Performance Visualization of Southbound Interface in Software Defined Networking

Fahrizal Djohar[1*], Eueung Mulyana[2], Suciana[3], Andi Muhammad Ilyas[4], Muhammad Natsir Rahman[5], Achmad Prajudin Sardju[6]
[1,4,5,6]Department of Electrical Engineering, Faculty of Engineering, Universitas Khairun, Ternate, Indonesia
[2]School of Electrical Engineering and Informatics, Bandung Institute of Technology, Bandung, Indonesia
[3]Faculty of Infrastructure and Regional Technology, Sumatera Institute of Technology, Lampung, Indonesia
[*]fahrizaldjohar@unkhair.ac.id

Software Defined Networking (SDN) makes Internet network configuration easier by separating the control plane and data plane. The control plane on the controller has information on network devices in the data plane and centrally control these devices. One of the controllers in SDN being developed is the Open Network Operating System (ONOS). ONOS provides interfaces such as Representational State Transfer (REST) Application Programming Interface (API). The ONOS core REST API provides some information from the network connected to it, such as devices, statistics, and the information in JSON file. The primary objective of this study is to develop an interface that simplifies performance monitoring through graphical representation. This involves testing the visualization with various topologies and conducting a comparative analysis of the visualization results across these topologies. The creation of the interface entails presenting statistical data, available in the form of a JSON file from the ONOS controller via the REST API, on the web interface in graphical format. The resulting visualization generates a graph that aligns with the performance characteristics of each topology, reflecting device details, ports, and additional parameters such as the count of sent and received packets, as well as sent and received bytes. The performance visualization outcomes specific to each topology are consistent with the number of connections and are prominently displayed on the web interface. Additionally, this research evaluates network throughput and bandwidth by sending ICMP packet and iperf tests across each topology. Among all the openflow tests performed on various network topologies, it was observed that the tree topology exhibited the lowest network capacity utilization, followed by the leaf-spine topology, and finally the ring topology.

Keywords: SDN, ONOS, Openflow, South Bound Interface, Topology

1. INTRODUCTION

The development of internet networks is now increasing and rapidly reaching various regions of the world, this development resulted in the need for network equipment to also increase. However, it can be said that the existing network equipment has not changed much since several years ago [1]. Configuration of network devices such as switches and routers is still done individually, when there is a need to add devices, adding these devices makes management even more difficult, and each device can have different configurations depending on the vendor. Network devices have an encapsulation layer that abstracts out the data fields. With the control plane (which manages network traffic) and the data plane (provided that the data is in accordance with the provisions of the control plane) that are still on each of these devices, besides making the network less flexible, it also waits for the development of network technology [2]. From some of the shortcomings of the internet network, an idea emerged to try to solve this problem, such as Software Defined Networking (SDN) technology.

In SDN, the control plane is no longer on each network device but is functioned by the Controller layer, and the data plane that remains on each device follows the rules of the control plane for sending data. SDN controllers can be used to directly program the devices connected to it so that the network can be centrally managed [3]. The control plane on the controller has information on network devices in the data plane so that it is able to display it in the form of a logical map. This capability is possible because of the interface that connects the two planes, Application Programming Interface (API). The interface provided can be used to review, control and modify the network. Control of parts of network equipment can also be done from several network operating systems which is one of the virtualization functions. Admins who have access and are in charge of managing the network can initiate network devices, install SDN controllers, applications, or manage the scope of the controller. The SDN architecture can be run in parallel between SDN and non SDN networks so that it is useful in the transition or migration stage [11]. One of the most developed controllers in SDN is the Open Network Operating System (ONOS).

ONOS provides a control plane on the SDN network to manage the devices connected to it such as switches, hosts, and their links. To communicate with application users, ONOS provides interfaces such as the Command Line Interface (CLI), Graphical User Interface (GUI), and Representational State Transfer (REST)

Application Programming Interface (API). ONOS core REST API provides some information from the network connected to it, such as devices, clusters, tenants, statistics, and others. This research is to test the visualization of southbound, Openflow which is tested on several network topologies.

Openflow is a southbound protocol that connects the control plane and data plane, which allows SDN controllers to interact with network devices such as routers and switches [12]. The development of Openflow as a solution for flexibility between devices of different vendors, where researchers can manage different switches and routers in the same way, so that the overall network is easier to program [13]. The aim of this research is to create an interface that facilitates performance monitoring by presenting data in graphical form, testing openflow on various topologies, and comparing the visualization results of each topology.

2. LITERATURE REVIEW

2.1. Related Work

Software Defined Networking (SDN) is a network architecture that is considered promising in today's computer network development. By separating the control plane and data plane, the control of network components can be done centrally, thus reducing configuration time compared to being done individually on each network device. Deploying an application can be done quickly on an SDN network, and can add to the problem if an application still has bugs that can affect the performance of network devices. A suggestion from Kang et al, a profiling application called SPIRIT, which aims to reduce network admin efforts to reveal bugs that may exist in SDN network applications. Experiments were carried out by checking the performance of several well-known SDN controllers such as ONOS and Floodlight, and trying to reveal any bottlenecks in these SDN network applications [27].

SDN not only provides centralized control, but is also open source so there is a possibility that these applications are carrying threats or suspicious applications that can threaten the network. Lee et al, proposed a system that analyzes SDN applications with a machine learning approach such as cluster analysis algorithms. By running a prototype of the system it produces a system that can detect suspicious applications with high accuracy and low errors [28].

There are several controllers developed in the SDN network and it is important to choose which controller to use. Jarschel et al, developed an application to analyze the performance of an SDN controller called OFCProbe. This application aims to measure the characteristics and bottlenecks of SDN controllers such as Nox and Floodlight. The results show that the Floodlight controller performs quite well on load balancing compared to Nox. Floodlight can distribute processing capabilities on each switch quite well even though the topology is less balanced [26].

2.2. Software Defined Networking (SDN)

SDN has several advantages over legacy networks, such as easily applying rules to a network with certain programs without following commands on certain proprietary devices. By creating a controller centrally, configuration of network devices is done without having to each device, but through a controller that connects the network devices [4]. The SDN architecture consists of three main parts, namely the SDN Controller, the Southbound API, and the Northbound API.

SDN controller as a network control center provides information and manages the devices connected to it. Having centralized network information makes network management more uniform and consistent. Southbound API as the interface from the Controller to network devices to transmit information. Controllers can change how routers and switches send data. The Northbound API connects the controller with the application. Applications depend on a controller to provide network state information including its resources. The application communicates with the controller to request the required information. Usually the northbound API used is the RESTful API. Among the advantages of SDN are that it can be programmed directly from the controller, provides network overview, and speeds up the installation or setup of new applications [6]. There are several SDN controllers being developed, one of which is ONOS.

There are several things that make SDN more valuable such as network programmability which allows the network to be controlled by software that is outside the network devices that provide physical connectivity, so that network operators can update the network to support new services. Traditional network control methods reside on each device. By changing to the centralized control provided by the SDN network, bandwidth management, recovery, security, can be optimized by getting a comprehensive view of the network. The application will interact with the network via the API, not the interface of each piece of hardware. SDN also opened an era of openness that allowed multi-vendor interoperability. An open API opens up various application development opportunities. In addition, the software can control hardware from multiple vendors with open interfaces such as OpenFlow. SDN allows users to develop applications that can monitor network conditions intelligently, and automatically adjust the network configuration as needed [25].

2.3. Open Network Operating System (ONOS)

ONOS is an SDN controller that can be used to manage network devices, run software programs and provide communication services to hosts and other network devices [7]. ONOS is an open source SDN controller that provides a northbound for application development and a southbound interface for controlling user interface devices with CLI and GUI, as well as a REST API for applications. ONOS was developed with service providers, network vendors, and network operators [8].

ONOS interface, the northbound interface, allows applications to request services from the network so that application developers can create programs on the network. The northbound interface also provides a global network view for the application by providing information such as hosts, switches, links and some other information. The southbound interface connects a controller with network elements, maintains the network without having to know the specific device, and allows ONOS to control multiple devices and different protocols [8]. ONOS can be used on multiple servers by using the CPU and memory of these servers and can be upgraded without disrupting network traffic. ONOS core applications, kernels, and services are written in Java which are contained in the Karaf OSGi container. OSGi allows installing modules and running them in one Java Virtual Machine (JVM), thus ONOS can run on several underlying platforms [22].

2.4. JavaScript Object Notation (JSON)

JavaScript Object Notation is a data format in the form of text from JavaScript derivatives which is lightweight in data exchange and easy for humans to read and write the format, also for machines to parse and generate [14]. JSON consists of two parts, key is a character that lies between two quotes and is in the form of a string, and value is in the form of an object, array, string [15]. Some of the advantages of JSON are its small file size, and its simple structure so that it is easily understood by humans [21].

2.5. REST API

REST API is an application program interface (API) based on representational state transfer (REST) that uses HTTP requests to perform post, get, put, patch, and delete methods. Post is used to create new resources or send data for processing to the server. Get is used to retrieve data from a server. Put is used to update data, Patch is used to change data, and delete is used to delete data from the server [19].

The REST API uses the status code from the HTTP response to report the client on the result of their request. HTTP specifies about forty standard status codes for sending the output of client requests, and is divided into five categories as follows; 1xx to show a temporary response, 2xx success, indicating request has been successfully received, 3xx to indicate more action the client needs to take to fulfill the request, 4xx indicating an error to the client, 5xx indicating the server is aware that it is unable to execute the request, and is responsible for it [20].

2.6. Mininet

Mininet is an emulator that can be used to form a network on a single machine consisting of a collection of hosts, switches, and links. The use of mininet is useful especially using OpenFlow and SDN in demo, development, or network testing [16]. Mininet works like a real machine that can run programs that can be installed on a linux system. Packets sent from the executable program can look like on an Ethernet interface with link speed and delay indicators. Some of the advantages of this emulator are starting a fast network, being able to create a custom topology, running programs that run on real machines, experimenting with using the command line interface or python scripts, and being open source [17].

2.7. CanvasJS

Javascript is a flexible language, many developers have provided tools to use other functions that can be implemented to the core of javascript, such as Application Programming Interfaces which allow interaction between applications, and libraries that can be combined with HTML to add features to websites or applications [23]. CanvasJS is a javascript library for graphic creation that uses a simple API and has good performance. The use of this library can be combined with many frameworks such as Vue, React, Angular, and supports various interactive features such as tooltips, and animation.

3. RESEARCH METHODS

This study aims to design a web interface to display the statistics of the SDN network and to compare the indicator of the southbound protocol, Openflow. This section will explain things related to this work, such as system design, and data collection. The explanation will be described in the following section

3.1. System Design

The interface that will be created is a web interface to display the statistics of the SDN network. Interface in the form of a web view that will retrieve JSON data from the ONOS controller, and fetching data from the controller is provided by REST API. The data to be retrieved are statistics in the form of packet received, packet sent, bytes received, and bytes sent. The data is in the form of a JSON file which will then be converted into graphics on the web GUI. The block diagram of the application design can be seen in the following figure

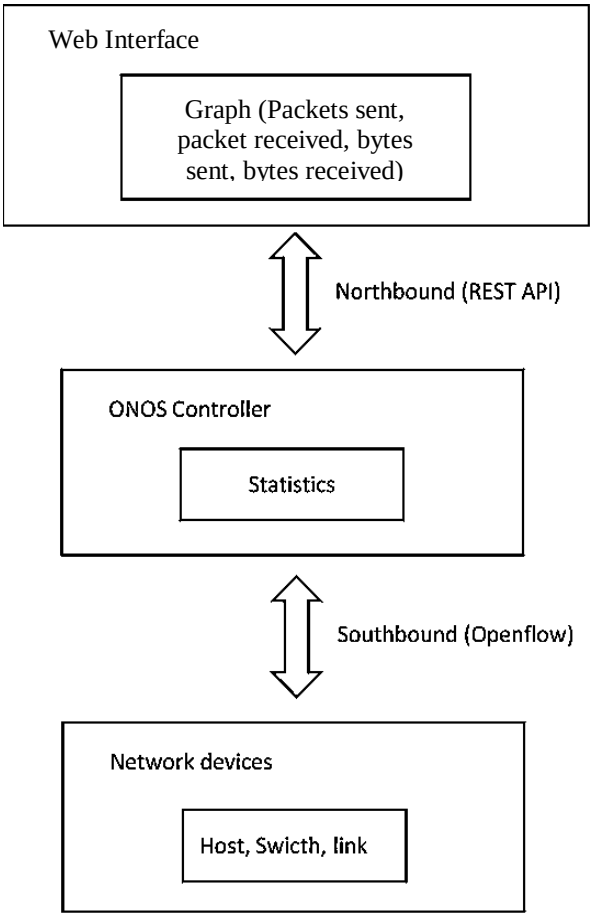


Figure 1. SDN-based system design block diagram

3.2. Data retrieval

REST API is used to connect to the interface to be created and controller. Applications that will be created use the canvasJS library to create graphics that will be displayed on the web [10]. The graph will display information from the data plane in the form of packets sent, packets received, bytes sent, bytes received. The data above will be used to compare the statistical indicator of the southbound protocol, then implemented in several topologies, ring, tree, and leafspine topologies. The end results of this study is to provide interface for openflow test in SDN.

4. RESULTS AND DISCUSSION

This section aims to present the results of Openflow's implementation, especially in different topologies. The network topology tested was ring topology, tree topology, and leafspine topology. The test results are described as follows

4.1. Openflow in ring topology

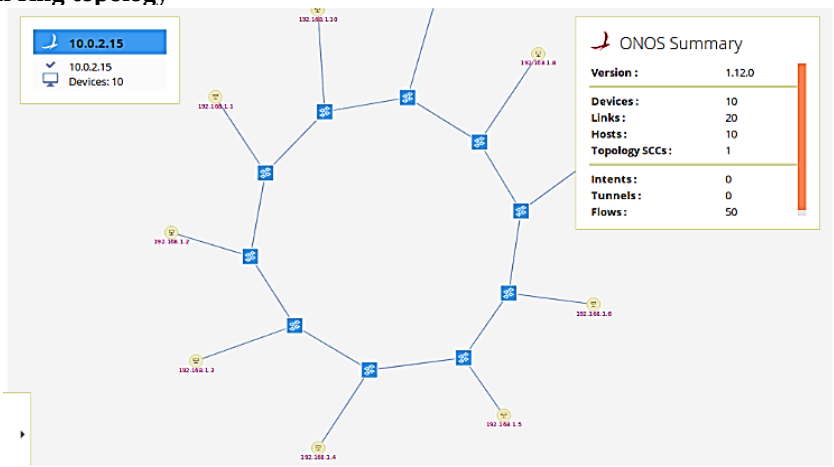


Figure 2. Implementation the first test in ring topology

In this test, 10 host and 10 switches were used and linked as figure 2, then ping was performed, and the results

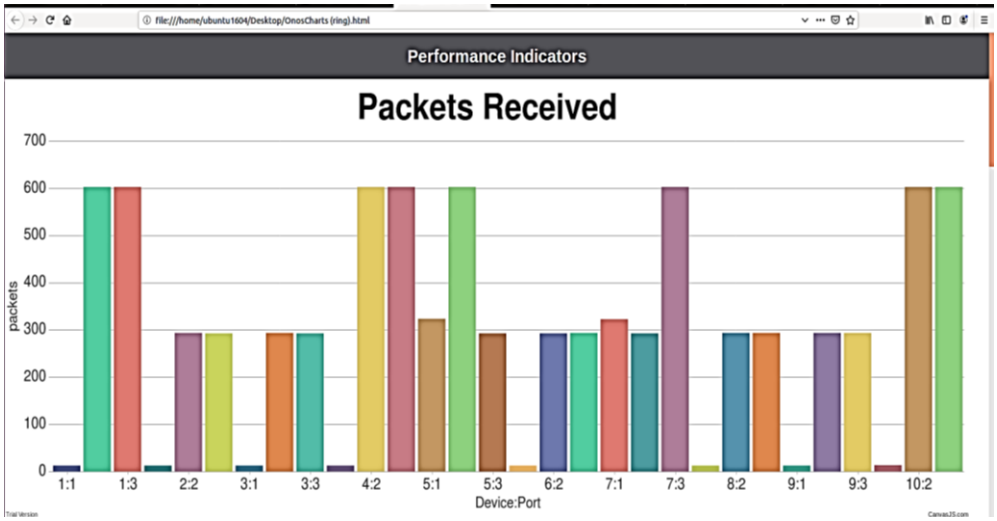


Figure 3. Interface shows the packets received in each device and its port

Figure 3 above shows the switch and port when sending ICMP packets or pinging from host 2 to host 8 for 5 minutes. From the display, it can be seen that the dominant switches and ports passed by the packet, namely switches 1, 4, 5, 7 and 10. From the results of sending ICMP packets from host 2 to host 8, the packet received by 10 port 3 switch is 54629 bytes for 407 seconds, so the throughput is 134.22 bytes/sec. From the iperf test, the result is that the bandwidth is 2.21Gbytes/sec, so that with the throughput 134.22bytes/sec, the utilization is $5{,}654\times10^{6}$%.

4.2. Openflow in tree topology

Ping h1 to h10 is done on network consisting of ten hosts and ten switches in tree topology as follows

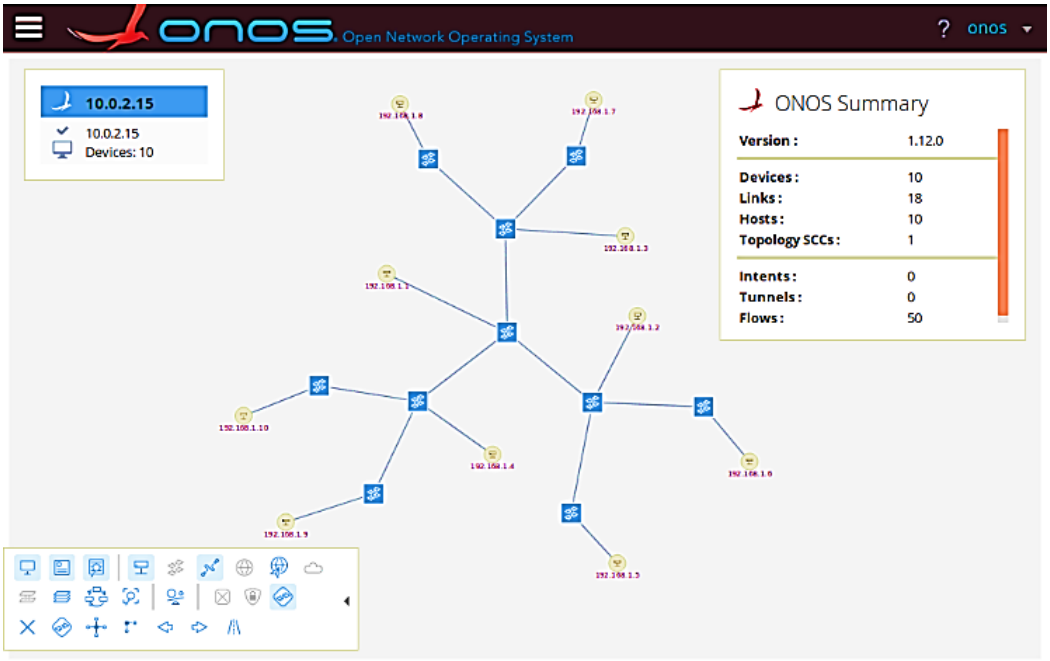


Figure 4. Implementation in tree topology

From the web interface of the received packets, the switch and port have a high number of packets, which is the link that the packet passes from host 1 to host 10.

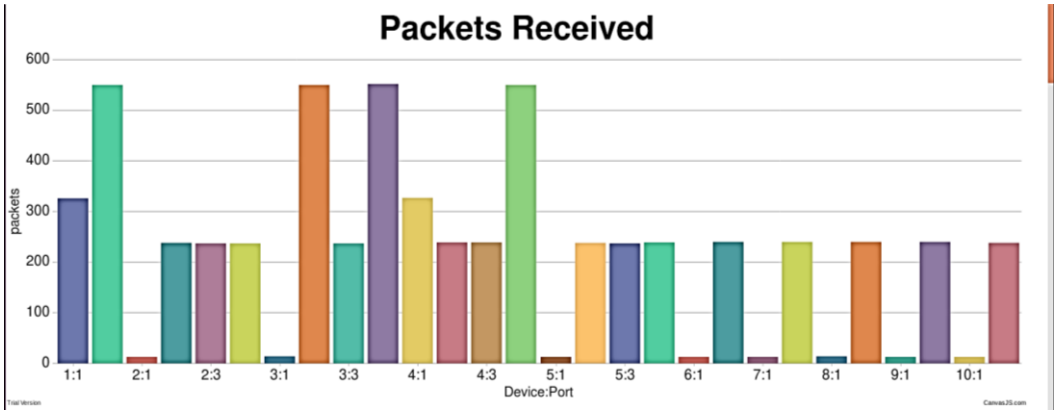


Figure 5. Packets received in tree topology

The size of the packet received on the switch 4 port 4 is 51803 bytes in 321 seconds, then the throughput is 159.13 bytes/sec. From the bandwidth test, it is known that the bandwidth between host 1 and host 10 is 4.525Gbytes/sec, with known throughput of 159.13bytes/sec and loss of 0, then the utilization is 3.275×10^{6}%.

4.3. Openflow in leafspine topology

The same experiment as before, sending ICMP packets on a network consisting of 10 hosts and 10 switches and forming a leafspine topology as in figure 6 below

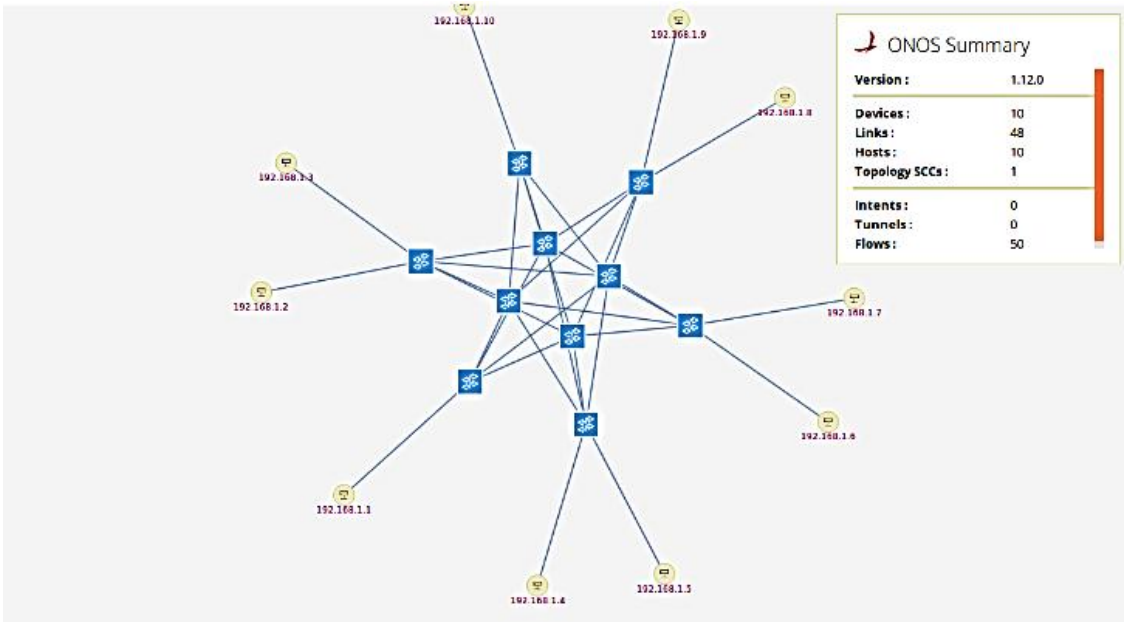


Figure 6. Implementation in leafspine topology

The picture below shows the results of the web interface for packets sent from host 3 to host 8. It can be seen that the dominant switches in the transmission are switches 4, 5, 9, and 10, which are the same as when the packet was received.

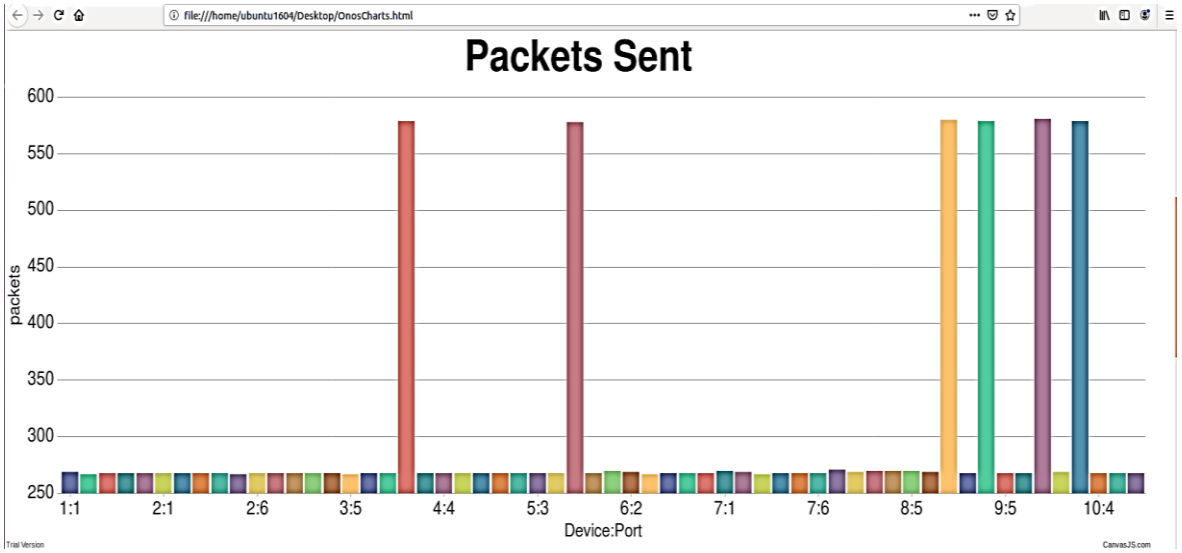


Figure 7. Statistic in leafspine topology.

There is a slight difference from the test results on the previous ring and tree topologies, where the results are almost the same between switches and ports when packets are sent and received. In this leafspine topology, the same switch sends and receives packets on different ports. The packet received on the switch 10 port 4 is 53429 bytes in 366 seconds, so the throughput is 145.98bytes/sec. and the result of bandwidth testing from host 3 to host 8 is 3,425 Gbytes/sec, so the utilization is 3.969×10^{6}%.

5. CONCLUSION

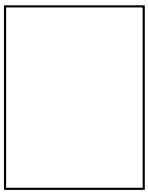
Based on the results and discussion of this study, it can be concluded as making the interface has been done by displaying statistical data from ONOS controller to the web interface in bar chart. Visualization performed with several topologies produces graphs in accordance with the statistics of each topology. The results of statistical visualization of each topology can be used to determine which switches and ports the packet passes when testing packet delivery, and when there is a down link, the switches and ports associated with the link will not see any packets passing. In all openflow tests across different network topologies, it was determined that the tree topology exhibited the lowest network capacity utilization at 3.275×10^{6}%, followed by the leaf-spine topology at 3.969×10^{6}%, and the ring topology at $5{,}649\times10^{6}$%, respectively.

REFERENCES

[1] Virtualization & Software Defined Networking, [Online] Available: https://speakerdeck.com/eueung/ppj-01-introduction.
[2] Kreutz, D., Ramos, F. M. V., Verissimo, P., Rothenberg, C. E., (2014): Software-Defined Networking: A Comprehensive Surver, IEEE.
[3] Software-Defined Networking (SDN), [Online] Available: https://www.opennetworking.org/sdn-definition/.
[4] Kim, H., Feamster, N., (2013): Improving Network Management with Software Defined Networking, IEEE Communications Magazine.
[5] Open Networking Foundation (2012): Software-Defined Networking: The New Norm for Networks, ONF White Paper.
[6] What is Software Defined Networking (SDN)? Definition [Online] Available: https://www.sdxcentral.com/networking/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/.
[7] ONOS, [Online] Available: https://wiki.onosproject.org/display/ONOS/Wiki+Home.
[8] ON.LAB (2014): Introducing ONOS – a SDN network operating system for Service Providers, ON.LAB Whitepaper.
[9] ONOS, [Online] Available: https://en.wikipedia.org/wiki/ONOS.
[10] Beautiful HTML5 Charts & Graphs, [Online] Available: https://canvasjs.com/.
[11] Pengantar SDN, [Online] Available: https://eueung.gitbooks.io/buku-komunitas-sdn-rg/content/pengantar_sdn/README.html.
[12] What is OpenFlow? Definition and How it Relates to SDN, [Online] Available:https://www.sdxcentral.com/networking/sdn/definitions/what-is-openflow/.
[13] Who is the Open Networking Foundation (ONF)? The steward of OpenFlow, [Online] Available: https://www.sdxcentral.com/networking/sdn/definitions/who-is-open-networking-foundation-onf/.
[14] Introduction JSON, [Online] Available: https://www.json.org/json-en.html.
[15] Penjelasan Singkat: Apa Itu JSON, [Online] Available: https://www.hostinger.co.id/tutorial/apa-itu-json/.
[16] Mininet: Rapid Prototyping for Software Defined Networks, [Online] Available: https://github.com/mininet/mininet.
[17] Introduction to Mininet, [Online] Available: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet.
[18] Mininet, [Online] Available: https://www.opennetworking.org/mininet/.
[19] JavaScript Charts & Graphs with Simple API, [Online] Available: https://canvasjs.com/javascript-charts/.
[20] Perbedaan antara API, REST API, dan RESTful API, [Online] Available: https://medium.com/jagoanhosting/perbedaan-antara-api-rest-api-dan-restful-api-6a66d655a6c2.
[21] HTTP Status Codes, [Online] Available: https://restfulapi.net/http-status-codes/.
[22] JSON: Pengertian, Fungsi dan Cara Menggunakannya, [Online] Available: https://www.niagahoster.co.id/blog/json-adalah/#Apa_Itu_JSON.
[23] ONOS, [Online] Available: https://wiki.onosproject.org/display/ONOS/ONOS.
[24] JavaScript, [Online] Available: https://developer.mozilla.org/id/docs/Learn/Getting_started_with_the_web/JavaScript_basics.

[25] What is SDN?, [Online] Available: https://www.ciena.com/insights/what-is/What-Is-SDN.html.
[26] Jarschel, M., Metter, C., Zinner, T., Gebert, S., Tran-Gia, P., (2014): OFCProbe: A Platform-Independent Tool for OpenFlow Controller Analysis, IEEE.
[27] Kang, H., Lee, S., Lee, C., Yoon, C., Shin, S., (2015): SPIRIT: A Framework for Profiling SDN, IEEE.
[28] Lee, C., Yoon, C., Shin, S., Cha, S., (2018): INDAGO: A New Framework For Detecting Malicious SDN Applications, IEEE.

BIOGRAPHIES OF AUTHORS



Fahrizal Djohar Attained his Bachelor of Engineering Degree in 2017 from the Department of Electrical Engineering at Universitas Ahmad Dahlan, followed by a Master of Science in Electrical Engineering from the School of Electrical Engineering and Informatics at Bandung Institute of Technology in 2021. Since 2022, he has served as a lecturer in the Faculty of Engineering at Universitas Khairun in Ternate, Indonesia. His research is centered on Communication Networks, and he can be reached via email at fahrizaldjohar@unkhair.ac.id.