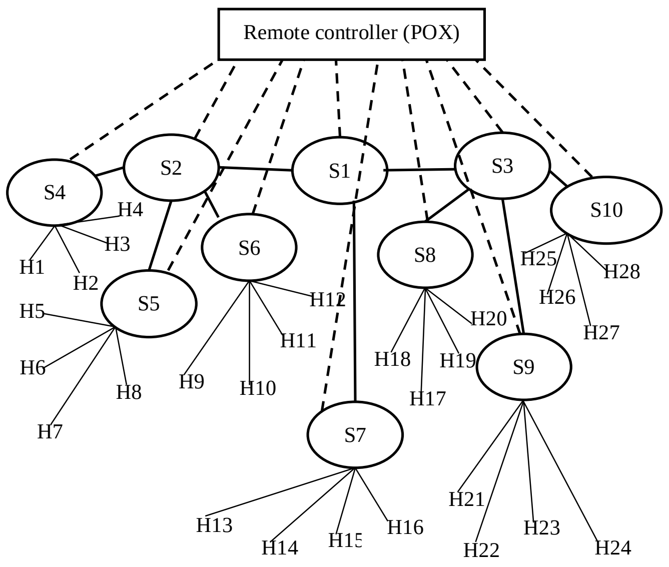


Fig. 4. Proposed OpenFlow based network model

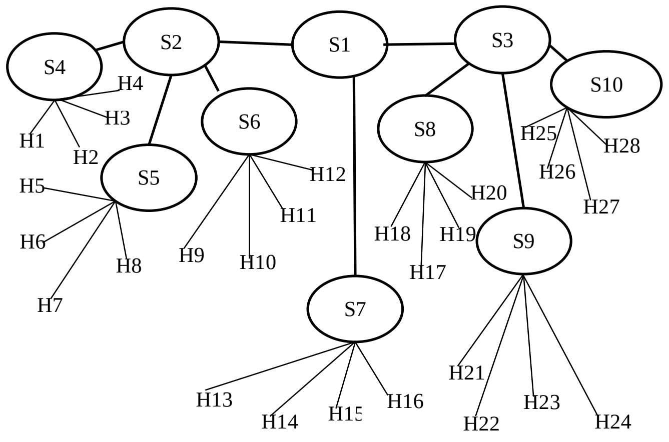


Fig. 5. traditional layer 2 network model

B. *Traditional Network:*

Fig. 5 illustrate the traditional network model in which Linux Bridge (a layer 2 virtual device) is used to create the layer network. To work with Linux Bridge, it is required to install bridge-utils in mininet. Linux Bridge consist of the set of network ports, a control plane, a forwarding plane, MAC learning database. To run the traditional network or the OpenFlow network following command is used-

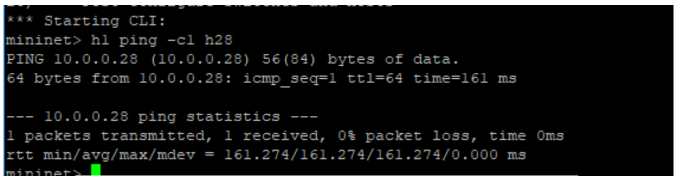
```
$ sudo python <file name>
```

V. RESULT AND ANALYSIS

The main objective of this research paper is to analyze and compare the performance of OpenFlow network using pox controller and traditional network. To accomplish this, we perform network connectivity test and measure throughput of the network.

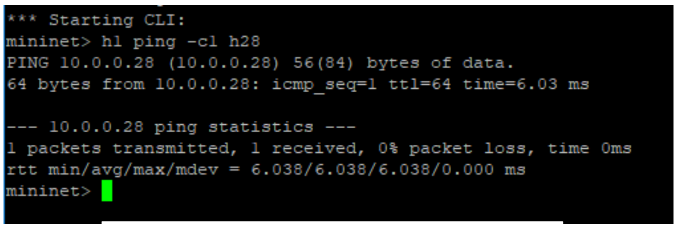
PING is used to test the network connectivity and to measure the latency. Ipref tool is used to generate the traffic and to measure the throughput over a TCP connection and a UDP connection.

A ping test between source node h1 and destination node h28 is performed. In OpenFlow network average rtt (round trip time) for first ping test is 161ms and in traditional network 6.03ms as shown in fig. 6 and fig. 7 respectively. Latency for the first ping test is high in OpenFlow network because flow table is empty and switch send the packet-in message to the controller.



```
*** Starting CLI:
mininet> h1 ping -c1 h28
PING 10.0.0.28 (10.0.0.28) 56(84) bytes of data.
64 bytes from 10.0.0.28: icmp_seq=1 ttl=64 time=161 ms

--- 10.0.0.28 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 161.274/161.274/161.274/0.000 ms
mininet>
```

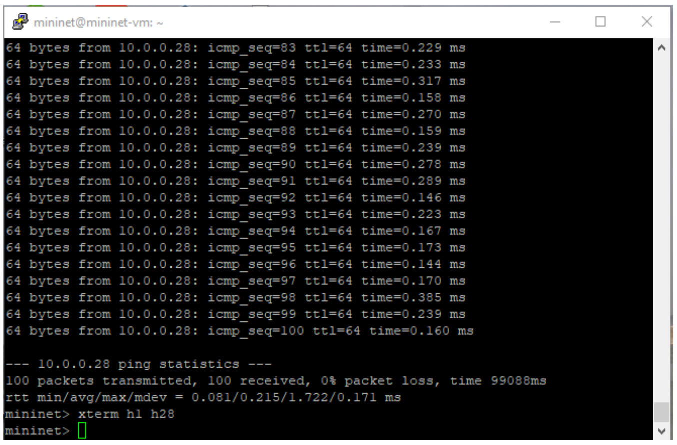
Fig. 6. first ping test in OpenFlow network



```
*** Starting CLI:
mininet> h1 ping -c1 h28
PING 10.0.0.28 (10.0.0.28) 56(84) bytes of data.
64 bytes from 10.0.0.28: icmp_seq=1 ttl=64 time=6.03 ms

--- 10.0.0.28 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 6.038/6.038/6.038/0.000 ms
mininet>
```

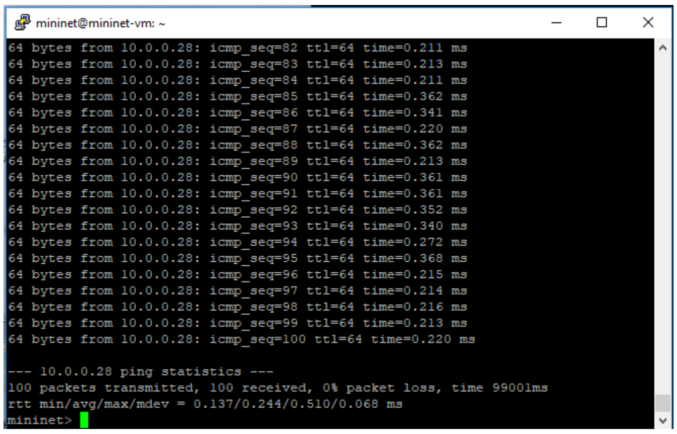
Fig. 7. first ping test in traditional *network*

Pingall command is executed to test the network connectivity between nodes. In pingall, each OpenvSwitch sends ICMP (internet control message protocol) echo request messages to all other OpenvSwitches and wait for responses from them. As shown in fig. 8 and 9, average latency of the OpenFlow network (when flows are installed in switches) is equal to or better than the traditional network.



```
64 bytes from 10.0.0.28: icmp_seq=83 ttl=64 time=0.229 ms
64 bytes from 10.0.0.28: icmp_seq=84 ttl=64 time=0.233 ms
64 bytes from 10.0.0.28: icmp_seq=85 ttl=64 time=0.317 ms
64 bytes from 10.0.0.28: icmp_seq=86 ttl=64 time=0.158 ms
64 bytes from 10.0.0.28: icmp_seq=87 ttl=64 time=0.270 ms
64 bytes from 10.0.0.28: icmp_seq=88 ttl=64 time=0.159 ms
64 bytes from 10.0.0.28: icmp_seq=89 ttl=64 time=0.239 ms
64 bytes from 10.0.0.28: icmp_seq=90 ttl=64 time=0.278 ms
64 bytes from 10.0.0.28: icmp_seq=91 ttl=64 time=0.289 ms
64 bytes from 10.0.0.28: icmp_seq=92 ttl=64 time=0.146 ms
64 bytes from 10.0.0.28: icmp_seq=93 ttl=64 time=0.223 ms
64 bytes from 10.0.0.28: icmp_seq=94 ttl=64 time=0.167 ms
64 bytes from 10.0.0.28: icmp_seq=95 ttl=64 time=0.173 ms
64 bytes from 10.0.0.28: icmp_seq=96 ttl=64 time=0.144 ms
64 bytes from 10.0.0.28: icmp_seq=97 ttl=64 time=0.170 ms
64 bytes from 10.0.0.28: icmp_seq=98 ttl=64 time=0.385 ms
64 bytes from 10.0.0.28: icmp_seq=99 ttl=64 time=0.239 ms
64 bytes from 10.0.0.28: icmp_seq=100 ttl=64 time=0.160 ms

--- 10.0.0.28 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99088ms
rtt min/avg/max/mdev = 0.081/0.215/1.722/0.171 ms
mininet> xterm h1 h28
mininet>
```

Fig. 8. ping test in OpenFlow network



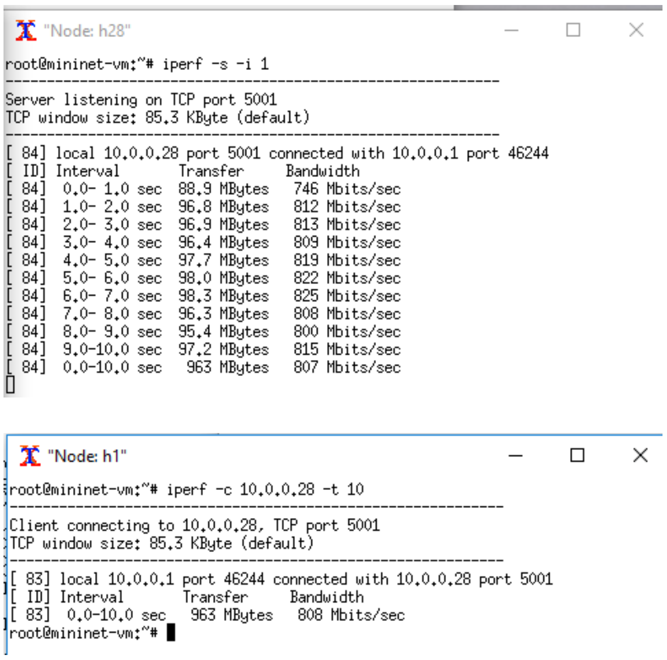
```
64 bytes from 10.0.0.28: icmp_seq=82 ttl=64 time=0.211 ms
64 bytes from 10.0.0.28: icmp_seq=83 ttl=64 time=0.213 ms
64 bytes from 10.0.0.28: icmp_seq=84 ttl=64 time=0.211 ms
64 bytes from 10.0.0.28: icmp_seq=85 ttl=64 time=0.362 ms
64 bytes from 10.0.0.28: icmp_seq=86 ttl=64 time=0.341 ms
64 bytes from 10.0.0.28: icmp_seq=87 ttl=64 time=0.220 ms
64 bytes from 10.0.0.28: icmp_seq=88 ttl=64 time=0.362 ms
64 bytes from 10.0.0.28: icmp_seq=89 ttl=64 time=0.213 ms
64 bytes from 10.0.0.28: icmp_seq=90 ttl=64 time=0.361 ms
64 bytes from 10.0.0.28: icmp_seq=91 ttl=64 time=0.361 ms
64 bytes from 10.0.0.28: icmp_seq=92 ttl=64 time=0.352 ms
64 bytes from 10.0.0.28: icmp_seq=93 ttl=64 time=0.340 ms
64 bytes from 10.0.0.28: icmp_seq=94 ttl=64 time=0.272 ms
64 bytes from 10.0.0.28: icmp_seq=95 ttl=64 time=0.368 ms
64 bytes from 10.0.0.28: icmp_seq=96 ttl=64 time=0.215 ms
64 bytes from 10.0.0.28: icmp_seq=97 ttl=64 time=0.214 ms
64 bytes from 10.0.0.28: icmp_seq=98 ttl=64 time=0.216 ms
64 bytes from 10.0.0.28: icmp_seq=99 ttl=64 time=0.213 ms
64 bytes from 10.0.0.28: icmp_seq=100 ttl=64 time=0.220 ms

--- 10.0.0.28 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99001ms
rtt min/avg/max/mdev = 0.137/0.244/0.510/0.068 ms
mininet>
```

Fig. 9. ping test in traditional network

Table 1. round trip time comparison between OpenFlow and traditional network

Network	minimum	average	maximum
Traditional network	.137ms	.244ms	.510ms
OpenFlow Network	.081ms	.215ms	1.722ms

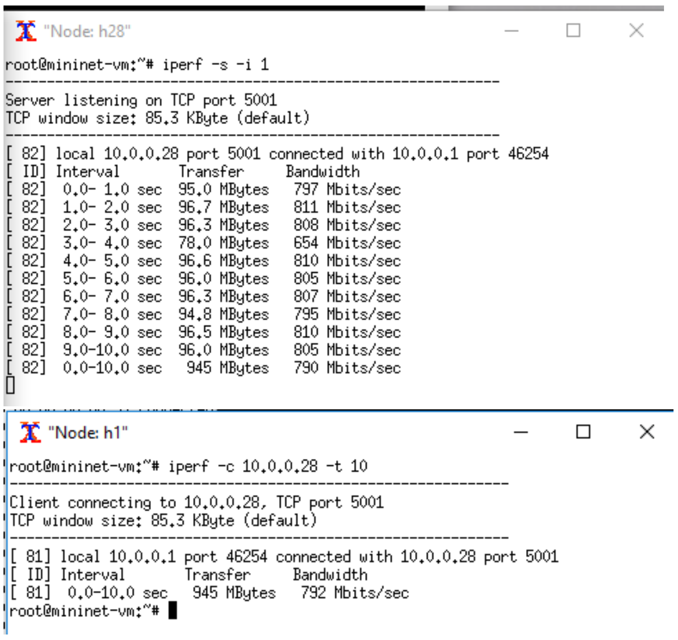
Iperf tool is used to analyze the utilization of bandwidth between source node h1 and the destination node h28 in the networks. We start TCP server at destination host 28 and TCP client at host h1. Commands that are used at the source node and destination node is shown in the screenshot of the result. To analyze the bandwidth utilization data is transferred over TCP connection and for 10 seconds.



```
"Node: h28"
root@mininet-vm:~# iperf -s -i 1
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 84] local 10.0.0.28 port 5001 connected with 10.0.0.1 port 46244
[ ID] Interval       Transfer     Bandwidth
[ 84]  0.0- 1.0 sec  88.9 MBytes   746 Mbits/sec
[ 84]  1.0- 2.0 sec  96.8 MBytes   812 Mbits/sec
[ 84]  2.0- 3.0 sec  96.9 MBytes   813 Mbits/sec
[ 84]  3.0- 4.0 sec  96.4 MBytes   809 Mbits/sec
[ 84]  4.0- 5.0 sec  97.7 MBytes   819 Mbits/sec
[ 84]  5.0- 6.0 sec  98.0 MBytes   822 Mbits/sec
[ 84]  6.0- 7.0 sec  98.3 MBytes   825 Mbits/sec
[ 84]  7.0- 8.0 sec  96.3 MBytes   808 Mbits/sec
[ 84]  8.0- 9.0 sec  95.4 MBytes   800 Mbits/sec
[ 84]  9.0-10.0 sec  97.2 MBytes   815 Mbits/sec
[ 84]  0.0-10.0 sec   963 MBytes   807 Mbits/sec
```

```
"Node: h1"
root@mininet-vm:~# iperf -c 10.0.0.28 -t 10
------------------------------------------------------------
Client connecting to 10.0.0.28, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 83] local 10.0.0.1 port 46244 connected with 10.0.0.28 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 83]  0.0-10.0 sec   963 MBytes   808 Mbits/sec
root@mininet-vm:~#
```

Fig. 10. Throughput over TCP connection in OpenFlow network


"Node: h28"

```
root@mininet-vm:~# iperf -s -i 1
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 82] local 10.0.0.28 port 5001 connected with 10.0.0.1 port 46254
[ ID] Interval       Transfer     Bandwidth
[ 82]  0.0- 1.0 sec  95.0 MBytes   797 Mbits/sec
[ 82]  1.0- 2.0 sec  96.7 MBytes   811 Mbits/sec
[ 82]  2.0- 3.0 sec  96.3 MBytes   808 Mbits/sec
[ 82]  3.0- 4.0 sec  78.0 MBytes   654 Mbits/sec
[ 82]  4.0- 5.0 sec  96.6 MBytes   810 Mbits/sec
[ 82]  5.0- 6.0 sec  96.0 MBytes   805 Mbits/sec
[ 82]  6.0- 7.0 sec  96.3 MBytes   807 Mbits/sec
[ 82]  7.0- 8.0 sec  94.8 MBytes   795 Mbits/sec
[ 82]  8.0- 9.0 sec  96.5 MBytes   810 Mbits/sec
[ 82]  9.0-10.0 sec  96.0 MBytes   805 Mbits/sec
[ 82]  0.0-10.0 sec   945 MBytes   790 Mbits/sec
```

"Node: h1"

```
root@mininet-vm:~# iperf -c 10.0.0.28 -t 10
------------------------------------------------------------
Client connecting to 10.0.0.28, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 81] local 10.0.0.1 port 46254 connected with 10.0.0.28 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 81]  0.0-10.0 sec   945 MBytes   792 Mbits/sec
root@mininet-vm:~#
```

Fig. 11. Throughput over TCP connection in Traditional network

Comparison between the bandwidth of OpenFlow network and traditional network over TCP connection is shown in table 2. Commands that are used at the source and destination nodes are shown in fig. 10 and 11.

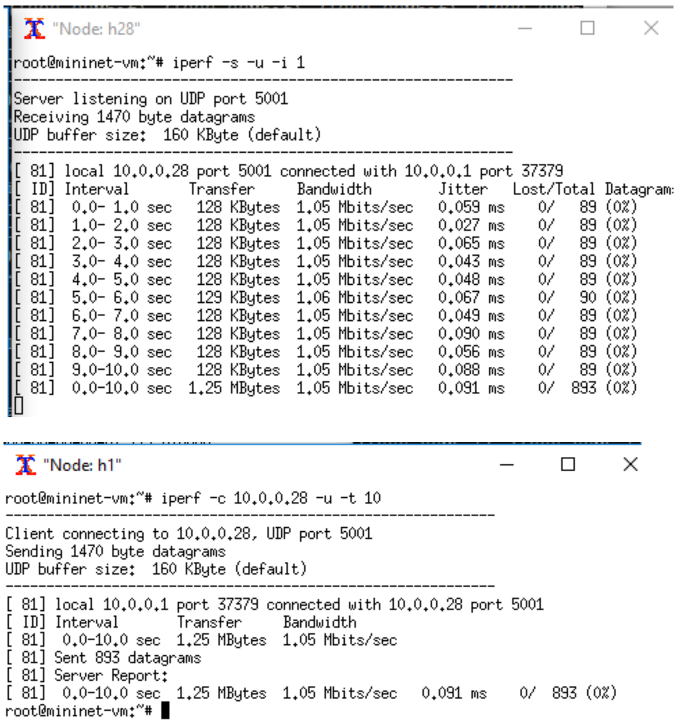
Table 2. Bandwidth comparison between OpenFlow and traditional network for TCP connection

Network	Data transferred	bandwidth
Traditional network	945MByte	792Mbps
OpenFlow Network	963MByte	808Mbps

In UDP connection, out order delivery of packets, jitter, packet loss are some parameters that affect the throughput of the network. For UDP testing, UDP client is started at source node h1 and UDP server is started at destination node h28. Data is transferred for 10 seconds over UDP connection. The result of UDP tests is shown in fig. 12 and 13 and comparison between networks based on server report is shown in table 3.

Table 3. Comparison between OpenFlow and traditional network for UDP connection

Network	Data transferred (MB)	Bandwidth (Mbps)	Jitter (ms)	Loss (%)
Traditional network	1.25	1.05	.091	0%
OpenFlow Network	1.25	1.12	.067	6%

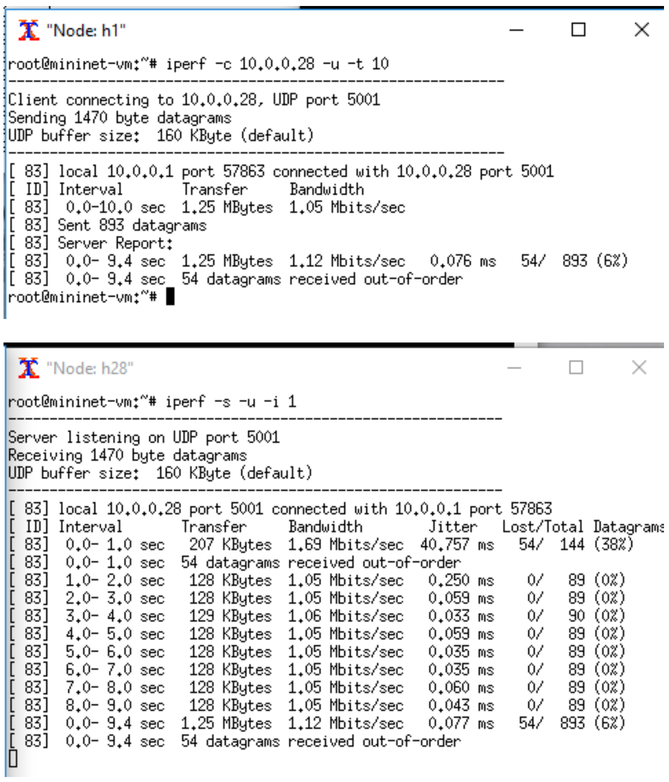

"Node: h28"

```
root@mininet-vm:~# iperf -s -u -i 1
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[ 81] local 10.0.0.28 port 5001 connected with 10.0.0.1 port 37379
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagram
[ 81]  0.0- 1.0 sec   128 KBytes  1.05 Mbits/sec   0.059 ms    0/   89 (0%)
[ 81]  1.0- 2.0 sec   128 KBytes  1.05 Mbits/sec   0.027 ms    0/   89 (0%)
[ 81]  2.0- 3.0 sec   128 KBytes  1.05 Mbits/sec   0.065 ms    0/   89 (0%)
[ 81]  3.0- 4.0 sec   128 KBytes  1.05 Mbits/sec   0.043 ms    0/   89 (0%)
[ 81]  4.0- 5.0 sec   128 KBytes  1.05 Mbits/sec   0.048 ms    0/   89 (0%)
[ 81]  5.0- 6.0 sec   129 KBytes  1.06 Mbits/sec   0.067 ms    0/   90 (0%)
[ 81]  6.0- 7.0 sec   128 KBytes  1.05 Mbits/sec   0.049 ms    0/   89 (0%)
[ 81]  7.0- 8.0 sec   128 KBytes  1.05 Mbits/sec   0.090 ms    0/   89 (0%)
[ 81]  8.0- 9.0 sec   128 KBytes  1.05 Mbits/sec   0.056 ms    0/   89 (0%)
[ 81]  9.0-10.0 sec   128 KBytes  1.05 Mbits/sec   0.088 ms    0/   89 (0%)
[ 81]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.091 ms    0/  893 (0%)
```

"Node: h1"

```
root@mininet-vm:~# iperf -c 10.0.0.28 -u -t 10
------------------------------------------------------------
Client connecting to 10.0.0.28, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[ 81] local 10.0.0.1 port 37379 connected with 10.0.0.28 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 81]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[ 81] Sent 893 datagrams
[ 81] Server Report:
[ 81]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.091 ms    0/  893 (0%)
root@mininet-vm:~#
```

Fig. 12. Throughput over UDP connection in Traditional network


"Node: h1"

```
root@mininet-vm:~# iperf -c 10.0.0.28 -u -t 10
------------------------------------------------------------
Client connecting to 10.0.0.28, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[ 83] local 10.0.0.1 port 57863 connected with 10.0.0.28 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 83]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[ 83] Sent 893 datagrams
[ 83] Server Report:
[ 83]  0.0- 9.4 sec  1.25 MBytes  1.12 Mbits/sec   0.076 ms   54/  893 (6%)
[ 83]  0.0- 9.4 sec  54 datagrams received out-of-order
root@mininet-vm:~#
```

"Node: h28"

```
root@mininet-vm:~# iperf -s -u -i 1
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[ 83] local 10.0.0.28 port 5001 connected with 10.0.0.1 port 57863
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[ 83]  0.0- 1.0 sec   207 KBytes  1.69 Mbits/sec  40.757 ms   54/  144 (38%)
[ 83]  0.0- 1.0 sec  54 datagrams received out-of-order
[ 83]  1.0- 2.0 sec   128 KBytes  1.05 Mbits/sec   0.250 ms    0/   89 (0%)
[ 83]  2.0- 3.0 sec   128 KBytes  1.05 Mbits/sec   0.059 ms    0/   89 (0%)
[ 83]  3.0- 4.0 sec   129 KBytes  1.06 Mbits/sec   0.033 ms    0/   90 (0%)
[ 83]  4.0- 5.0 sec   128 KBytes  1.05 Mbits/sec   0.059 ms    0/   89 (0%)
[ 83]  5.0- 6.0 sec   128 KBytes  1.05 Mbits/sec   0.035 ms    0/   89 (0%)
[ 83]  6.0- 7.0 sec   128 KBytes  1.05 Mbits/sec   0.035 ms    0/   89 (0%)
[ 83]  7.0- 8.0 sec   128 KBytes  1.05 Mbits/sec   0.060 ms    0/   89 (0%)
[ 83]  8.0- 9.0 sec   128 KBytes  1.05 Mbits/sec   0.043 ms    0/   89 (0%)
[ 83]  0.0- 9.4 sec  1.25 MBytes  1.12 Mbits/sec   0.077 ms   54/  893 (6%)
[ 83]  0.0- 9.4 sec  54 datagrams received out-of-order
```

Fig. 13. Throughput over UDP connection in OpenFlow network

We have measured the performance of both the network using ping and iperf. Ping command send the ICMP () echo request message to the specified destination IP address. If the destination is reachable, it replies with ICMP echo reply message. We get the rtt from ping testing. ping command sends one ICMP request message in every second. Initially, we perform our ping test by sending 1 echo request message from the source node h1 to the destination h28 in both networking environment. We have used the reactive approach, so is required to install flow entry for the first packet in the switch. Hence for the very first ping test rtt time in the OpenFlow environment is much more (161.274ms) than rtt in the traditional networking environment that is (6.038ms). we have analyzed the average latency of the network by continuously sending the 100 ICMP packets in both the networking environment. The latency of the OpenFlow network is similar to the traditional networking environment as shown in comparison table 1.

To analyze the maximum bandwidth for TCP connection, and bandwidth, jitter, and packet loss for UDP connection we have used the iperf tool. Iperf creates UDP and TCP data stream to measure the throughput of the network. to measure the bandwidth utilization in both networks between source node h1 and destination node h28, it is required to run h1 in TCP client mode and h28 in server mode. TCP data stream is sent from the client to the server for 10 seconds. Bandwidth comparison between the OpenFlow and conventional network for TCP connection is shown in table 2. We have performed several tested and analyzed that throughput of OpenFlow is similar to the conventional network. we have performed all the analysis using the virtualization software so the networks performances are ultimately depending on the CPU load that may vary.

Similarly, for testing the performance for UDP connection, UDP data stream is sent from source to the destination for 10 seconds. Comparison between the performance of UDP connection is shown in table 3. Packet loss, jitter are some factors that affect the performance of the network for UDP (a connectionless protocol). UDP does not use the bandwidth specified in the TCLink. By default, the bandwidth for UDP connection is 1Mbps. We can also set the bandwidth using -b option. Data loss in OpenFlow network for UDP connection in more as compared to the conventional network.

VI. CONCLUSION AND FUTURE SCOPE

In traditional networking approach, control and data planes are integrated with the networking devices that are difficult to manage and configure. Software-defined networking removes the controlling mechanism from the networking equipment and makes the networking devices a simple forwarding node. These nodes are controlled by the logically centralized controller.

In this paper, a comparative performance analysis of conventional network and OpenFlow enabled software-defined network is done using the mininet network emulator. we have performed network connectivity test in which ping command is used to test the connectivity and to analyze and compare the latency of the network. Based on the obtained result it is concluded that for the first echo request ICMP message the round-trip time of OpenFlow network is much greater than the traditional network. But when flow rules are installed in the switches OpenFlow perform similarly to the traditional network. The Throughput of the OpenFlow network is also better or similar to the traditional network in TCP and UDP connection. Deployment of OpenFlow network will make the networking system programmable, easily manageable, scalable and fast.

REFERENCES

[1] Diego Kreutz, Fernando M. V. Ramos, Paulo Esteve, Christian esteve Rothenberg, Siamak Azodolmolky, Steve Uhlig "Software-Defined Networking: A Comprehensive Survey", proceedings of IEEE | vol. 103, No. 1, January, 2015.
[2] Myung-Ki Shin, Ki-Hyuk Nam, Hyoung-Jun Kim, "Software-Defined Networking (SDN): A Reference Architecture and Open APIs", IEEE, ICTC 2012.
[3] Open Networking Foundation, ONLINE: https://www.Opennetworking.org
[4] Fei Hu, Qi Hao*, Ke Bao, "A Survey on Software-Defined Network (SDN) and OpenFlow : From Concept to Implementation", IEEE communication survey and tutorial, 2013.
[5] OpenvSwitch, ONLINE: https://www.openvswitch.org
[6] Natasha Gude, Ben Pfaff, Teemu koponen, Martin Casado, Scott Shenker, justin Pettit, Nick McKeown, "NOX : Towards an Operating System for Network", Unpublished.
[7] Pox Controller, ONLINE: https:// openflow.stanford.edu/display/ONL/POX+Wiki
[8] Ryu Controller, ONLINE: https://osrg.github.io/ryu/
[9] Floodlight Controller ONLINE: http://www.Projectfloodlight.org.
[10] OpenDaylight controller, ONLINE: https://www.Opendaylight .org/
[11] "OpenFlow Switch Specification version 1.0.0", Open Networking Foundation, December, 2009.
[12] Mininet vm setup steps, ONLINE: http://mininet.org/vm-setup-notes/