Journal of Theoretical and Applied Information Technology
15th July 2018. Vol.96. No 13

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

OPENFLOW SWITCH SOFTWARE-BASED PERFORMANCE TEST ON ITS IMPLEMENTATION ON CAMPUS NETWORK

[1]RIKIE KARTADIE, [2]FAHRUR ROZI, [3]EMA UTAMI

[1,2]Information Technology Education Department , STKIP PGRI Tulungagung, Jl. Mayor Sujadi Timur No.7, Tulungagung, Indonesia

[3]Universitas Amikom Yogyakarta, Jl. RingRoad Utara, Condong catur, Yogyakarta, Indonesia

E-mail:[1]rikie.kartadie@stkippgritulungagung.ac.id, [2]fahrur.rozi@stkippgritulungagung.ac.id, [3]emma@nrar.net

ABSTRACT

OpenFlow experiments are conducted by researchers often used hardware/OpenFlow Switch issued by vendors. Actually, the performance of OpenFlow switch software-based (starting while switching software-based) was only tested on a laboratory scale. The problem to be raised in this research can be stated some problems as follows. How is OpenFlow software-based OpenWRT software performance when implemented into the Software-Defined Network (SDN) infrastructure on campus and is there a significant difference between mininet switch and prototype. In this study showed that the performance of which was owned by the OpenFlow switch-base software and can be implemented on campus. Testing OpenWRT OpenFlow software-based switching performance on campus implementation provides the resulting prototype latency value fluctuated quite diverse compared mininet with gap is 2.3361 msec, the average value of TCP and the absolute data gap and prototypes is 10.2114 KByte/second, and the average UDP value and the value of the data gap absolute mininet and prototypes is 151.419 KByte / second. Mininet switches compared to prototype switches do not give significant difference, so it can be said prototype successfully produced and can be implemented on campus network.

Keywords: *Implementation, OpenFlow, OpenFlow Switch, Performance, Software-Defined Network*

1. INTRODUCTION

The current network was the result of protocol and network design decisions made in the 1970s. Once established, the network topology is not expected to change much, probably will not change at all. However, in reality, the need for the network continues to grow and the network design continues to experience drastic changes.The networks are typically constructed from a large number of network devices such as routers, switches, and other devices. Each device runs the packet forwarding manipulation, with a complex protocol embedded within the device. Network operators are directly responsible for the configuration, rules, and not infrequently up to the applications used in the network. Carriers typically manual configuration on each device that is connected, it does give a gap in configuration errors because of human error (human error), especially when it has to handle the number of devices a lot. STKIP PGRI Tulungagung also realized the trend of rapid network development, thus requiring continuous adjustments and renewals.

Anticipate these conditions were answered with the advent of new architectures and protocols, called SDN/OpenFlow, and Suwadi, et al said that network coding is one technique to improve[1]. Software-Defined Network (SDN) emerged from research in 2004 as part of a new network management paradigm study, which resulted in two different results: the work of the routing control platform (RCP 40) completed at Princeton and Carnegie Mellon University under the auspices of Caesar et al. in 2005, and the security of the SANE Ethane project network completed at Stanford University and the University of California by Cassado et al. In 2006 [2]–[4].

To do the experiment of OpenFlow, researchers often use high-cost hardware/ dedicated OpenFlow switch released by vendors. Furthermore, there is also switch, namely OpenFlow switch software-based, that is the low-cost OpenFlow based on OpenWRT. In fact, the performance of OpenFlow switch software-based

Journal of Theoretical and Applied Information Technology
15th July 2018. Vol.96. No 13

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

(henceforth as OF switch software-based) was tested only on a lab scale. The results of previous research, OF-based software switches can be used to replace dedicated OpenFlow switches to be implemented on intermediate and campus networks based on throughput and jitter test results [5].

Research on switch OF software-based has been done on several types of researchers. Yik (2012) has been researching by making Switch OF software-based prototype with NetFPGA [6], while the researchers themselves have made prototypes using commercial switches with OpenWRT-based [5], [7], [8], Applement and Boer analyze the performance of hardware OpenFlow are like NetFPGA card, Pica8 OpenFlow on a Pronto switch and OpenvSwitch. The test is done with variables such as QoS, Port Mirroring, fail over speed and performance overhead [9], [10]. However, this study uses OpenWRT OpenFlow switch software-based which is lower cost and the testing done in this research is throughput and latency. Siebertz[11], in his thesis test OpenWRT as OpenFlow switch but in wireless mode, while the researcher made in wire mode. In the previous research OpenWRT switch software-based is tested in small scale that is lab, meanwhile we need to know how good is OpenWRT switch software-based if implemented in large scale such as campus network. If OpenWRT switch software-based can be implemented in campus network it will be reduce the cost but before that there are need to compared between mininet and prototype switches.

Based on the explanation above, the problem that will be raised in this study can be stated several problems as follow. How is the performance of OpenWRT switch software-based when implemented into a Software-Defined Network infrastructure on campus network. The major research focuses on throughput that can be generated and the resulting latency. The study compared the results obtained mininet with the results obtained the prototype.

2. RESEARCH METHOD

The research method used in this study is as follows: (1) Design topology that will be used both in the test mininet and test using prototype, (2) Modify Switch TP-Link which will be prototype with firmware OpenWRT and given OpenFlow agent [5], (3) testing mininet with a pre-designed topology and testing the throughput and latency of reference data, (4) performing the test throughput and latency of the prototype, (5) Analyzing the data generated by the comparison method. The expected result is that there is no significant difference between the mininet that will be the reference data of the prototype so that the prototype is successful and is declared acceptable. Research method can be seen in Figure 1.

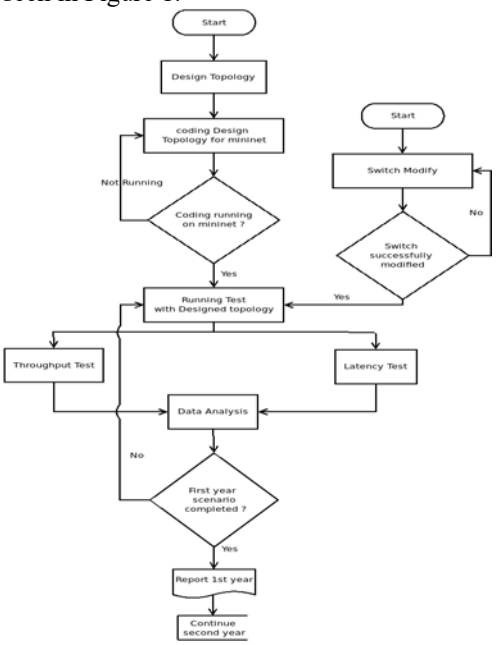


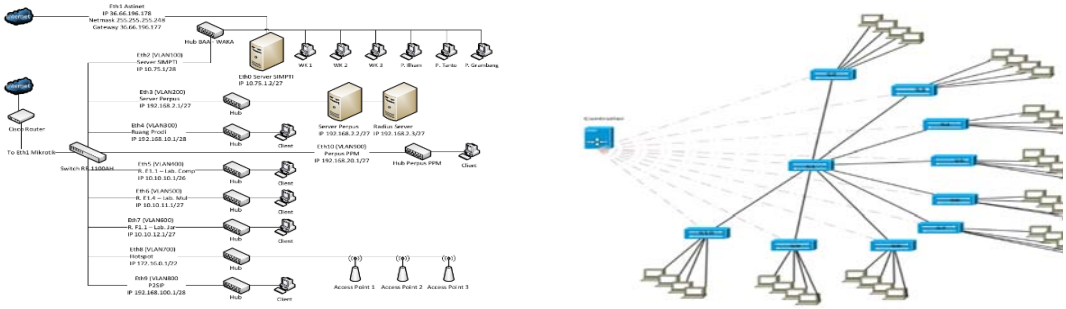
Figure 1. Research Method

A. *Topology Design*

In this study, researchers used topology which is used on campuses, although it does not describe the topology completely. STKIP has several node switches that serve the host/user dispersed throughout campus locations. 70% of the use of the network using a wireless connection; So, in this study, researchers focused only on the cable network in the campus environment. Cable network topology which is used on campus network as shown in Figure 2(a) and topology that is used for this study as shown in Figure 2(b). There are 10 switches used in a star topology and centered on switch no.1 as the center of the data stream.

Journal of Theoretical and Applied Information Technology
15th July 2018. Vol.96. No 13

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195



2(a). The campus network topology *2(b). topology that used for this study*

Figure 2. Topology Used In Research

The topology used in mininet runs a proposed (custom) topology (10 pieces of switches with 4 hosts per switch). Coding run with python language that can be executed by mininet, while coding for the topology that has been designed is as follow on figure 3.

```
"""
Sepuluh switch terkoneksi langsung,
dengan masing-masing
switch terkoneksi dengan 4 host:
"""

from mininet.topo import Topo

class MyTopo( Topo ):

"Topologi prototipe."
def __init__( self ):

"Membuat topologi."
# Initialize topology
Topo.__init__( self )

# Add hosts and switches
"""
Switch 1 dan 2
"""
Host-s1 = self.addHost( 'hs1-1' )
Host-s2 = self.addHost( 'hs1-2' )
Host-s3 = self.addHost( 'hs1-3' )
Host-s4 = self.addHost( 'hs1-4' )
Host-s5 = self.addHost( 'hs2-1' )
Host-s6 = self.addHost( 'hs2-2' )
Host-s7 = self.addHost( 'hs2-3' )
Host-s8 = self.addHost( 'hs2-4' )
  ..... Coding cut ....
        Switch1 = self.addSwitch( 's1' )
        Switch2 = self.addSwitch( 's2' )
        Switch3 = self.addSwitch( 's3' )
        Switch4 = self.addSwitch( 's4' )
        Switch5 = self.addSwitch( 's5' )
        Switch6 = self.addSwitch( 's6' )
        Switch7 = self.addSwitch( 's7' )
        Switch8 = self.addSwitch( 's8' )
        Switch9 = self.addSwitch( 's9' )

        Switch10 = self.addSwitch( 's10' )
# Add links
self.addLink( Switch1, Host-s1 )
self.addLink( Switch1, Host-s2 )
self.addLink( Switch1, Host-s3 )
self.addLink( Switch1, Host-s4 )
self.addLink( Switch2, Host-s5 )
self.addLink( Switch2, Host-s6 )
self.addLink( Switch2, Host-s7 )
self.addLink( Switch2, Host-s8 )
self.addLink( Switch3, Host-s9 )
self.addLink( Switch3, Host-s10 )

        .....Coding Cut ....
""" Link antar switch"""
self.addLink( Switch1, Switch2 )
self.addLink( Switch1, Switch3 )
self.addLink( Switch1, Switch4 )
self.addLink( Switch1, Switch5 )
self.addLink( Switch1, Switch6 )
self.addLink( Switch1, Switch7 )
self.addLink( Switch1, Switch8 )
self.addLink( Switch1, Switch9 )
self.addLink( Switch1, Switch10 )


topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Figure 3: Topology Coding Used on Mininet

B. Controller Used

The controller plays an important role in the SDN/OpenFlow network. There are many existing controllers, whether they are open-source or that are closed source or paid. According to Muntaner, controllers are a major factor in the SDN/OpenFlow network [10]. Because of the importance of the controller in this study, before performing the selection of the controller, the researcher conducts a predecessor study (literature study) with respect to the controller. In this study, the researcher chooses to use Floodlight controller with the consideration of this controller has the following features [12] : (1) Module loading system, which makes this controller has a simple system and easy to develop, (2) Easy to set with minimal dependencies, support switch virtual or physical, (3) Be able to recognize the mixed network of OpenFlow and non-OpenFlow, (4) Have high performance, because that floodlight becomes the core of commercial product design of big switch network, (5) OpenStack support (link) orchestration cloud platform. As mentioned earlier, Floodlight is a controller with modular architecture that includes modular device management, topology modules, load balancing, Web graphical user interface (Web GUI). The core module of floodlight is a module that can handle I/O from network devices and change OpenFlow messages [13].

In addition to the above features, researchers have conducted research on the performance of this controller that has been published with the result that this controller has a good latency on the number of switches under 60 pieces. The highest latency response at the switch condition is 20 pieces. The throughput performance of this controller is good too because it can handle 450 hosts with the performance of 1500.38 flow/second [8], in another study Floodlight

Journal of Theoretical and Applied Information Technology
15th July 2018. Vol.96. No 13

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

obtained an average performance of 81.863 flow/detik[14], however floodlight gives a good performance compared with other controllers. This is in line with Bholebawa's statement [15] that floodlight controllers are more efficient than other controllers.

C. Comparative test with Mininet

Mininet was used as an emulator to obtain comparison data in this study. Besides being mininet as the only emulator that can give an exact picture of the network SDN / OpenFlow [8], mininet also used in testing the controller that has been done on the sub-section 2.2 above. Installation mininet performed on Laptop with Intel i3-4005U CPU specification 170GHzx4, 4Gb Ram with Operation System ubuntu kernel 4.4.0-92-generic 16.04LTS. Installation mininet did with 2 treatments; (1) installing the Ubuntu OS kernel directly on the host laptop, (2) using Virtual Boxes in running mininet as OpenFlow switch. In this study, the researchers used the second way for the reason that if an error occurs in the installation does not damage the operating system kernel used. Corrections made in the running mininet will be easier to do. Topologies that have already been designed will run on mininet. The designed topology will run in mininet, run with the command

```
sudo mn --topo = mytopo --controller = remote, ip = 192.168.56.102, port = 6653 --custom = toporikie.py
```

Mininet run custom topology (10 pieces switch with 4 hosts per switch). Mininet topology which was executed later in the capture of data throughput and latency, and are used as comparative data. Throughput and Latency data are taken periodically for 10 weeks with an average value per week. Results from mininet and controller connections as shown in Figure 3.

After mininet simulated variables, the next step is to create a prototype network using OpenWrt switch. Switches used are TP-Link WR1043nd with hardware version 1.11 as many as

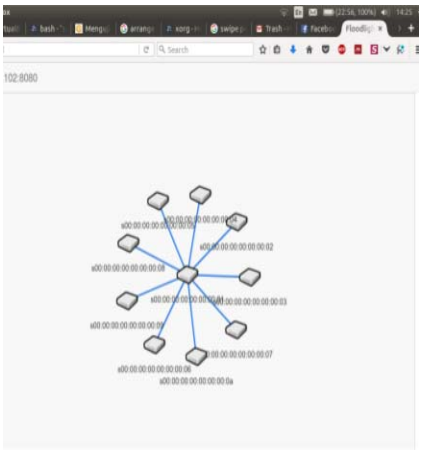


Figure 4: Test Topology (non-host) Described By Floodlight

10 pieces and compiled based on research topology shown in Figure 2. The prototype built is a software-based OpenFlow Switch prototype.

D. Modify Firmware

To modify the firmware, there are two ways you can do, (1) Modify the firmware from the firmware update menu on the switch. On the first modification, this is the experienced failure, because the firmware entered does not remove the original firmware available. The new firmware is entered only occupy space on the flash/ROM. (2) Modify the firmware by flashing/reinstalling on the ROM switch.

The switch has several pins that will be connected to the computer via TTL to USB cable. The pin (on the circuit notated P1) is pin serial consisting of 4 pins.Pin1 is Tx, pin 2 is Rx, pin 3 is ground and pin 4 is Vcc (voltage), pin 1 in Figure 5 are marked with a red circle.

The circuit path on the Rx pin on the circuit with the resistor R362 does not exist and resulted in not being able to enter into the uboot of the switch. To get into the uboot switch and make firmware modifications, it is necessary to add the path between R362 and Rx pin as done in figure 6.

Before the new firmware image is uploaded, the old firmware image is removed first, so there is enough space in the ROM switch to place the new firmware image. Once entered into the prompt ROM of the switch, which is marked with "ar7100>". The command given is "erase 0xf020000 + 7c0000", with this command ROM will delete the memory contents located at 0xf020000 to 0x7c0000, this memory location containing the old firmware image, then installed with OpenFlow firmware image.

Journal of Theoretical and Applied Information Technology
15th July 2018. Vol.96. No 13

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

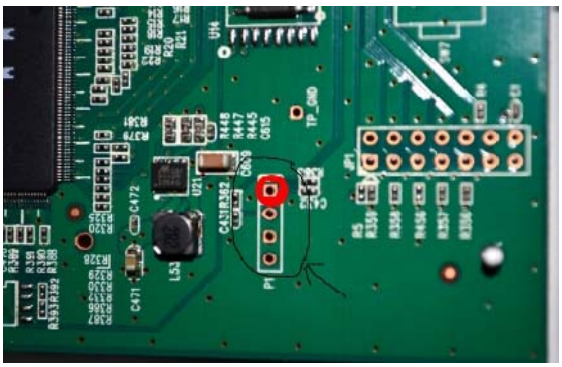


Figure 5: The Pins Are Connected To A PCc With A USB To TTL

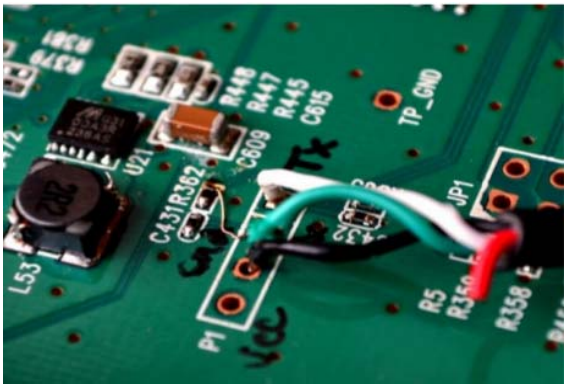


Figure 6: The Addition Of Circuit Lines On Rx Pin Serial With R362 To Get Into Uboot

The IP assignment of each switch is configured directly in the /etc/config/OpenFlow file as below, for switches 1 to 10 with different PIDs.

```
config 'ofswitch'
        option 'dp''dp0'
        option 'dpid''000000000001'
        option    'ofports''eth0.1    eth0.2
eth0.3 eth0.4'
                option'ofctl''tcp:192.168.56
.102:6653'
        option 'mode''outofband'
```

As for IP configuration is done in /etc/config/network with the following configuration:

```
... config 'interface'
        option 'ifname''eth0.5'
        option 'proto''static'
        option 'ipaddr''192.168.56.11'
        option 'netmask''255.255.255.0'
```

E. Testing latency and throughput

The latency tests performed on minerals and prototypes are performed on a large ICMP packet from 64 bytes to 8192 bytes. Tests carried out by repeating 10 times on each data packet transmitted ICMP. Throughput and latency testing were performed on mininet and prototypes did on the TCP protocol with the TCP window size from 128 Kbps to 1024kbps and UDP buffer size and bandwidth of 128 kbps up to 1024kbps. On the server side, iperf is setup with a configuration where large windows sizes are assigned variables ranging from 1024kBps.

```
iperf -s -P 0 -i 1 -p 5001 -w 1024K -f k
```

For UDP used settings starting from the same packet size as well.

```
iperf -s -u -P 0 -i 1 -p 5001 -w 1024K -f k
```

On the client side, iperf is configured

```
iperf -c 10.0.0.1 -P 1 -i 1 -p 5001 -w
            1024K -f K -t 10
```

For UDP on the client side the following settings are used

```
iperf -c 10.0.0.1 -u -P 1 -i 1 -p 5001 -w
1024K -f k -b 1024M -t 10 -T 1
```

3. RESULT AND ANALYSIS

Once the prototype switch ready switches arranged in accordance with the topology that was designed before. The data is retrieved with the same technique both mininet and prototype, the data were taken for 10 weeks, testing was conducted on the data of 64Byte to 8192byte.

A. Latency Test

Testing latency on mininet and prototype switch can be seen in table I and table II , in table I and II the results of trials on each switch are average results obtained in the experiment for 10 weeks and is the result of per-switch latency.

Journal of Theoretical and Applied Information Technology
15th July 2018. Vol.96. No 13

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 1: Latency on Mininet (msec)

Switch	64 byte	128 byte	256 byte	512 Byte	1024 byte	2048 byte	4096 byte	8192 byte
1	2,228	3,999	4,855	5,191	4,341	4,736	4,477	9,149
2	2,341	3,193	4,416	5,787	4,436	4,773	4,515	9,44
3	2,221	3,211	4,117	5,433	4,499	4,554	4,655	9,765
4	2,611	3,566	4,567	5,011	4,321	5,145	4,66	9,432
5	2,288	3,458	4,532	4,65	5,031	4,555	4,789	9,435
6	2,581	3,413	4,765	4,667	4,432	4,872	4,123	9,444
7	2,344	3,871	5,376	4,432	4,333	4,321	5,011	9,123
8	2,731	3,876	4,566	4,32	4,547	4,387	5,811	9,812
9	2,444	3,431	5,111	4,567	4,556	4,21	4,667	9,456
10	2,553	3,67	4,353	5,001	4,678	5,091	4,558	9,112
avg	2,434	3,569	4,666	4,906	4,517	4,664	4,727	9,417
stdev	0,177	0,280	0,372	0,467	0,213	0,316	0,444	0,243
var	0,031	0,078	0,139	0,218	0,046	0,100	0,197	0,059

Table 2: Latency on Prototype Switch (msec)

Switch	64 byte	128 byte	256 byte	512 byte	1024 byte	2048 byte	4096 byte	8192 byte
1	2.622	5.545	5.613	5.585	5.678	6.48	7.233	10.421
2	3.211	4.552	4.568	4.445	5.734	5.89	6.77	10.441
3	2.345	4.322	5.67	5.566	5.422	6.41	6.89	10.822
4	3.445	4.311	5.123	5.758	5.121	6.11	7.04	10.112
5	3.467	4.666	5.431	6.697	6.123	6.432	7.554	10.115
6	3.567	4.544	4.987	6.667	5.889	6.544	6.899	10.001
7	3.765	4.446	5.437	6.976	5.431	6.345	6.785	10.448
8	3.541	5.765	4.877	6.674	5.666	6.599	6.78	10.766
9	3.333	5.778	5.12	6.69	5.789	6.679	7.12	11.011
10	3.489	5.012	5.234	6.476	5.66	6	7.556	10.886
avg	3.279	4.894	5.206	6.153	5.651	6.349	7.063	10.502
stdev	0.448	0.590	0.344	0.792	0.277	0.264	0.301	0.357
var	0.201	0.349	0.118	0.627	0.076	0.070	0.091	0.127

The highest average mininet latency as shown in table I, occurs on a large 512byte data packet with the value of 5.001 msec and will have a very significant increase in data packets of 8192byte. In the table above, it is also seen that the existing standard deviation has the highest value on the 512byte packet size. Data variance on the package size has a value of 0.218.

The latency value obtained by the prototype has decreased in the 512byte packet the average value obtained is 6.153 msec although it then rises again in accordance with the size of the data packet. Data variance on the package size has a value of 0.076.The prototype has provided results close to the results of mininet, the average throughput value and the value of the data gap absolute mininet and prototypes can be seen in table 3 and figure 8 .

Journal of Theoretical and Applied Information Technology
15th July 2018. Vol.96. No 13

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 3: Latency Average and Gap (in msec)

	64 byte	128 byte	256 byte	512 byte	1024 byte	2048 byte	4096 byte	8192 byte
Avg. Prototipe	3.2785	4.8941	5.206	6.1534	5.6513	6.3489	7.0627	10.5023
Avg.Mininet	2.4342	3.5688	4.6658	4.9059	4.5174	4.6644	4.7266	9.4168
Gap	0.8443	1.3253	0.5402	1.2475	1.1339	1.6845	2.3361	1.0855

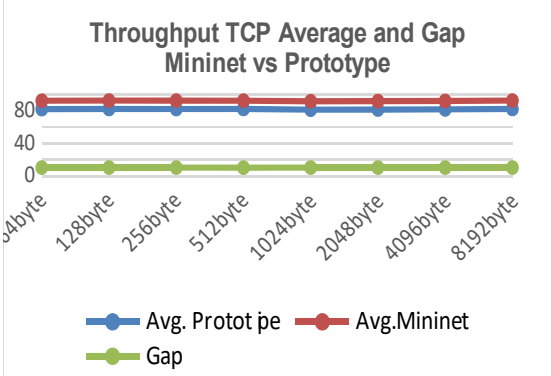


Figure 7: Latency Average And Gap Between Mininet And Prototype

In table 3 and figure 7 , it can be seen that the highest gap value is in the 4096 byte data packet, where the prototype produces the highest average latency value compared to mininet. The resulting prototype latency value fluctuated quite diverse compared mininet. The latency value in the 8192byte packet (2.3361 msec) has a high increase but has a low gap.

B. Throughput TCP Test

For throughput testing, both mininet and prototype switches are tested on TCP and UDP throughput.The result of TCP throughput testing can be seen in table 4 and table 5.

Table 4: Throughput TCP on Mininet (KByte/sec)

Switch	64 byte	128 byte	256 byte	512 byte	1024 byte	2048 byte	4096 byte	8192 byte
1	91,984	91,983	92,085	91,533	92,075	90,639	90,966	91,962
2	92,042	90,586	91,421	91,872	91,294	90,763	91,022	92,062
3	91,963	93,042	92,961	92,982	91,874	91,659	92,288	91,656
4	91,475	92,142	92,236	91,012	91,644	92,273	91,942	92,966
5	92,963	92,845	92,983	92,481	91,686	91,783	93,074	93,048
6	92,983	93,094	92,942	93,021	90,573	92,073	92,966	93,077
7	91,633	92,957	92,973	93,073	91,943	92,155	91,633	91,631
8	92,984	92,963	91,576	92,665	90,852	91,422	91,474	93,181
9	90,962	90,944	90,942	91,482	90,564	90,969	90,961	91,132
10	91,983	92,074	91,672	91,312	91,982	92,055	91,652	91,762
avg	92,097	92,263	92,179	92,143	91,449	91,579	91,798	92,248
stdev	0,688	0,900	0,761	0,787	0,589	0,604	0,775	0,749
var	0,473	0,810	0,579	0,620	0,347	0,364	0,600	0,560

The mininet TCP throughput value in table 4, the average value of the measurement indicates a relatively stable value, the largest data variance value in the 128byte data packet size is 0.810. In the data 1024byte average value of TCP throughput decreased, though not large, the resulting value is 91.449 KByte/second.

The throughput TCP shown by the prototype experiences the same properties as those shown by mininet. Average throughput TCP has generated a stable value despite a decline in large data packets 8192byte with the value 81.938 KByte/second, but then give stable results on

Journal of Theoretical and Applied Information Technology
15th July 2018. Vol.96. No 13

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

large data packets further. The standard deviation of the prototype in table 5 shows the value approximately equal to the value generated mininet. The data packet of 128byte prototype has the same data variant with mininet of 0.810.

The prototype has given identical results as the mininet result, the average value of TCP and the absolute data gap and prototypes can be seen in table 6 and figure 8. With the resulting gap values, the prototype has no significant difference in this throughput test, although there is a difference in the 512byte data packet with 10.2114 KByte/second gap.

Table 5: Throughput TCP on Prototype (KByte/sec)

Switch	64 byte	128 byte	256 byte	512 byte	1024 byte	2048 byte	4096 byte	8192 byte
1	81,674	81,673	81,775	81,21	81,765	80,329	80,656	81,652
2	81,732	80,276	81,111	81,223	80,984	80,453	80,712	81,752
3	81,653	82,732	82,651	82,657	81,564	81,349	81,978	81,346
4	81,165	81,832	81,926	81,562	81,334	81,963	81,632	82,656
5	82,653	82,535	82,673	82,171	81,376	81,473	82,764	82,738
6	82,673	82,784	82,632	82,711	80,263	81,763	82,656	82,767
7	81,323	82,647	82,663	82,763	81,633	81,845	81,323	81,321
8	82,674	82,653	81,266	82,553	80,542	81,112	81,164	82,871
9	80,652	80,634	80,632	81,172	80,254	80,659	80,651	80,822
10	81,673	81,764	81,362	81,297	81,672	81,745	81,342	81,452
avg	81,787	81,953	81,869	81,932	81,139	81,269	81,488	81,938
stdev	0,688	0,900	0,761	0,700	0,589	0,604	0,775	0,749
var	0,473	0,810	0,579	0,490	0,347	0,364	0,600	0,560

Table 6: Throughput TCP on Prototype (KByte/sec)

	64 byte	128 byte	256 byte	512 byte	1024 byte	2048 byte	4096 byte	8192 byte
Avg. Prototipe	81.7872	81.953	81.8691	81.9319	81.1387	81.2691	81.4878	81.9377
Avg.Mininet	92.0972	92.263	92.1791	92.1433	91.4487	91.5791	91.7978	92.2477
Gap	10.31	10.31	10.31	10.2114	10.31	10.31	10.31	10.31

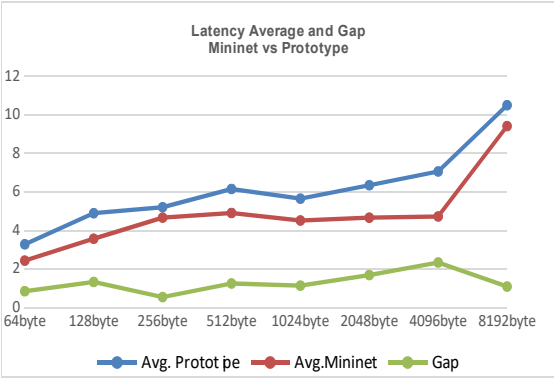


Figure 8: Throughput TCP Average And Gap Between Mininet And Prototype

C. Throughput UDP Test

For throughput testing, both mininet and prototype switches are tested on UDP and TCP throughput. The result of TCP throughput testing can be seen in table 7 and table 8 .

The mininet TCP throughput value in table 8, the average value of the measurement indicates a relatively stable value. The highest value is shown on the packet size of 8192byte with an average value of 177.606 Kbyte/second.

The throughput UDP shown by the prototype experiences the same properties as those shown by mininet. The increase in

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

throughput due to a large increase in data packets. The highest throughput value on packet size 8192byte with value 151.419 KByte / second.

Prototype has provided results close to the results of mininet, the average UDP value and the value of the data gap absolute mininet and prototypes can be seen in table 9 and figure 9 .

Table 7: Throughput UDP on Mininet (KByte/sec)

Switch	64 byte	128 Byte	256 byte	512 byte	1024 byte	2048 byte	4096 byte	8192 byte
1	14.390	15.600	33.854	69.771	144.542	156.921	177.000	187.788
2	14.090	15.300	33.133	69.331	140.200	173.101	173.053	184.376
3	14.230	15.440	33.094	70.167	143.110	158.091	176.000	185.968
4	14.010	15.220	32.632	69.197	140.654	151.000	172.187	183.466
5	13.800	15.010	32.191	68.271	140.976	149.279	164.876	175.790
6	12.977	15.000	32.170	68.227	141.654	150.550	162.116	172.892
7	12.880	15.345	32.800	69.550	141.857	156.712	173.053	183.376
8	12.981	14.191	30.471	64.659	140.665	135.037	149.211	159.341
9	12.120	13.330	28.663	60.862	144.542	137.313	151.714	161.969
10	12.089	15.011	32.193	68.276	140.357	150.78	169.923	181.089
avg	13.357	14.945	32.120	67.831	141.856	151.878	166.913	177.606
stdev	0.859	0.685	1.505	2.896	1.656	10.771	9.821	10.016
var	0.738	0.469	2.265	8.385	2.741	116.010	96.461	100.315

Table 8: Throughput UDP on Prototype (KByte/sec)

Switch	64 byte	128 byte	256 byte	512 byte	1024 byte	2048 byte	4096 byte	8192 byte
1	14.621	15.441	30.394	67.796	139.992	146.752	150.427	154.176
2	14.780	15.841	31.720	68.470	141.378	148.197	151.901	155.679
3	14.535	15.355	30.218	67.431	139.243	145.970	149.630	153.362
4	13.511	14.331	29.119	63.085	130.317	136.661	140.134	143.677
5	13.740	14.310	29.588	64.057	132.313	138.743	142.258	145.843
6	15.094	16.345	30.708	68.445	141.326	148.143	151.846	155.623
7	14.676	15.496	30.507	68.029	140.472	147.252	150.937	154.696
8	15.337	15.981	31.611	68.245	140.916	147.716	151.410	155.178
9	14.444	14.560	29.588	64.057	132.313	138.743	142.258	145.843
10	14.192	15.012	30.515	65.975	136.253	142.852	146.449	150.118
avg	14.493	15.267	30.397	66.559	137.452	144.103	147.725	151.419
stdev	0.561	0.703	0.838	2.093	4.299	4.484	4.574	4.665
var	0.315	0.494	0.703	4.381	18.484	20.107	20.920	21.765

Journal of Theoretical and Applied Information Technology
15th July 2018. Vol.96. No 13

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 9: Throughput UDP Average and Gap.

	64 byte	128 byte	256 byte	512 byte	1024 Byte	2048 byte	4096 byte	8192 byte
Avg. Prototipe	13.357	14.945	32.120	67.831	141.856	151.878	166.913	177.606
Avg.Mininet	14.493	15.267	30.397	66.559	137.452	144.103	147.725	151.419
Gap	1.136	0.322	1.723	1.272	4.404	7.775	19.188	26.187

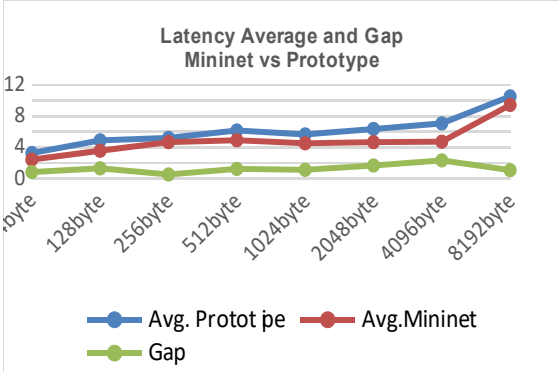


Figure 9: Throughput UDP Average And Gap Between Mininet And Prototype

Based on three test, that is latency: throughput TCP, and throughput UDP on prototype switch and mininet switch produce minimal gap between them. This happened because there are no significance differences in using prototype switch or mininet switch. Therefore OpenWRT switch software-based can be implemented in campus network.

4. CONCLUSIONS

In this study, it can be seen that OpenWRT OpenFlow software-based switching performance on campus implementation provides satisfactory results, in some cases may still need further development and research. Whereas when compared with mininet, prototype switches do not give significant difference, so it can be said prototype successfully produced and can be implemented further. Testing OpenWRT OpenFlow software-based switching performance on campus implementation provides the resulting prototype latency value fluctuated quite diverse compared mininet with gap is 2.3361 msec, the average value of TCP and the absolute data gap and prototypes is 10.2114 KByte/second, and the average UDP value and the value of the data gap absolute mininet and prototypes is 151.419 KByte / second. Based on the result we can conclude that OpenWRT switch software-based can be implemented in campus network. The prototype provides the same habits with mininet on latency but needs further evaluation on throughput because prototype habit and mininet habit is quite different.

ACKNOWLEDGMENT

This project was financially supported by a Research Grant from the Indonesian government through the ministry of research - technology and higher education with PKPT research scheme. We thank our colleagues from STKIP PGRI Tulungagung and Universitas Amikom Yogyakarta especially for Prof. Ema Utami who provided insight and expertise that greatly assisted the research.

REFERENCES:

[1] S. Suwadi, E. Endroyono, T. Suryani, and U. Khair, '*Performance Of Cooperative Communication System With Network Coding Using Software Defined Radio*', J. Theor. Appl. Inf. Technol., vol. 96, no. 1, 2018.

[2] P. Morreala and J. Anderson, *Software Defined Networking*, 1st ed. Boca Raton, FL: CRC Press, 2015.

[3] J. Chen, X. Zheng, and C. Rong, 'Survey on software-defined networking', *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9106, no. 1, pp. 115–124, 2015.

[4] A. Gelberger, N. Yemini, and R. Giladi, 'Performance analysis of Software-Defined Networking (SDN)', in *Proceedings - IEEE Computer Society's Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, MASCOTS*, 2013, pp. 389–393.

[5] R. Kartadie and B. Satya, '*Uji Performa Implementasi Software-Based OpenFlow Switch Berbasis OpenWRT Pada Infrastruktur Software-Defined*

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Network', *DASI*, vol. 16, no. 3, p. 87, 2015.

[6] E. C. Yik, '*Implementation of an Open Flow Switch on Netfpga*', Universiti Teknologi Malaysia, 2012.

[7] G. Pujolle, *Software Network-Virtualization, SDN, 5G, and Security*, 1st ed. Hpboken, NJ: ISTE Ltd And Jhon Wiley & Sons, Inc., 2015.

[8] R. Kartadie, T. Suryanto, '*Uji performa software-based openflow switch berbasis openwrt*', *J. Ilm. SISFONETIKA*, vol. 5, no. 2, pp. 130–142, 2015.

[9] M. Appelman and M. De Boer, '*Performance Analysis of OpenFlow Hardware*', University of Amsterdam, 2012.

[10] G. R. D. T. Muntaner, '*Evaluation of OpenFlow Controllers*', 2012.

[11] F. Siebertz and P. K. Jonas, '*Masterprojekt Software Defined Networking*', University of Applied Sciences, 2014.

[12] 'Floodlightt', *Project Floodlight*, 2018. [Online]. Available: http://www.projectfloodlight.org/floodlight/. [Accessed: 17-Jan-2018].

[13] H. Akcay and D. Yiltas-Kaplan, '*Web-Based User Interface for the Floodlight SDN Controller*', Int. J. Adv. Netw. Appl. Vol., vol. 8, no. 5, pp. 3175–3180, 2017.

[14] S. M. Anggara, '*Pengujian Performa Kontroler Software-defined Network (SDN): POX dan Floodlight*', Institut Teknologi Bandung, 2015.

[15] I. Z. Bholebawa and U. D. Dalal, '*Performance Analysis of SDN / OpenFlow Controllers : POX Versus Floodlight*', Wirel. Pers. Commun., no. September, 2017.