Implementation of Network Traffic Monitoring using Software Defined Networking Ryu Controller

OMRAN M. A. ALSSAHELI, Z. ZAINAL ABIDIN, N. A. ZAKARIA, Z. ABAL ABAS

Fakulti Teknologi Maklumat dan Komunikasi
Universiti Teknikal Malaysia Melaka 76100 Melaka, MALAYSIA

Abstract: - Network traffic monitoring is vital for enhancing the overall network performance and for optimizing the traffic flows. However, an emerging growth of use in cloud services, internet-of-things, block-chain and data analytics, demand the hardware-based-network-controller to provide more features for expanding network architecture. Therefore, Software Defined Networking (SDN) offers a new solution in terms of scalability, usability and programmable software-based-network-controller for the legacy network infrastructure. In fact, SDN provides a dynamic platform for the network traffic monitoring using international standard. In this study, SDN setup and installation method uses a Mininet emulator containing a controller Ryu with switching hub component, OpenFlow switches, and nodes. The number of nodes is adding until reaches to 16 nodes and evaluated through different network scenarios (single, linear and tree topology). Findings show that the single topology gives a winning criterion compared to other topologies. SDN implementation is measured with performance parameters such as Throughput, Jitter, Bandwidth and Round-Trip Time between scenarios using the Ryu controller. Future research explores on the performance of SDN in larger network and investigates the efficiency and effectiveness of SDN implementation in mesh topology.

Key-Words: - Network management, network performance, network traffic monitoring, software-defined networking (SDN), SDN traffic monitoring, Mininet and Ryu controller.

Received: January 9, 2021. Revised: April 16, 2021. Accepted: April 30, 2021. Published: May 16, 2021.

1 Introduction

Internet offers a wide variety of resources and software platforms for a rigorously growing numbers of users in the cyber world. It is essential to monitor the internet growth since it is used in daily communication, education, industrial automation, health management and online business. This enormous expansion of internet brings many challenges to the network administrator such as cyber-attack, network management issues, network configuration, network performance, safety, and monitoring the overall network's health [1].

The networking infrastructure is widely implemented in organizations with different types of network services and mechanisms. With a huge number of applications to monitor how the traffic behavior and test the traffic network requirements requested, a high usage in network controller arise. Moreover, the scalability problem in network performance is affected by different scenarios due to bottleneck problem.

Network monitoring is a method of capturing and carefully inspecting network traffic to determine activities in the network. Several methods to perform the network traffic are network analysis, protocol analysis, packet sniffing and packet analysis. The monitoring of network traffic is a collection of techniques that are used successively to know the nature of traffic per packet or per rate of flow. Moreover, the network is monitored based on traffic statistics and the network analysis depends on the availability of network traffic measurement. The network topology is analyzed using network controller to monitor how the traffic behavior in a network. Using a virtual machine, a Linux Operating System is installed to run all the program. A network topology is setup and experiment test on the network traffic based on data packet is implemented using the simulation software.

The scope of this research is to implement a network monitoring using SDN Ryu controller. Therefore, the network controller used in this study is SDN-Ryu Controller. Section 2 describes about the SDN and

scenarios for further testing and analysis phase. Section 3 explains about the experiment environment in implementing the SDN-Ryu Controller. Section 4 describes about the results and section 5 concludes the study about load balancing using SDN-Ryu Controller in improving the network performance.

2 Related Works

A SDN is a software-based network controller producing a flexible, scalable, cost effective and adaptive features that is ideal for high-bandwidth and complex application. Through incorporation of a few of low-level features of the network application instead of hardware implementation helps the network administrators to be more manageable to control the complex networks. Moreover, SDN architecture integrates the network control and forwarding function that allow a dynamic and programmable configuration in a cloud-based network monitoring controlled by a remote-control plane.

However, due to the increasing demands for internet bandwidth from the new devices and users, the internet is still not sufficient to meet the high network requirements, such as flow requests and network statistics monitoring. For instance, one of the network controllers, NOX is appropriate for small to medium-sized networks serves 30K stream demand in every second with a reaction time less than 10ms. In fact, this amount is not appropriate for some network settings such as data center [5]. The packet request handled by a network controller is compounded by an increase in the number of network devices.

The network expand layers in SDN consists of application and control layers, which responsible for unified control action to solve the scalability issues, as it decouples the control plane. Thus, the decoupling condition creates the bottleneck function in the control plane and demand the use of micro controllers for better traffic management. Furthermore, the controller struggles to manage more events and flow requests, due to its limited computing assets such as CPU and memory, which make the network controllers as a bottleneck point.

To improve the network performance, SDN offers a simulation platform for a better performance compared to the traditional network management [2]. Layers in SDN consists of application, control and data planes [3]. The upper layer is the application plane, the middle layer is a control layer, which is known as control plane and the lowest layer is the data plane. Figure 1 illustrates the software-defined network architecture.

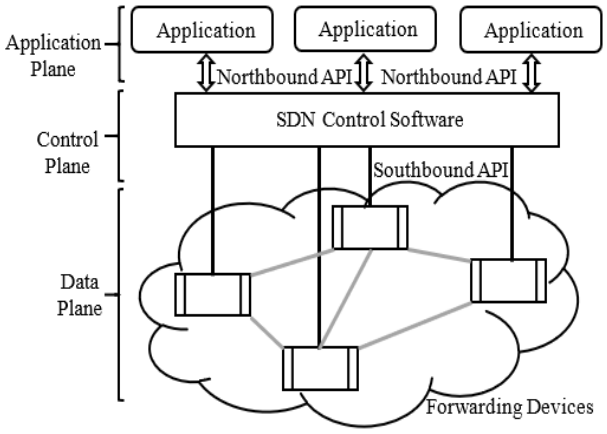

Fig 1. SDN Architecture

A top layer is an application layer, also known as management plane, this layer is responsible for tasks related to management and data transmission (for instance, data traffic monitoring, mobility management, routing, security and load balancing) and for data traffic flow efficiency. The control plane is separated from the core devices of the network and responsible for unified the control action. A single SDN controller is responsible for centrally managing all data flows of an underlying network infrastructure. [4]. In fact, the network control APIs provides open device management, global network view and forwarding space for user-friendly network programming [5]. Data plane consists of ethernet switches, packet switches and routers that tis responsible for transmitting data packets regardless of network architecture.

SDN control software sits in between the control plane and data plane, which solving the scalability issues, as it decoupling control plane from the data plane and requires remote control mechanism that is referring to the network controller management of network devices. Thus, this separation performs in significant overhead signaling due to the network architecture scenario and applications executing on the network controller, which creates the control plane shows a bottleneck function with respect to the system's scalability [6].

To increase the scalability and performance of the control plane in the SDN architecture, Dynamically Partial Reconfigurable Processor System is used to micro controllers to allows for attempting performance scale-out for the virtual device or optimization of the physical processing location in the network system [7]. Thus, improvements in the

use efficiency of the network bandwidths and device resources, as an issue on the data center network, reduce the event resulting from network controllers routing decisions to determine the shortest path.

3 Proposed Method

The proposed method for performing the network traffic control uses the Mininet software as a simulator to create the network scenarios. The network scenarios are represented in topology. In the SDN context, OpenFlow protocol is selected as the interaction standard between controller and network devices. OpenFlow helps network controllers to establish network packet paths across a switch network.

The performance and round-trip time (rtt) are used as the parameter to measure the network performance. The output of the throughput and round-trip time (RTT) is recorded and analyzed using multiple criteria. RTT is the period in milliseconds (ms) from a starting point to a destination and back to the starting point for a network request. Throughput refers to the number of units that can be handled by a system within a specified time period. This parameter was used for analyzing the performance in a network [8]. If the information is not successfully transmitted between hosts in the specified network topologies, then it affect the RTT value and produces a direct effect to the network performance.

There are 3 scenarios implemented in this study, which are single topology, linear topology, and tree topology. All of the experiment design is implemented using simulation software, which is Mininet since it is widely used for SDN network controller as a research and development of prototype [9]. This paper is executed using a simulation environment. Every network topology design consists of SDN Controller, switches, and hosts at the different client connected, which is important for the testing phase.

3.1 Experiment Environment

The experiment environment discusses the method of a network configuration to be implemented. Section 3.1.1 to 3.1.3 explain on the topology (single, linear and tree) based on the scenario. Section 3.2 explains on experiment setup for the network controller and 3.3 used to monitor the behaviors in single, linear and tree topology based on the integration of Wireshark software application and the network controller.

The Mininet emulator is chosen since it has tools for modeling the SDN, replicates and interacts with the real-world SDN controllers. Moreover, Mininet is an open-source application, stable and provide a better control in network performance for SDN. However, the NS3 mimics the functions of SDN switch, demand another application to be combined with NS3 to adopt SDN functions and more tools are needed for SDN implementation [10].

3.1.1 Environment Setup for Single Topology

Figure 2 shows the flow of process created using a single topology. This topology contains sixteen hosts, one network controller and one switch. Two of total hosts are changed to the HTTP server pools connect to the single switch. It creates with an OpenFlow controller using port 6633. The switch attaches to a control plane available, and network controllers are enabled remotely in the Mininet console using their respective commands. Remotely enabled Ryu controller is ready to connect to the IP address of 127.0.0.1 with the underlying network switches. A single line command creates a single topology and connect to the Ryu controller. To install the Mininet, get the source code at git clone git://github.com/,ininet/mininet.

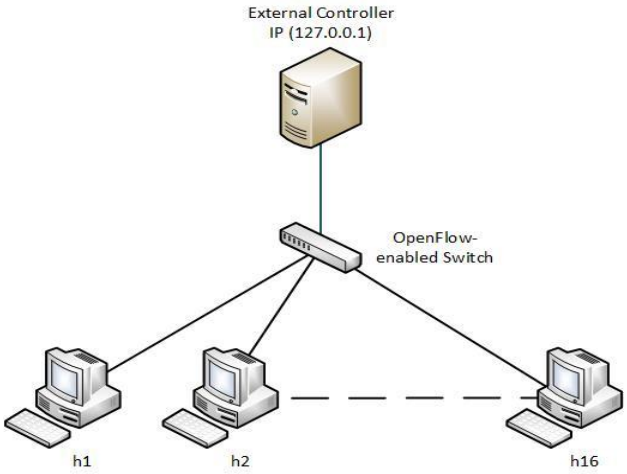

Fig 2. Scenario Single Topology

3.1.2 Environment Setup for Linear Topology

The second design is a linear topology, which consists of a network consisting of 3 hosts and each host connects with its particular switch. The switches are connected with each other linearly as shown in Figure 3.

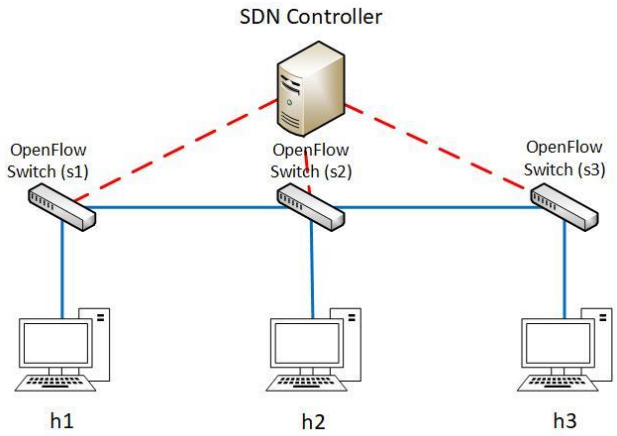

Fig 3. Scenario Linear Topology

The OpenFlow-enabled switches are enabled and connected with a remote network controller. The network controllers are enabled remotely in the Mininet console using their respective commands. Remotely enabled Ryu controller is ready to connect to the IP address of 127.0.0.1 with the underlying network switches. Linear topology has an OpenFlow network controller connected and using port 6633.

3.1.3 Environment Setup for Tree Topology

The third design shows all the OpenFlow-enabled switches and hosts are connected with each other in a hierarchical fashion.

The tree topology uses customs topology. For the custom topology, it needs to create using python coding in order to activate Mininet using a single command. Figure 4 depicts the tree topology using a connected four switches plus a host and four hosts connected as servers. All the OpenFlow-enabled switches in turn gets connected with a remote controller. The remote network controllers are enabled remotely in the Mininet console using their respective commands. Remotely enabled Ryu controller is ready to connect to the IP address of 127.0.0.1 with the underlying network switches.

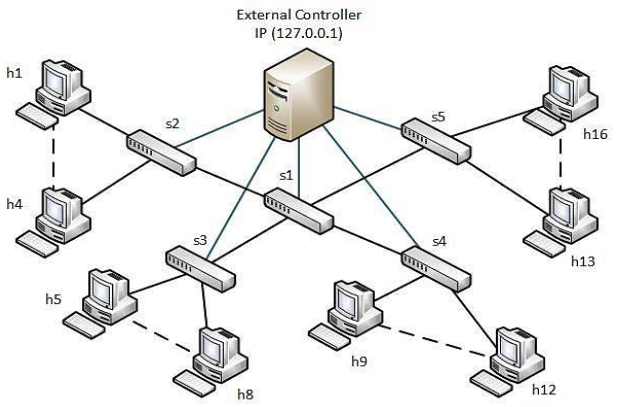

Fig 4. Scenario Tree Topology

3.2 Experiment Setup for Network Controller

OpenFlow network controller is configured and connected to the switch. It has several types of controllers that could be implemented with Mininet software. Table 1 shows that the Ryu controller using a Load Balancer protocol. The Network uses IP address 10.0.1.1, and the server uses IP address 10.0.0.1 for server one and 10.0.0.2 for server two.

A comparison of performance is made by comparing the total result obtained between the end hosts. A round trip between hosts can be obtained using the 'ping' (Echo request and reply message) command to execute ICMP request message for a connectivity test. An overall network performance from the available throughput is achieved. A throughput is defined as the quantity of data delivered over a given time period. All the network topologies created have 100 Mbps bandwidth physical connection parameters with 1 ms propagation delay.

Table 1. Configuration of Network Controller

Parameter	Description
Servers	10.0.0.1, 10.0.0.2
Debug Core	PDX 0.5.0 eel, On CPython
Test Platform	Linux 5.0.0-37, Ubuntu 19.04 x86_64
OpenFlow debug info	Listening on [0.0.0.0:6633]

Network controllers' performance in previously defined network topologies is achieved using the 'ping' command (Echo request and reply message) to perform the ICMP connectivity test. A 'ping' test for the measurement of round-trip time (RTT) between hosts is performed between end hosts h1 and h16 of the specified network topologies.

3.3 Experiment Setup for Network Monitoring

OpenFlow Wireshark is an application software or an open-source tool that use to monitor a network performance. To analyze the network performance, Wireshark software is integrated with Mininet in

understanding the scenario in single topology, liner topology and tree topology. Moreover, behaviors of network topologies are demonstrated through the simulation enabled, IP traffic is captured according to the time stamp packet, which has the packet type, source time and arrival packet at the destination IP and the protocol of the IP traffic.

To capture packet transfer, Mininet needs Wireshark to be executed at the background. "$ sudo Wireshark &" command line was used to executed the Wireshark in the background to capture OpenFlow packet. The condition of "of" is needed to fill in the filter box. Figure 5 shows Wireshark monitoring IP packet in real-time as the network topology is executed. Thus, based on the Wireshark, the network monitoring process is successfully implemented for further improvement. The red box shows that the OpenFlow packet type is captured by the Wireshark, which falls under the TCP packet protocol.

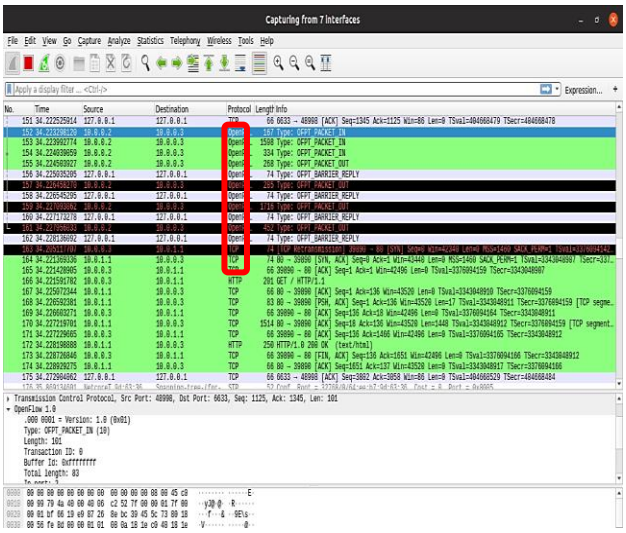

Fig. 5 Monitoring IP Traffic using Wireshark

4 Results and Findings

Section 4 shows an overall result and findings of the SDN performance according to scenarios (single, linear and tree topology). The parameter of network performance is based on throughput, bandwidth utilization and jitter for every scenario.

The result is captured and automatically drawn delay between each packet and the highest amount of data in given time is obtained. Moreover, in the era of industrial revolution, the obtained data at the specific time is widely used and all three created topologies are using Mininet software application.

Network throughput with different types of topologies, show an overall network performance and produces an average result of network throughput between end nodes in different types of topologies. The result of throughput is collected in 60 second of test. the graph for throughput in single topology for each 10 seconds. For each 10 second, the result captures and automatically draw by using gnuplot. The highest throughput obtain is 65.5 Mbps and the lowest is 55.6 Mbps. Figure 6 shows the throughput result for a single topology.

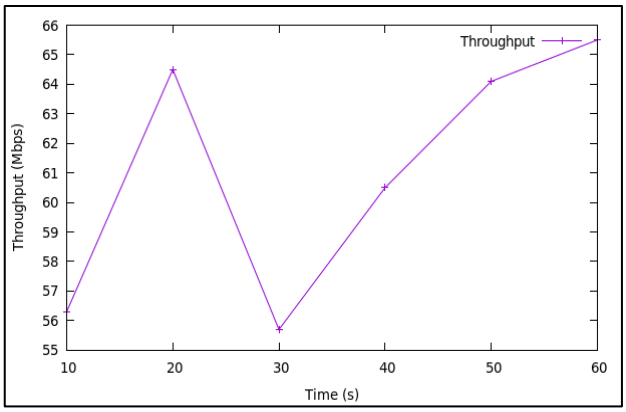

Fig. 6 Throughput – Single Topology

The average throughput result is obtained. The result for average throughput is calculated in between 10 to 60 seconds. For 100Mbps bandwidth link that we created for this topology, the throughput that obtained is around 56 Mbps to 61 Mbps.

On the other hand, throughput for linear topology is implemented as depicted in Figure 7. For each 10 second, the result captures and automatically drawn by the computer using gnuplot. The highest throughput obtain is 65 Mbps and the lowest is 44 Mbps.

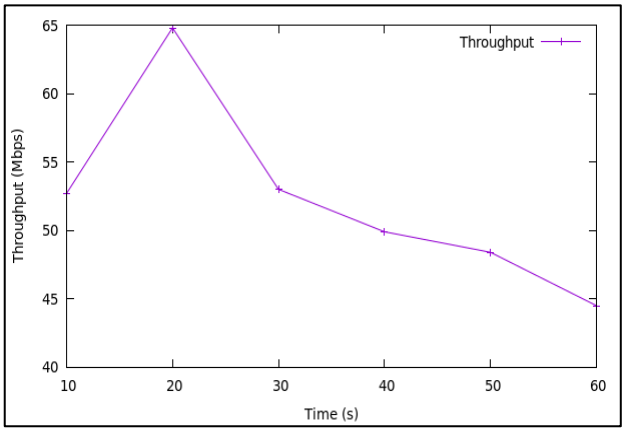

Fig. 7 Throughput – Linear Topology

Figure 8 explains on an average throughput that is measured for a tree topology between 10 to 60 seconds. For 100Mbps bandwidth link has been created and the throughput that obtained is around 52.7 Mbps to 52.2 Mbps. Throughput in the tree topology has been executed for each 10 seconds. For each 10 second, the result captures and automatically draw by using gnuplot. The highest throughput obtain is 70 Mbps and the lowest is 36 Mbps.

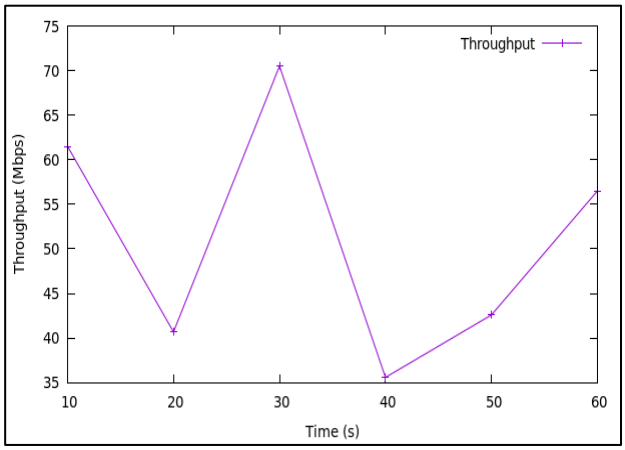

Fig. 8 Throughput – Tree Topology

Jitter is defined as a variance in packets arrivals at the destinations due to network congestion and incorrect queuing. The longer route for data packets to travel, the higher the jitter, which reduce the quality of the data transmission. In fact, this steady data stream produced to be unbalanced.

For a single topology in figure 9, the User Datagram Protocol (UDP) packet type is analyzed for monitoring the jitter. UDP is faster than Transmission Control Protocol (TCP) packet since UDP uses smaller packet header and no speed limit. Compared to TCP, it has speed limits and fixed at eight bytes.

By using "iperf" command and set the protocols to UDP, the jitter is recorded for 60 seconds test with 10 second of interval. The highest jitter for single topology produced 1.78 ms.

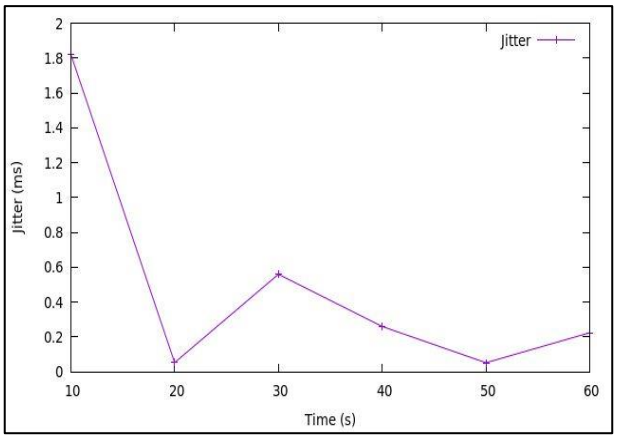

Fig. 9 Jitter – Single Topology

According to the result in figure 10, it shows that the highest jitter value is 9 ms. Moreover, this high value of jitter occurred since every node has a dedicated connection to each OpenFlow switches, which produce data packet delay. The data packet is delayed because update of information at every switch needs to be implemented in switch table and this condition consumes computational processing time. If the updates are in one switch, then less processing time is required. Figure 10 shows the jitter performance in linear topology.

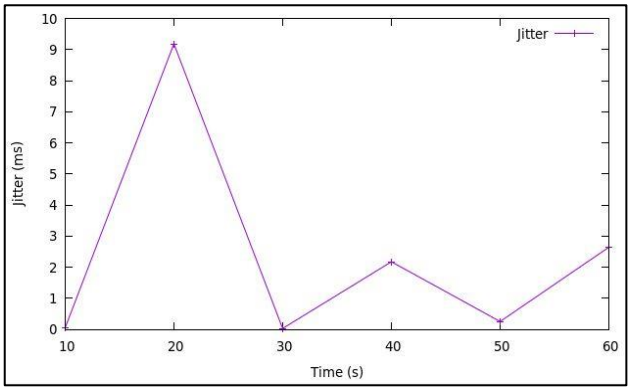

Fig. 10 Jitter – Linear Topology

For single and linear topology, the jitter sparks at the early stage of the data packet transmission and later there is a constant distribution of data packet in the network.

However, in figure 11 illustrates the jitter performance in the tree topology. The jitter value starts to drastically increase as the number of nodes are added to the network. The jitter value is almost 0.8 ms at the time of 50 second nonetheless shows a steady performance at the early distribution of data

packets. Based on the outcome, the tree topology handles a better jitter situation compared to single and linear topologies.

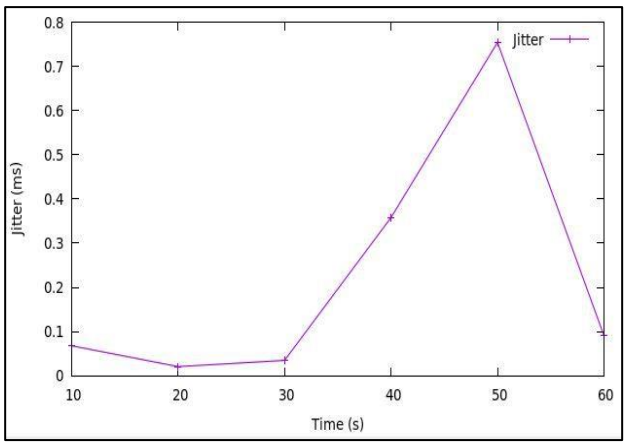

Fig. 11 Jitter – Tree Topology

Bandwidth is measured as how many amounts of data that can be transferred over a one point to another within a network in amount of time. Bandwidth utilization for all three topologies is obtain by using "iperf" command in Mininet. Table 2 produced the highest bandwidth obtain in the result is single topology since the single topology using an Open-Flow Switch to connect with all hosts. Rather than other topologies are using more than one Open-Flow Switch to connect with all hosts.

Table 2. Bandwidth Utilization

Bandwidth	Single Topology	Linear Topology	Tree Topology
Maximum	72.4 Mbps	58.8 Mbps	59.7 Mbps
Minimum	60.1 Mbps	57.0 Mbps	57.9 Mbps

To test the performance of bandwidth utilization using the Ryu controller is through video streaming. Video streaming is a parameter and widely used for measuring the bandwidth. In this study, the VLC media player is used for video streaming. VLC is a free and open-source cross-platform media player. VLC plays many media files and various streaming protocol.

Three topologies (single, linear and tree) created are using the same size of links that is 100 Mbps bandwidth. Then, the media files used in this study is MP4. All topologies are streaming at the same type of media files that is 100Mbps of bandwidth link since a small bandwidth gives less network problems to delivered the data. Figure 12, 13 and 14 shows video packet captured during video streaming for bandwidth performance for single, linear and tree topology. Even though the topology (single, linear and tree) is not similar in architectural design but the bandwidth performance shows almost a similar output graph for a three topology (single, linear and tree).

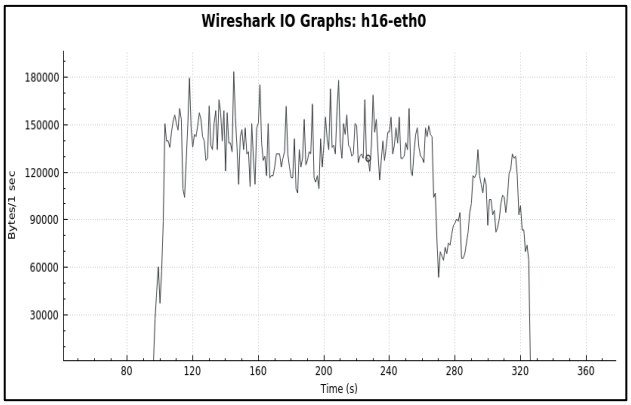

Fig. 12 Bandwidth – Single Topology

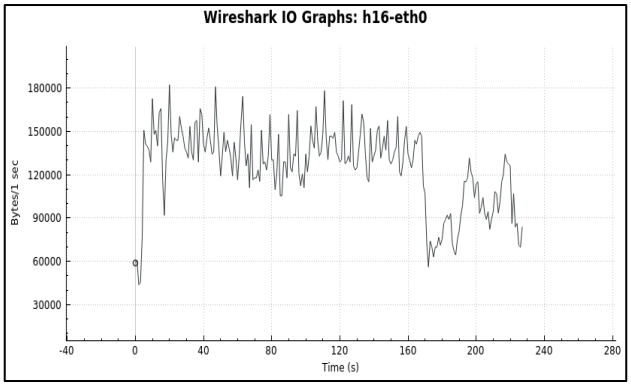

Fig. 13 Bandwidth – Linear Topology

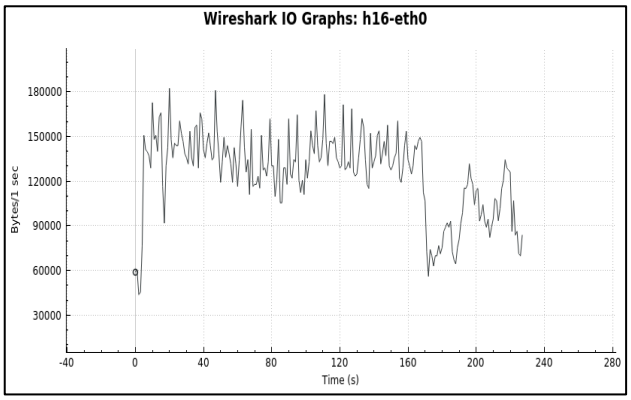

Fig. 14 Bandwidth – Tree Topology

5 Conclusion

In this investigation, the aim was to implement and monitor the network traffic using SDN controller. Furthermore, the SDN performance is evaluated based on parameters such as throughput, jitter and bandwidth utilization. To measure the SDN performance, a simulation-based experiments have been conducted to replicate the real environment of the SDN functions based on scenarios (single, linear and tree).

Results of this study shows that for throughput performance, the single topology provides a high throughput, the linear topology gives low throughput and tree topology shows an unbalance throughput.

Findings of throughput summarized a single topology has a successful data transmission from one node to another node in a given time of duration. On the other hand, linear topology gives a low throughput that indicates an unsuccessful data transmission due to data packet lost. In tree topology, the throughput shows an unbalance condition since to hardware failure factor.

Findings in jitter indicates a decreasing value in single and linear topologies. This means the irregular time delay during sending and receiving the data packet is small that not more than 1% of packet loss. If the jitter performs an imbalance condition in the tree topology, thus this outcome indicates that number of loads really effect the jitter performance.

Findings in bandwidth performs a similar pattern of graph, for single, linear and tree topology in a normal condition. However, bandwidth is not affected by data speed, bottleneck forming and number of nodes in the network, which the latency needs to be reduced for higher speed in data transmission.

It is recommended that further research be undertaken in exploring on the performance of SDN in cloud computing and investigates the efficiency and effectiveness of SDN implementation in larger network.

6 Acknowledgement

Thank you to Fakulti Teknologi Maklumat dan Komunikasi and Universiti Teknikal Malaysia Melaka.

References:

[1] M. Hartung and M. Korner, SOFTmon-Traffic Monitoring for SDN, in International Workshop on Application of Software-Defined Networking in Cloud Computing, *Procedia Computer Science*, Vol. 110, 2017, pp. 516-523.

[2] K., Benzekki, El Fergougui, A, Elbelrhiti A., Elalaoui, Software- defined networking (SDN): a survey. *Security and Communication Networks*, Vol. 9, No.18, 2018, pp. 5803-5833.

[3] I.Z. Bholebawa and U.D. Dalal, Design and Performance Analysis of OpenFlow-Enabled, *Network Topologies Using Mininet. International Journal of Computer and Communication Engineering*, Vol. 5, 2016, pp. 419-427.

[4] I.Z. Bholebawa, and U.D. Dalal, Performance Analysis of SDN/OpenFlow Controllers: POX Versus Floodlight, *Wireless Personal Communication.* Vol. 98, No. 2, 2018, pp. 1679-1699.

[5] T., Feng, J., Bi, and H., Hu, TUNOS: A novel SDN-oriented networking operating system. In 2012 20th IEEE International Conference on Network Protocols (ICNP), 2012, (pp. 1-2).

[6] M. Karakus and A. Duressi, Quality of Service (QoS) in Software Defined Networking (SDN): A survey, *Journal of Network and Computer Applications*, Vol. 80, 2017, pp. 200-218.

[7] M. Kirchhoff, P. Kerling, D. Streitferdt and W. Feng, A Real-Time Capable Dynamic Partial Reconfiguration System for an Application-Specific Soft-Core Processor, *International Journal of Reconfigurable Computing,* Vol. 2019, Article ID 4723838, pp. 1-14, 2019.

[8] I.Z., Bholebawa, R.K. Jha, and U.D. Dalal, Performance Analysis of Proposed OpenFlow-Based Network Architecture Using Mininet, *Wireless Personal Communication*, Vol 83, 2015, pp. 1-18.

[9] Omran M. A. Alssaheli, Z. Zainal Abidin, N.A. Zakaria, Mininet Network Emulator: A Review, *International Journal of Computer Science and Network Security*, Vol. 19, No 9, 2019, pp. 147-155.

[10] I., Jared, Y., Hemin, Z., Chuanji and R., George, Comparing a Scalable SDN Simulation Framework Built on ns-3 and DCE with Existing SDN Simulators and Emulators. ACM Conference, 2016, pp. 153-164. 10.1145/2901378.2901391.