Performance Comparison of the Proposed OpenFlow Network with the Pox Controller and the Traditional Network in Software Defined Networks (SDN)

Priya Gautam, Surbhi Jha, Chandra Shekhar Yadav

Noida Institute of Engineering Technology, India

Corresponding author: Priya Gautam, Email: priya10800@gmail.com

The dynamic structure of today's networking system, in which the control plane is fully dispersed throughout the network and networking equipment (e.g., switches, routers) have their own local control-plane and data-plane, makes networking system management more complicated. Networking paradigm that can be programmed by separating the management function from the networking elements and putting it in a logically centralised control plane, SDN makes it simple to administer and operate this dynamic networking system. The separation of the control and data planes facilitates the forwarding devices. These straightforward forwarding devices are managed, controlled, and configured by a logically centralised control-plane. The network operating system is a common term used to describe the control plane of SDN. The purpose of this dissertation is to evaluate the performance of an OpenFlow enabled software defined network model with a POX controller to that of a traditional network. Mininet is a tool that is used to design and analyse traditional and programmable networks. Latency, throughput, packet loss, and jitter characteristics are used to evaluate performance.

Keywords:SDN, OpenFlow, Mininet, POX, Throughput, Latency.

2023. In Saroj Hiranwal & Garima Mathur (eds.), *Artificial Intelligence and Communication Technologies*, 537–552. Computing & Intelligent Systems, SCRS, India. https://doi.org/10.52458/978-81-955020-5-9-52

1 Introduction

The control plane of a typical networking system is totally dispersed. A networking device's control and data planes are independent. The data-plane forwards packets according to the control-forwarding plane's regulations. When packets arrive at a networking device, integrated firmware instructs the hardware where the packet should be forwarded. Any changes to the forwarding policy necessitate reconfiguring the nodes with their own interface, which necessitates the network administrator manually doing low-level setup on these vendor-specific networking devices via the CLI. Researchers are also limited in their ability to design and test their applications due to the lack of an open standard interface. In the traditional networking paradigm, horizontal network scalability is also a tough task. As a result, managing such a dynamic and state-changing network is a difficult undertaking.

By isolating the control and data planes, SDN simplifies the creation and administration of networks. It removes the networking devices' control functionality, resulting in a simple data forwarding element (e.g., OpenFlow switch). The control mechanism for the network is kept in a logically centralised controller (Network Operating System), which gives an abstract image of the network and aids in direct programmability. The SDN concept involves separating network regulation creation, implementation in hardware devices, and traffic forwarding [1]. SDN contains three open APIs to govern the communication protocol across logically centralised controllers: Interfaces for the southbound, northbound, and east-westbound directions (e.g., Flow visor). The goals of these interfaces are described in [2.]

One of the most prominent standard protocols and the most widely used SDN technology is OpenFlow. Stanford University was the first to suggest it. OpenFlow is the first industry-recognized protocol for data-plane and control-plane communication in SDN architecture, according to the open network foundation (ONF) [3].

2 Traditional Networking Architecture

The control-plane, data-planes, and management plane are all integrated with dedicated networking devices in a typical networking system (switch, router). The data-planes are in charge of forwarding incoming packets. The arriving packets are handled by the data-plane according to the controlling function that is configured and stored in the dedicated device's firmware. All arriving packets for the same destination are processed in the same way, according to the firmware's regulations.

Smart switches with application integrated circuits (ASICs) are programmed in such a way that they can recognise and treat various types of packets in different ways. Some networking devices (for example, Cisco routers) can handle distinct types of packets in different ways. The Cisco router also allows you to prioritise flows in order to properly handle traffic. These devices are quite expensive, and their performance is limited when there is a lot of network traffic. The automation of the network is also limited by traditional network architecture. Figure 1 illustrates this. Each networking device has its own data plane and control plane. The current networking system's control method is distributed. It also restricts the network's automation. The entire network is not under centralised control. To obtain the network's overall status, each networking device must synchronise with one another.

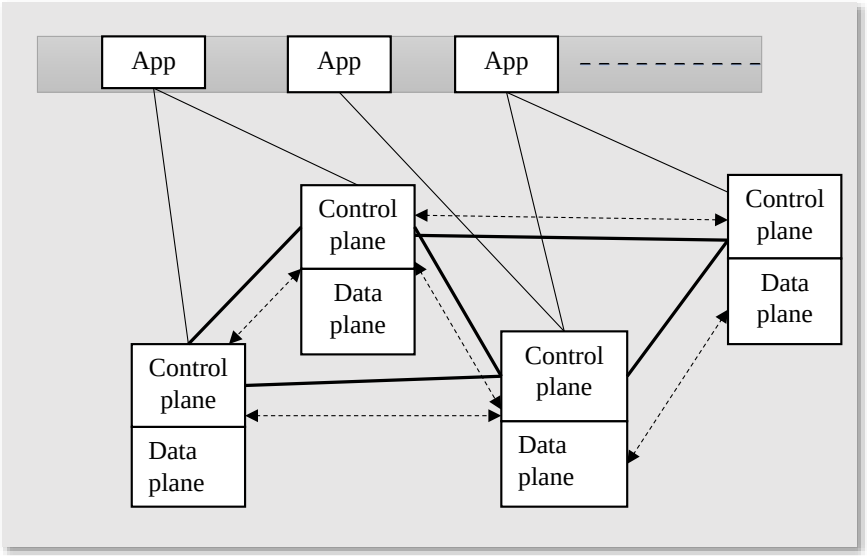


Figure 1. Architecture of Traditional network

3 Software Defined Networking (SDN) Architecture

SDN is a relatively new method to programmable networking. Before SDN and OpenFlow, various methods to programmable networking systems were explored. SOFTNET was one of the first. The commands were inserted to the content of each packet in SOFTNET to change the functionality of a network device. The commands were created using the SOFTNET programming language.

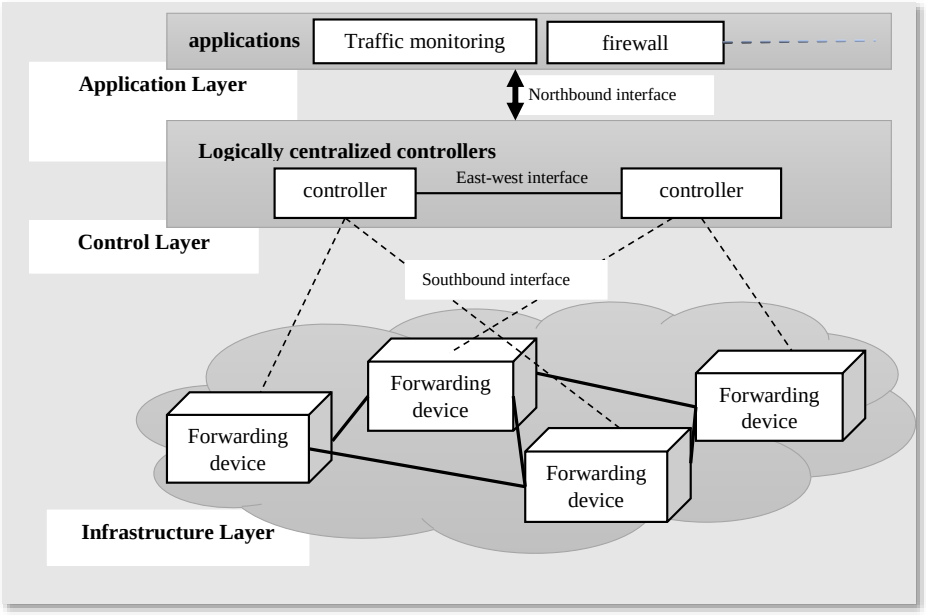


Figure 2. Architecture of SDN

Figure 2 depicts the simplified design of a software defined network (SDN), which is a three-layer architecture –

- Application Layer
- Control Layer
- Infrastructure Layer

Simple forwarding networking nodes are connected in the infrastructure layer. The data-infrastructure plane's layer oversees forwarding packets and monitoring local data. The control layer, also known as the network operating system (NOS), contains the network's intelligence and provides centralised control over the data plane [16]. It comprises of a logically centralised controller that controls the forwarding elements in concert. The application layer contains the application (for example, a traffic monitoring or firewall programme). The network flows are controlled by the logic of these apps. In SDN, three interfaces are used –

- Southbound interface
- Northbound interface
- East-west interface

The southbound interface allows controllers and forwarding devices to communicate more easily. The OpenFlow API is the most widely used southbound interface in SDN.

The control plane and application layer communicate via a northbound interface. The controllers interact using an East-West interface.

4 Difference Between SDN Architecture And Current Network Architecture

Current networking architecture	SDN architecture
Protocol that is fully distributed	APIs that are logically controlled by a single piece of software
Automation is conceivable, but it is time-consuming	All devices have a same interface (API).
Proprietary interface	APIs for data access and manipulation
Individual configuration of devices	Central control
On devices, the flowchart is closed.	Formats and actions on tables that are explicitly defined.

5 Openflow Based Software Defined Network Architecture

OpenFlow is a flow-based protocol that allows the SDN concept to be implemented in both hardware and software [4]. The OpenFlow-based network design is shown in Figure 3. OpenFlow networks typically consist of three key elements: (a) OpenFlow switches, (b) OpenFlow Controller, and (c) OpenFlow protocol.

A. OpenFlow Switch

The flow table is used by switches to manage incoming packets. Flow entries are kept in descending order of priority in the flow table. Each flow entry has (a) a header field to match against arriving packets, (b) an action (a set of zero or more actions) to perform on the packet when the header field matches, and (c) a counter to record packet statistics [15]. The header field of each flow item in the flow table is compared to the header file of the arriving packet, starting with the first flow entry in the flow table. The packet is transmitted to the controller using the Packet-in message if no match is detected. The flow table is not examined for local traffic (traffic to and from the encrypted channel). The most extensively used software-based OpenFlow switch is OpenvSwitch [5.]

B. Controller

A central controller or numerous physically distributed but conceptually centralised controllers make up the control plane. The communication protocol between these logically centralised controllers is defined by the East-West bound interface. It provides an abstract representation of the application layer. The first OpenFlow controller, NOX [6], was written in C++ and Python. OpenFlow controllers include POX [7], sometimes known as NOX's younger sibling, Ryu [8], floodlight [9], and OpenDaylight [10].

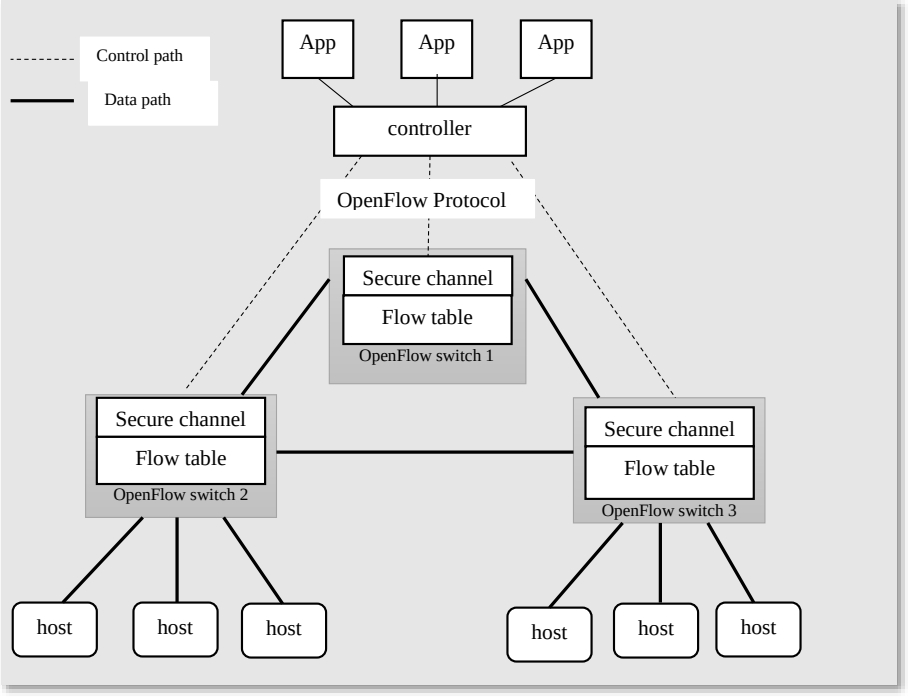


Figure 3. OpenFlow based SDN architecture

C. OpenFlow Protocol

Between the controller and the switches, the OpenFlow protocol establishes a secure channel (TLS/TCP). The controller maintains, configures, and communicates with the switches over this secure channel. (1) Controller-to-switch messages, which are sent by the controller to configure, manage, or obtain the state of switches; (2) Asynchronous messages, which are sent to the controller by switches when the switch state changes, an error occurs, or there is no flow entry for an incoming packet; (3) Symmetric messages, which are sent voluntarily by either the controller or the switch. [19]

6 Methodology

6.1. Simulation Tool and Experiment Setup

In this section, we'll go through a quick overview of a simulation tool that's needed to design a layer 2 OpenFlow-enabled SDN network as well as a traditional network. Researchers have a lot of freedom when it comes to experimenting with physical testbeds. Many components (e.g., OpenFlow switches, controller machine, physical infrastructure) are necessary to establish a physical testbed for OpenFlow application research. Mininet is a free and open-source network emulator that allows you to build and run realistic software-based networks on a single computer.[20]

6.1.1. Mininet

Mininet is a free and open-source network emulator that allows you to build and run realistic software-based networks on a single computer. Mininet runs numerous switches and hosts on a single operating system (LINUX) kernel using lightweight virtualization. For deployment, testing, and performance analysis, the code we write and test on Mininet may easily be moved to the real network with minimum changes.

Mininet virtual machine installation in a virtualization programme is the simplest way to install Mininet on a non-Linux operating system. We tested Mininet vm version 2.2.2 on Ubuntu 14.04 LTS, which was running in Oracle vm VirtualBox on a Windows operating system. [12] contains installation instructions and setup notes. Putty and the Xming server are used to establish an X11 forwarding enabled ssh connection with mininet vm. The Xming server runs on the Windows operating system, allowing it to execute X11 applications (e.g., gedit, xterm, Wireshark). To connect to Mininet vm via ssh, you must first obtain the IP address of Mininet vm using the command-line interface.[14]

```
$ sudo dhclient eth1
```

```
$ sudo ifconfig eth1
```

7 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:

As depicted in Figure 4, we build an OpenFlow network with 10 OpenvSwitch (s1 to s10), 28 virtual hosts (h1 to h28), and an OpenFlow controller pox. Every host has its own IP address. Virtual ethernet cable with 1000Mbps capacity connects hosts and switches. The flow table of each OpenvSwitch in an OpenFlow network is initially empty. All ten switches are controlled by the remote OpenFlow controller POX (carp branch) [13]. It is necessary to construct a component class for the controller that enables the hosts to communicate with one another through the use of learning switch logic.

If a switch gets a packet and the switch has no flow rule for the packet, the packet is delivered to the controller, according to learning switch logic. The MAC address of the sender and the switch port where the packet was received are both saved by the Controller. In order to discover the recipient's MAC address and port, the controller will flood the packet [17]. The controller will then insert the flow rule for the sender and receiver into the switch's flow table. Learning sw.py, a component for layer 2 learning switches, is saved in the /pox/ext folder, allowing Open Switches to operate 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

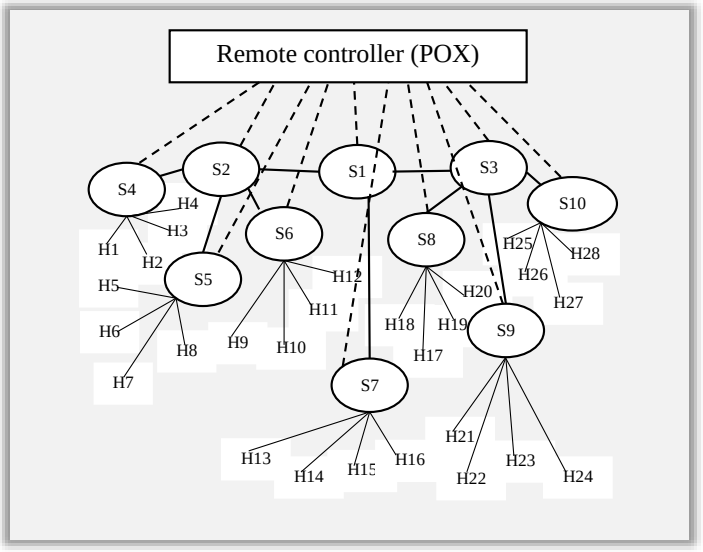


Figure 4. Proposed OpenFlow based SDN model

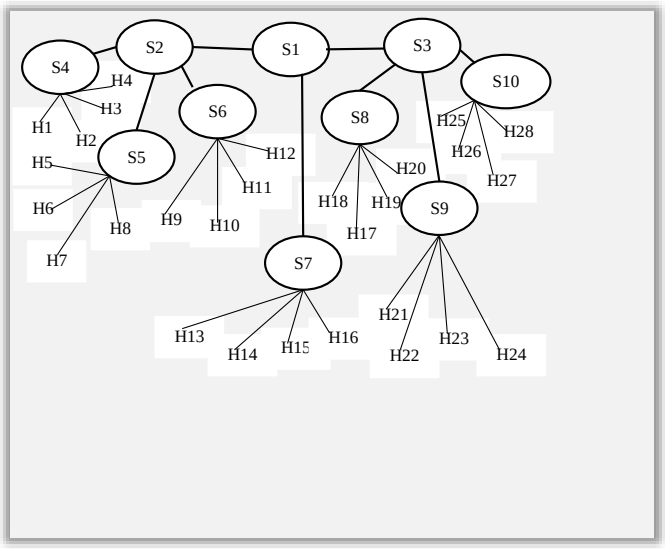


Figure 5. Traditional layer 2 network model

B. Traditional Network:

Figure 5 shows a typical network model in which the layer network is created using Linux Bridge (a layer 2 virtual device). Bridge-utils must be installed in Mininet in order to use Linux Bridge. The Linux Bridge is made up of a collection of network ports, a control plane, a forwarding plane, and a MAC learning database [18]. The following command is used to run the traditional network or the OpenFlow network:

$ sudo python

8 Result and Analysis

The major goal of this study is to examine and contrast the performance of an OpenFlow network with a Pox controller to a traditional network. To do so, we run a network connectivity test and analyse the network's throughput.

PING is a network connectivity test and latency measurement tool. The Ipref utility is used to generate traffic and measure throughput over both a TCP and a UDP connection. Between source node h1 and destination node h28, a ping test is conducted. As demonstrated in figs. 6 and 7, the average rtt (round trip time) for the first ping test in an OpenFlow network is 161ms and 6.03ms in a traditional network. In an OpenFlow network, latency for the first ping test is important since the flow table is empty and the switch delivers the packet-in message to the controller.

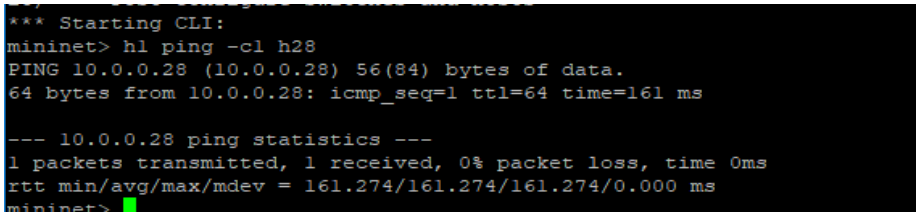


Figure 6. First ping test in OpenFlow network

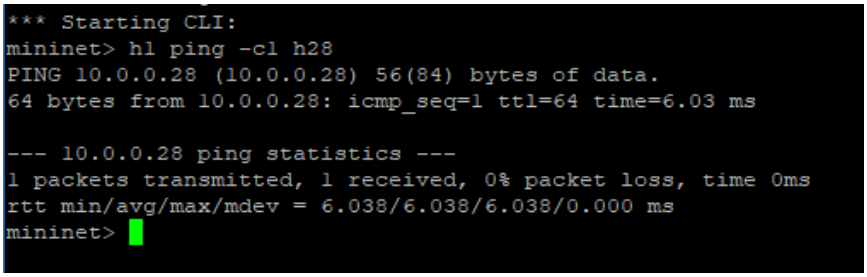


Figure 7. First ping test in traditional network

To test network connectivity across nodes, the ping all command is used. To all other Open Switches in ping all, each Open Switch sends ICMP (internet control message protocol) echo request messages and waits for responses. The average latency of the OpenFlow network (when flows are implemented in switches) is equivalent to or better than the old network, as demonstrated in figures 8 and 9.

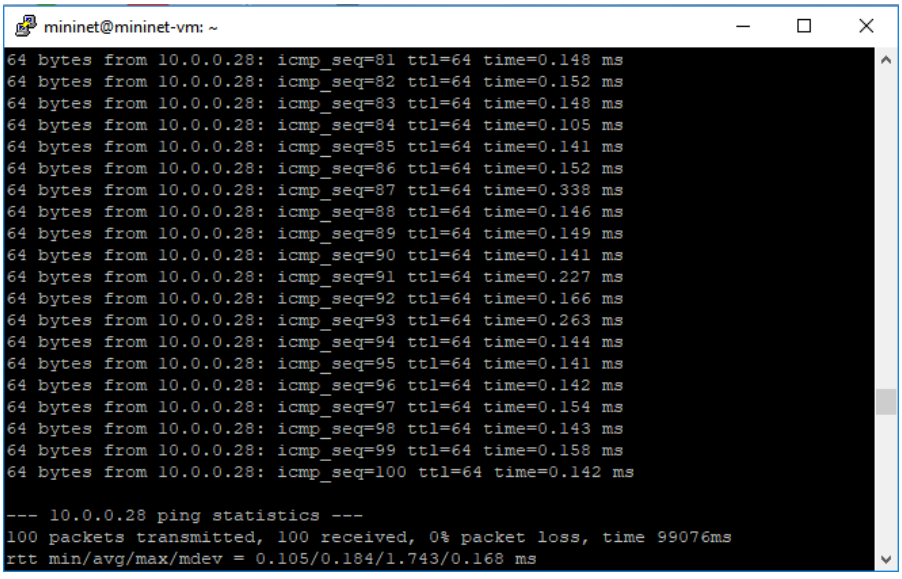


Figure 8. Ping test in OpenFlow network

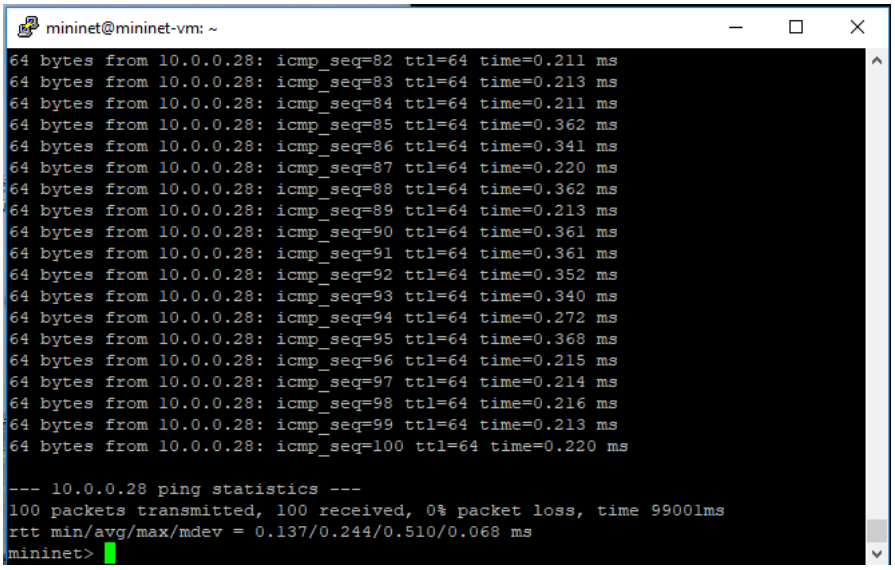
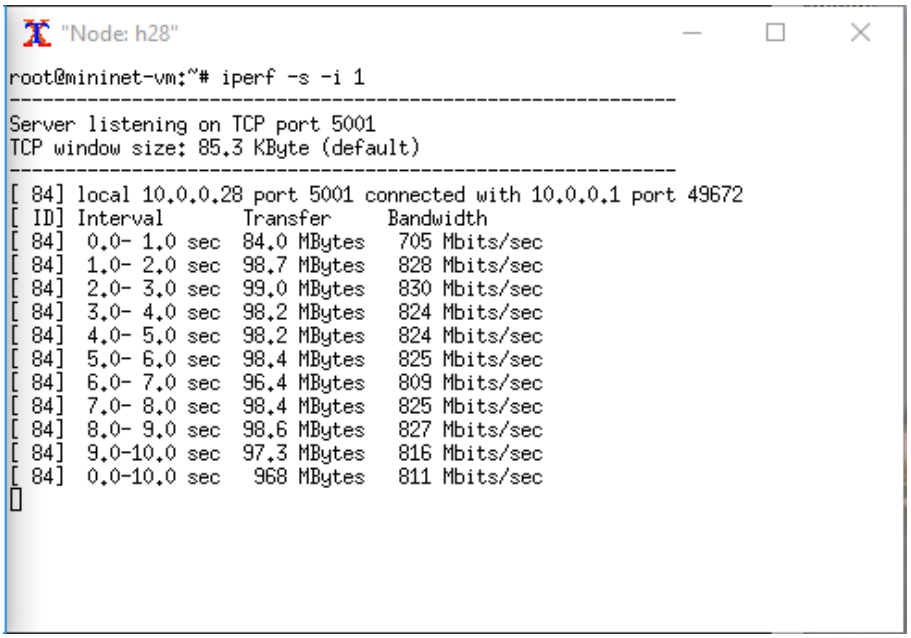


Figure 9. Ping test in traditional network

Table 1. Round trip time comparison between OpenFlow and traditional

Network	minimum	Average	maximum
Traditional network	.137ms	.244ms	.510ms
OpenFlow Network	.105ms	.184ms	1.743ms

The Iperf utility is used to analyse bandwidth consumption in networks between source node h1 and destination node h28. TCP server is started at destination host 28 and TCP client is started at host h1. The snapshot of the result shows the commands that were used at the source and destination nodes. Data is transmitted via a TCP connection for 10 seconds to analyse bandwidth utilisation.



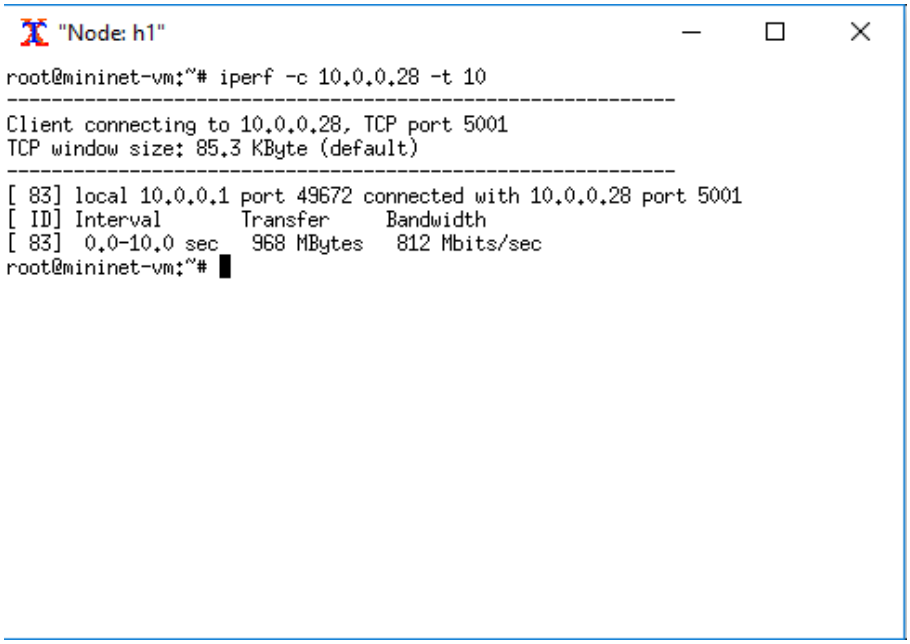


Figure 10. Throughput over TCP connection in OpenFlow network

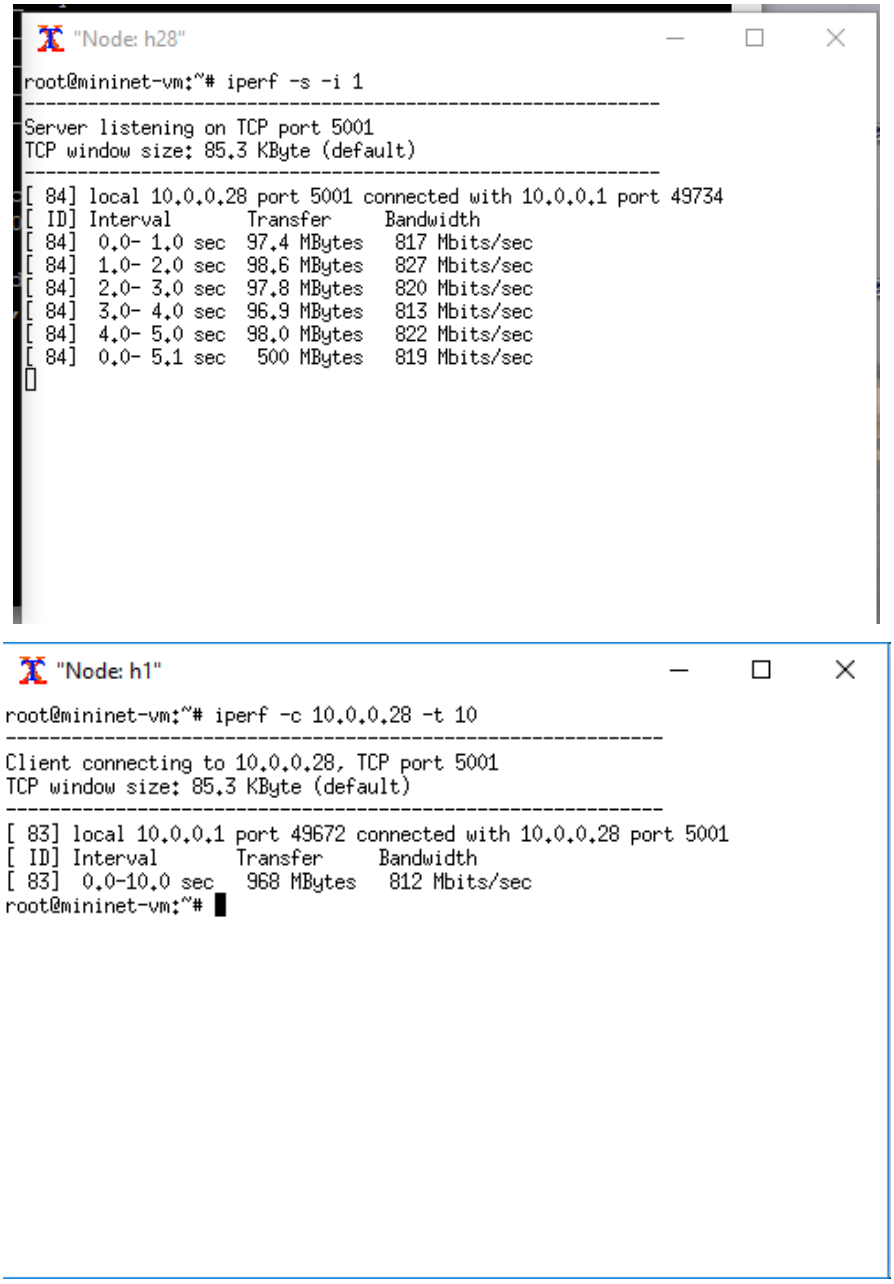


Figure 11. Throughput over TCP connection in Traditional network

Table 2 shows a comparison of the bandwidth of an OpenFlow network and a typical network via a TCP connection. Figures 10 and 11 demonstrate the commands that are used at the source and destination nodes, respectively.

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

Network	Data transferred	Bandwidth
Traditional network	945MByte	792Mbps
OpenFlow Network	968MByte	808Mbps

In a UDP connection, jitter, packet loss, and out-of-order packet delivery are a few factors that might reduce the throughput of the network. UDP client is initiated at source node h1 and UDP server is started at destination node h28 for UDP testing. Over a UDP connection, data is transferred for 10 seconds. Figures 12 and 13 illustrate the results of UDP tests, while table 3 shows a comparison of networks based on server reports.

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	0

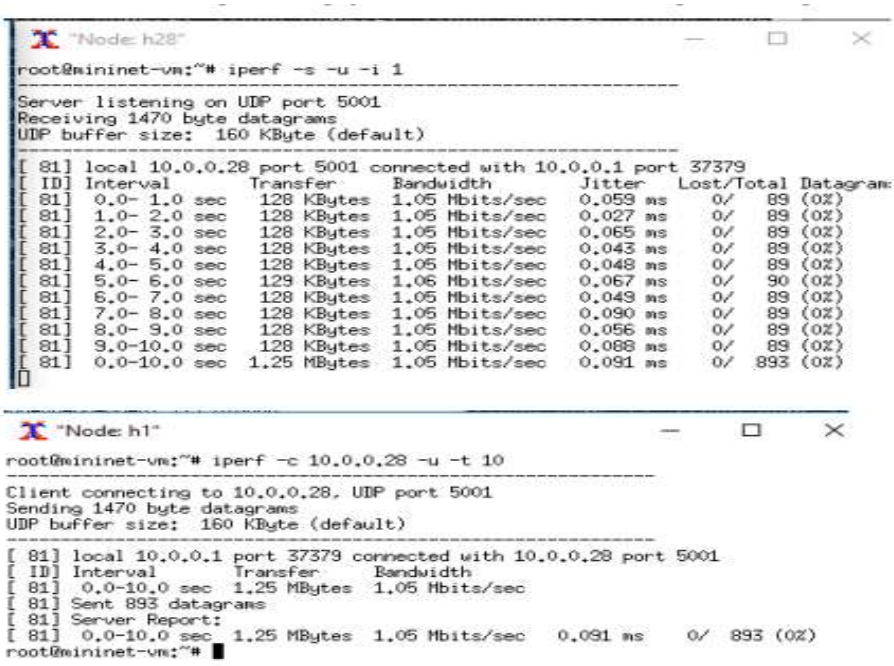


Figure 12. Throughput over UDP connection in Traditional network

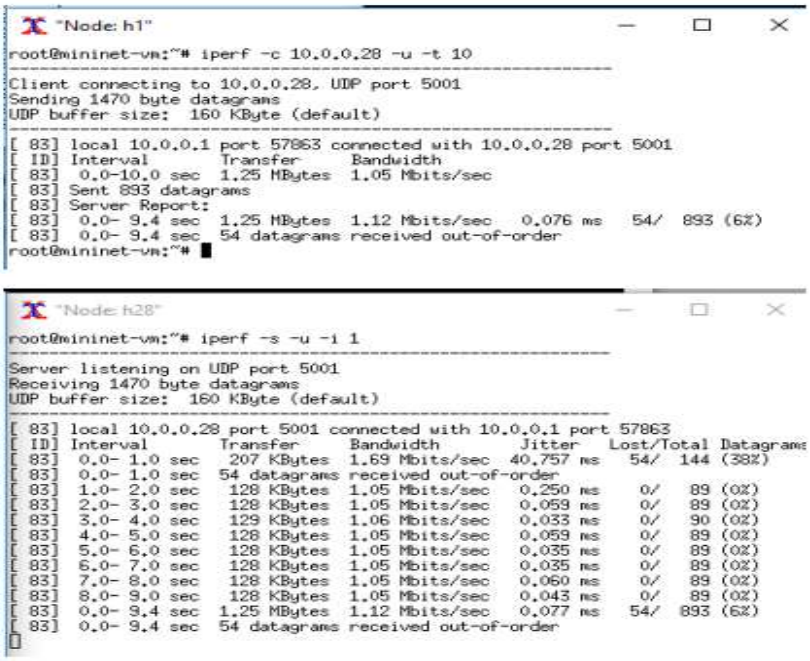


Figure 13. Throughput over UDP connection in OpenFlow network

In order to assess the network's performance, ping and iperf were utilised. The ICMP () echo request message is sent by the ping command to the specified destination IP address. If the target can be reached, an ICMP echo reply message is sent. Ping testing gives us the rtt. Every second, the ping command transmits one ICMP request packet. In both networking environments, Sending a single echo request message from source node h1 to destination node h28 kicks off our ping test. The first packet in the switch must enter the flow because we used the reactive technique. As a result, the rtt time in the OpenFlow environment is substantially longer (161.274ms) than the rtt time in the traditional networking context (6.038ms). By continuously transmitting 100 ICMP packets in both networking environments, we were able to determine the network's average latency. As shown in comparison table 1, the latency of the OpenFlow network is comparable to that of traditional networking.

The iperf utility was used to determine the maximum bandwidth for a TCP connection, as well as bandwidth, jitter, and packet loss for a UDP connection. Iperf provides a UDP and TCP data stream to measure the network's throughput. It is necessary to operate h1 in TCP client mode and h28 in server mode to monitor bandwidth consumption in both networks between source node h1 and destination node h28. For ten seconds, a TCP data stream is transferred from the client to the server. Table 2 compares the bandwidth of an OpenFlow and a traditional network for a TCP connection. We conducted multiple tests and found that OpenFlow's throughput is comparable to that of a traditional network. We conducted all of our analyses using virtualization software, thus the network's performance is ultimately determined by the CPU load, which can fluctuate.

Similarly, a UDP data stream is delivered from the source to the destination for 10 seconds to assess the performance of the connection. Table 3 shows a comparison of the performance of UDP connections. Packet loss and jitter are two elements that affect UDP (a connectionless protocol) network performance. The bandwidth set in the TCLink is not used by UDP. The bandwidth for a UDP connection is set to 1Mbps by default. The -b option can also be used to set the bandwidth. When compared to a conventional network, data loss in an OpenFlow network for UDP connections is higher.

9 Conclusion and Future Scope

Control and data planes are merged with networking devices that are challenging to maintain and configure in a standard networking strategy. Software-defined networking removes the networking equipment's governing mechanism and turns it into a simple forwarding node. The logically centralised controller is in charge of these nodes. Using the mininet network emulator, this article compares the performance of a traditional network versus an OpenFlow-enabled software-defined network. We ran a network connectivity test using the ping command to check for connectivity as well as evaluate and compare network latency. Based on the findings, it can be inferred that the round-trip duration of the OpenFlow network for the initial echo request ICMP message is significantly longer than the traditional network. OpenFlow, on the other hand, performs similarly to a traditional network when flow rules are put in the switches. In TCP and UDP connections, the OpenFlow network has similar or better throughput than the standard network. By deploying an OpenFlow network, the networking system will become programmable, manageable, scalable, and fast.

References

[1] S. Dong, K. Abbas and R. Jain, "A Survey on Distributed Denial of Service (DDoS) Attacks in SDN and Cloud Computing Environments," in IEEE Access, vol. 7, pp. 80813-80828, 2019.

[2] Guerber, Christophe, Mickaël Royer, and Nicolas Larrieu. "Machine Learning and Software Defined Network to secure communications in a swarm of drones." *Journal of Information Security and Applications* 61 (2021): 102940.

[3] Hamdan, M., Hassan, E., Abdelaziz, A., Elhigazi, A., Mohammed, B., Khan, S., ... & Marsono, M. N. (2021). A comprehensive survey of load balancing techniques in software-defined network. *Journal of Network and Computer Applications*, *174*, 102856.

[4] Isyaku, B., Mohd Zahid, M. S., Bte Kamat, M., Abu Bakar, K., & Ghaleb, F. A. (2020). Software defined networking flow table management of openflow switches performance and security challenges: A survey. *Future Internet*, *12*(9), 147.

[5] Du, Jilong, Peng Han, and Yuchen Xi. "SDN applications and related algorithms in network architecture." In *International Conference on Electronic Information Engineering and Computer Technology (EIECT 2021)*, vol. 12087, pp. 100-112. SPIE, 2021.

[6] Torres, Eliseu, et al. "A SDN/OpenFlow framework for dynamic resource allocation based on bandwidth allocation model." *IEEE Latin America Transactions* 18.05 (2020): 853-860.

[7] Prabakaran, D., S. Mohammed Nizar, and K. Suresh Kumar. "Software-defined network (SDN) architecture and security considerations for 5G communications." *Design methodologies and tools for 5G network development and application*. IGI Global, 2021. 28-43.

[8] Tadros, Catherine Nayer, Mohamed RM Rizk, and Bassem Mahmoud Mokhtar. "Software defined network-based management for enhanced 5G network services." *IEEE Access* 8 (2020): 53997-54008.

[9] Tadros, Catherine Nayer, Mohamed RM Rizk, and Bassem Mahmoud Mokhtar. "Software defined network-based management for enhanced 5G network services." *IEEE Access* 8 (2020): 53997-54008.

[10] A. Shirvar and B. Goswami, "Performance Comparison of Software-Defined Network Controllers," 2021 International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT), 2021, pp. 1-13.

[11] Gomez-Rodriguez, Jose Ricardo, et al. "A survey of software-defined networks-on-chip: Motivations, challenges and opportunities." *micromachines* 12.2 (2021): 183.

[12] Rasool, Zaid Ibrahim, Ridhab Sami Abd Ali, and Musaddak Maher Abdulzahra. "Network management in software-defined network: A survey." *IOP Conference Series: Materials Science and Engineering*. Vol. 1094. No. 1. IOP Publishing, 2021.

[13] Isolani, Pedro Heleno, Juliano Araujo Wickboldt, Cristiano Bonato Both, Juergen Rochol, and Lisandro Zambenedetti Granville. "Interactive monitoring, visualization, and configuration of OpenFlow-based SDN." In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 207-215. IEEE, 2015.

[14] Bhardwaj, Shanu; Panda, S. N.; Muskaan, ; Datta, Priyanka (2020). [IEEE 2020 International Conference on Emerging Smart Computing and Informatics (ESCI) - Pune, India (2020.3.12-2020.3.14)] 2020 International Conference on Emerging Smart Computing and Informatics (ESCI) - Comparison and Performance Evaluation of Software-Defined Networking Controllers. , (), 276–281.

[15] Haji, Saad H., et al. "Comparison of software defined networking with traditional networking." *Asian Journal of Research in Computer Science* (2021): 1-18.

[16] Noman, H. M., & Jasim, M. N. (2020, July). POX controller and open flow performance evaluation in software defined networks (SDN) using mininet emulator. In *IOP conference series: materials science and engineering* (Vol. 881, No. 1, p. 012102). IOP Publishing.

[17] Islam, M., Islam, N. and Refat, M., 2020. Node to node performance evaluation through RYU SDN controller. *Wireless Personal Communications*, *112*(1), pp.555-570.

[18] Ali, S., Haque, M.R., Nisar, K., Kannan, R., Khan, T.A. and Ali, B., 2022. Performance analysis for SDN POX and open daylight controller using network emulator Mininet under DDoS attack. *Computational Intelligence in Software Modeling*, *13*, p.177.

[19] Bhardwaj, S. and Panda, S.N., 2022. Performance Evaluation Using RYU SDN Controller in Software-Defined Networking Environment. *Wireless Personal Communications*, *122*(1), pp.701-723.

[20] Askar, Shavan, and Faris Keti. "Performance Evaluation of Different SDN Controllers: A Review." (2021): 67-80.