# Performance Evaluation of SDN Controllers: FloodLight, POX and NOX

**Mohamed Eltaj, Ahmed Hassan M. Hassan**

*Abstract*-**Networking technologies achieved an enormous jump toward an appealing notion, known as software-defined networking (SDN). to the best of our knowledge, there has been no comprehensive compression discussion about floodlight, POX and NOX controller in the literature. the aim of this research is to evaluate different types of a controller according to various parameters such as average TCP/UDP throughput, average bandwidth, packet loss, latency, topology discovery time and prediction inspection. we did a series of simulation studies in the mininet framework. it was found that the floodlight controller shows best performance (throughput) in a tree topology with congestion window size 32 mb, and a poor performance in linear and custom topologies, this result motivates extra experiments to investigate floodlight, we test the controller with different congestion window sizes 2, 20 and 32 mb, best performance recorded for 2 mb window size in a linear topology. POX and NOX controllers record best throughput results than floodlight, in all topologies, especially POX controller which scored best throughput in a custom topology. in UDP bandwidth investigation POX and NOX performed better with higher bandwidth utilization, while floodlight shows modest performance in return. lost packet tests, reveal that the highest rate of lost packets was recorded by floodlight with a significant difference between all tested controllers. latency test concluded with performance capabilities for responding to messages_in, POX controller scored best result with highest response per milliseconds. topology discovery time results shows that floodlight controller is the fastest in all topologies, especially in tree topology. the prediction of controller succeeded with POX controller in a throughput test, which reflects stability in a controller performance, unlike latency prediction which failed against POX.**

***INDEX TERMS—****SDN, OpenFlow, Floodlight, POX, NOX.*

## 1 INTRODUCTION

SDN for short is a programmable solution for customizing labor networks services and routines at run time, along with OpenFlow network communication protocol [1]. It requires networking platform provided with OpenFlow enabled devices. This paradigm is only completed with the interference of an intelligent entity known as "***SDN controller***". SDN Controllers introduced as an operating system for the network, to utilize underlying services. An "ethernet switch-based" protocol "OpenFlow" works as a coordinator between a controller and network switches. Obviously the "coordinator" used with separated entities; in SDN networking those entities are control and data planes. Traditionally network devices supported with built-in control unit for taking decisions in routing jobs.

Evaluating the SDN controller takes into account the controller's efficiency perspective. This consideration requires necessary measures for evaluating, throughput and latency. In this context throughput and latency are the very essence of evaluation. Throughput measures and expresses the usage of bandwidth between connected devices, while latency denotes controller' "request-response" time interval. Nemours number of benchmarking experiments can be performed to estimate performance of controllers under investigation. A selection of configured tests introduced in our experiments, the output shows interesting evaluation results.

The significant of our project is to performing wide range of experiments in a controlled environment, Filling the knowledge gap helps to spread the use of promising technology such as SDN, Enrich the experience of administrating SDN networks at least in the local area, documenting straightforward testing methods and result visualization, Predicting SDN network behavior is an appealing solution for optimizing the usage of business resources, and enhancing quality of service.

In the following sections, we state the details related to the related work in section II, the methodology of our article in section III, and then the system model in section IV with the results and discussion are providing in Section V. Finally, Section VI describes the details related to the conclusion and recommendations.

## 2 RELATED WORK

The need for modeling hosts, switches and link can be satisfied by using an open flow network emulation software such as ofent, Mininet, Omnet, etc.... Ofent and Mininet are software for modeling fake openflow networks suggested by many researchers, they motivated

**Mohamed Eltaj**, [1]Department of Computer and Information Technology, Mashreq University, Sudan.

**Ahmed Hassan**, Department of Telecommunication Engineering, Mashreq University, Sudan

by the ability of designing large networks as a testing environment [2] and [3] respectively. In same objectives of utilizing emulation applications for experimenting with networks, a SDN with OpenFlow [4], reviewed for implementing "OpenFlow laboratory", in a virtual environment, Mininet used as an emulation application. The author describes resources needed for implementing a test-bed, such as hardware, applications and virtualization technology was used, in this article we adopted same implementation with regard to hardware differences, the Test-Bed section describes our implementation.

In reviewed research [4] , the researcher describes experiments included software known as Wireshark [5] as a port capturing tool, the author uses the software to capture packets from test-bed interfaces, the software capable of rendering packets according to specific filter. Filtering Wireshark captured-packets allows analyzing involved TCP/IP protocols interactively, such as DHCP, ICPM, ARP and OpenFlow protocol as well, for example: in a response to a Ping request issued from on host to another, this request can be used to calculate discovery time for the topology, by recognizing messages used for discovery processes, Wireshark enables referencing captured packets according to specific filter selection, in a time-line fashion. This method used in our experiments to compute topology discovery time for SDN controllers. An adopted technique in this research, is to communicate with ports for analyzing traffic, packet analysis tools, are frequently present in SDN controllers testing environments.

In SDN controller, the topology has been discovered in [6]. Furthermore, a discovery time "can be treated as an interval", begins with the first discovery message and ends with last topology discovery message [6]. Our analysis utilizes increasing switches gradually, specifically in discovery time test, for the all tested SDN controllers. The controller's required time for handling a packet_in, denoted as latency. An observation for Floodlight controller tested by sending Packets_In and waiting for a response, the controller evaluated with different number of switches, the experiment came up with a slight increase in response to packet in requests [7]. In a study conducting the Denial-Of-Service impacts, which can be measured by calculating dropped packets, an experiment concluded that packets dropped decreases as the control plane bandwidth increases [8], this experiment used to measure several aspects in our research, bandwidth and packet loss averages in particular. Experiment calculating bandwidth, the author uses TCP streams to calculate "Achievable bandwidth", the mechanism for calculating bandwidth depends on tracking the total data transferred between hosts over time, the author uses iperf [9], [10]. Floodlight SDN controller has been investigated with an experiment with 8, 16 and 32 switches, result show that a slight increase in response occurs, the Cbench tool used for this experiment. The most presented measure in controller investigations is how much data can be consistently exchanged from end-point to end-point. This measure is a fundamental in SDN controller's efficiency benchmarking, throughput reviewed by many researchers in the field of open flow networks, reviewed techniques employed in our tests, for example a paper [11] about evaluating open flow networks, the author in a throughput test experiments with streams of packet_in messages sent from all open flow emulated switches in a continuous manner, the test result revealed that the controller incapable of processing all the messages in real-time, the controller reported a full-load state. The data collected by tests, therefore, will be handled by statistical calculations, to produce meaningful results. Statistical equations and line equation reviewed as well as regression analysis methods. An elementary knowledge for the best-fit line regression test reviewed in statistical resources, to implement a special investigation about SDN Controllers behavior predictability.

Investigating SDN controllers performance builds upon a well understanding of an SDN modeling, depending on an abstracted architecture adopted by several SDN controller developers and SDN researchers by far, an article published by future internet journal [12] adopted a decent architecture which will be used to focus the area of the interest upon the three SDN layers: application layer, control layer and infrastructure layer. Figure 1 describes architecture of SDN technology in a simple abstracted fashion.

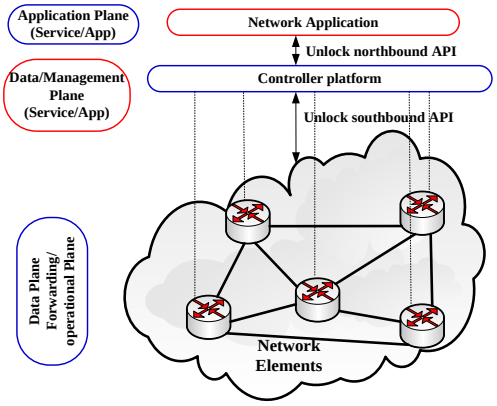



**Figure 1 SDN Structure**

The Black-Box testing method conducted in experimenting SDN controllers, the internal design of SDN controllers is out of the scope of this research, keeping in mind the fact that the SDN controller is a

software application, not a network device. The methodology used in this experimental research insures reflecting the efficiency of an SDN controller, a software testing methodologies involved in a certain degree to saturate the scope of the research, the theory and practice introduced in [12] is adopted to investigate the SDN controllers.

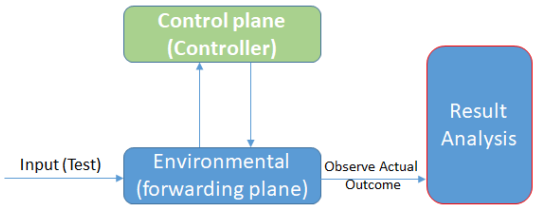



**Figure 2 Testing processes flow diagram**

The outcome of tests performed will result in lists of entries generated by a specific application such as iPerf, Ping or Cbench. For example, testing a burst of ICMP message sends from host to host by using iPerf application, iPerf configured to send ICMP message in time interval manner (almost 100 second), iPerf sends messages from one host to another using the full capacity of the link, this scenario in the context of efficiency can reveal the performance of the network under the administration of specific SDN controller, which reflects controller's capabilities of handling network's events and demands.

The amount of data collected by tests can be interprets to some measures of interest, knowing that various conclusions can be derived from the same raw data, in this research the major concern focus on several measures of performance such as throughput, latency, topology discovery time, packet lost rate and bandwidth utilization impacts. Recall that, Black-Box method only aware of the outcome of the execution of controller, without considering internal details of the controller as a program, only functionality and features will be considered [13]. The methodology used in this research aimed to experiment with different SDN controllers in a different network configuration and link capacity, through controller's south-bound interface to an emulated network, to facilitate the mission of suggesting an adequate controller with lesser effort, by proposing technical mechanisms as an implementation of an evaluation of SDN controllers. Evaluating SDN controllers is an amalgamation of benchmarking applications and techniques, that is. To perform an SDN controller investigation, test-bed should be prepared and equipped as needed for experimenting, minimum hardware and suitable operating system should be present, benchmarking tools is required, techniques and scenarios to follow are essentials, then calculating averages and measures of desperation, finally plotting results and stating conclusions. This is a suggested point of view answering "how evaluating a controller can be managed".

In this research, preparing Linux environment for installing test-bed and controllers consumes precious time, a good knowledge in driving Linux is generally required for networking related jobs, no farther knowledge about Linux references reviewed for installing, configuring or any other similar processes will be discussed in this research section, we considered it as out of scope, nevertheless, we provide some time saving hints in a test-bed settings section later on.

A common practice in networking experiments is to emulate networks, for a variety of reasons; they are inexpensive and affordable besides their ease in the configuration. Network emulators enable performing multiple different scenarios; with the most extreme topology can be imagined. Mininet an open source network emulator, suggested frequently by researchers [3], we found this emulator is very handy, especially when emulated custom networks.

Internet protocols is a prime aspect, observing their messages enable network monitoring, in this paper, we depend on internet protocols and "OpenFlow" protocol to analyze traffic over an emulated network. ***Wireshark*** is a free packet capturing tool employed in our research in multiple experiment for recording timestamps, which used for calculating time consumed for message processing by controller. Analyzed data by ***Wireshark*** and other benchmarking tool, translated to measures: firstly, "***Throughput***", measures the most extreme burst for sending data between end-points, additionally can be depicted as an amount of data travelled over time (kbyte/sec). Secondly, "***Latency***", measures SDN controller's responses to packet_in messages per second(resp/sec). Thirdly, "***Topology Discovery Time"***, measures the interval length, started from first discovery message, ending with last discovery messaged received by controller [6].

## 3 SYSTEM MODELING

Network Topology is a physical and logical design of a network. Physically, topology is an arrangement "mapping" of links, nodes, switches and other network equipment's used for constructing a network, while logical design is representing actual data transferring in opposite to the specific physical design. SDN controller can be connected to whatever topology design as long as the topology is used to form a packet-switched network with an OpenFlow enabled devices.

## 3.1 Network Topology

In this experimental article, network topology variations are the most considered attribute of tests, which categorize all tests performed on controllers. Network ubiquitous models constructed by Mininet, such as linear and tree, while Mininet is used for custom topology emulation.

### A) Linear Topology:

Linear topology in the context of SDN networking is a network configuration consists of "back-to-back" connected switches with a single host connected to each. The command used to instruct Mininet to construct linear topology is ***($ mn --controlle=remote,ip=127.0.0.1,port=6633 –topo=linear,7 –switch=ovs*).**

Figure 3 shows the linear topology constructed with Mininet.

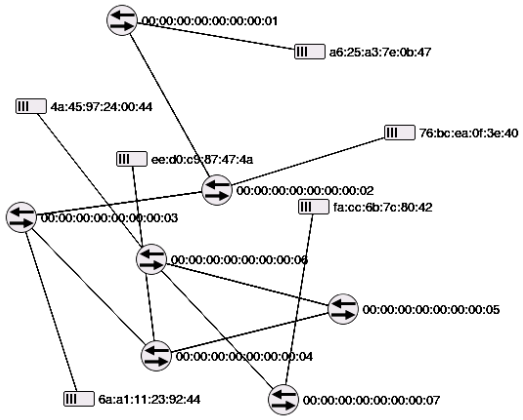



**Figure 3 Linear Topology**

### B) Tree Topology:

A tree topology can be configured according to "depth" and "fanout" attributes, the "depth" attribute specifies the number of switches connected to a core switch, where is "fanout" denote the number of connected switches to each "leaf/edge" switch. Figure 4 present the tree topology with 7-switchs and 12-hosts.

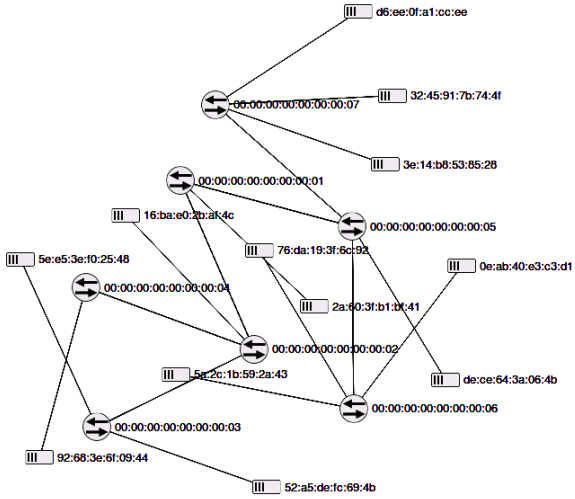



**Figure 4 Tree Topology**

### C) Custom Topology:

In custom topology, a model construction is derived from a top-level-node which in our case is an SDN controller application running on intel machine, multiple controllers allow SDN networking practice and they share the same trend of the need of being capable of supporting NFV requirements to enable SDN controller to run over almost any on-shelf machine, using virtualization technology [14]. Figure 5 provides the custom topology with 7-switchs, 7-hosts and 3-controllers.

However, Mininet, Miniedit and Cbench are complying the required specifications of NFV, providing the capability of constructing an emulated network with the capacity that may exceeds the controller ability of handling connections.

In this experiment we use 7 OpenFlow enabled switches, with 100Mbit bandwidth link and congestion window up to 32Mbyte in all topologies, each switch connected with a single host, the same topologies used for all experiments to compare controllers in identical environment.

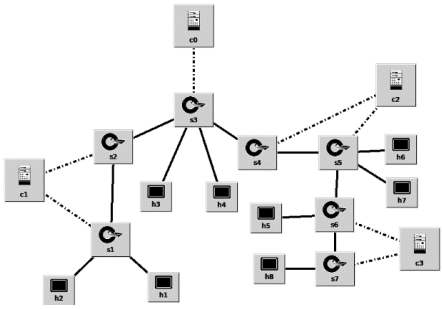


**Figure 5 Custom Topology**

## 3.2 Experiments Setup

The test bed consists of one machine supplied with Intel core 2Duo CPU at 2.33GH for both ”4 threads” and 4 GB RAM. The machine is running windows 7 as host and Ubuntu 14.04 LTS (GNU/Linux x86 64) as guest OS in Oracle Virtual Box, 3GB of ram allocated for the VM.

## 3.3 Network Settings

In this project we adopt 3 topologies to experiment with: Linear, Tree and Custom topologies. The link bandwidth is 100Mbit/sec for all experiments with maximum window size (socket buffer) available 32 Mbytes (conceptually). The same number of switches and hosts (7switches and 1 hosts per switch) is used in all experiments. The following command used to emulate linear and tree topologies, respectively:

***($ mn --controlle=remote,ip=127.0.0.1,port=6633 –topo=linear,7 –switch=ovs)***

Those command instruct the application to seek for a remote SDN controller in the same subnet with port number configured to 6633 or 6653, almost all SDN controller initialized with same configuration. Minienit emulate the custom topology through a graphical interface, which facilitate the constructing of the desired topology with the co-operation of Mininet. Mininet performs several activities such as: Connecting to controller, creating switches, creating hosts, creating links, and Establishing network connectivity.

After Mininet conforms a successfully session establishment, we start 2 terminals (h1 and h7) to run iPerf for generating traffic between hosts, the following command executed in Mininet’s command-prompt to start (external terminals): ***($ xterm h1 h7).***

Generating traffic task achieved using iPerf application, iPerf is traffic generator and throughput measurement tool, running the following command in hosts terminal invokes iPerf test:

***(iperf -s -i 1 -t 100 -w 100M)***

***(iperf -c 10.0.0.7 -i 1 -t 100 -w 100M)***

The above command used to run one machine as a server (10.0.0.7) and the other as a client (10.0.0.1). The iPerf configured to run this test for 100 second (time interval is 1 sec) with window size configured to 32 Mbytes as maximum, and with ICMP messages as much as the link afford. Table (1) list the summary for iPerf configuration parameters.

**Table 1 iPerf parameters**

| Parameter | Description | Action | Value |
|---|---|---|---|
| s | Identify session as server | Identifying 7h to act as server, listing to requests from other hosts in topology. | No value passed. |
| i | Reporting intervals | Sets periodic time between intervals. | 1 |
| t | Time interval | Sending messages interval. | 100 |
| f | Output format | Select reporting format | M |
| w | Window size | Setting socket buffer size | 100M |
| c | Identify session as client | Setting server ip | 10.0.0.7 server’s IP |

The Performance Metrics to evaluate the system model are 1) **Throughput**: “payload over time”, measures the most extreme burst of transferring data (Mbytes/sec), time is a period specified by examiner (100 seconds). 2) **Bandwidth**: express the maximum bandwidth utilized by network (Mbit/sec) and 3) **Packet Lost**: can be measured according to received packets with regard to sent packets, packet loss can be expressed as a percentage

## 4 RESULTS AND DISCUSSION

In this section we represent the experimental efforts to assess SDN networking experience, as an implementation and practice within testing and evaluating framework. We provide the Average Throughput, Average Bandwidth, Packet loss, Latency and prediction inspection for network with three different SDN controllers (FloodLight, POX and NOX), based on linear, tree and custom topology, respectively.

## 4.1 Average TCP/UDP Throughput

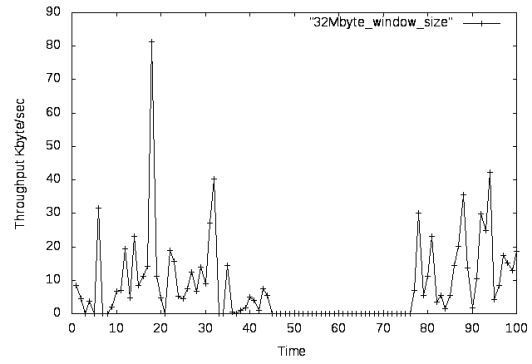
The SDN controllers used in this experiment is FloodLight, POX and NOX, controller runs for optimum performance. To invoke controller’s software in each experiment we execute the following command:

**$ sudo java -jar target/floodlight.jar**

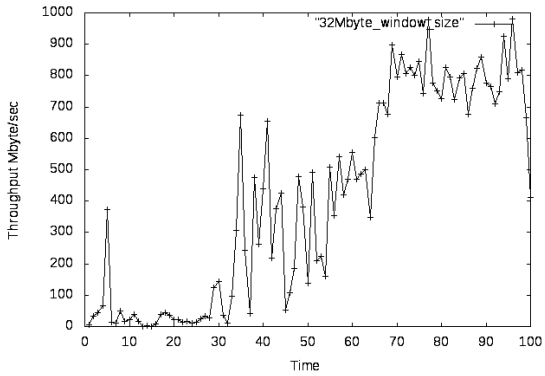
**$ sudo python pox.py forwarding.l2_pairs**

## A) Linear Topology

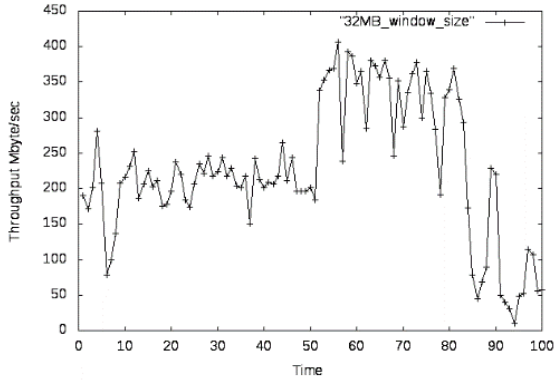
Figures 1, 2 and 3, plotted the achieved bandwidth in experiment for measuring network performance, iPerf generate a detailed report, which can be used to interpret the observations, in this experiment, we observe the emulated network with socket buffer sizes 32Mbyte and bandwidth 1000Mbit, the average statistic used as measure for evaluation.




(a)




(b)




(c)

**Figure 6 Throughput - Linear Topology - (a) FloodLight (b) POX (c) NOX**

The following table contains average values for three experiments with different SDN controllers, Floodlight, POX and NOX.

**Table 2 Average Throughput for network with three different SDN controllers.**

| Topology | Average | | |
|---|---|---|---|
| | Floodlight | POX | NOX |
| Linear | 8.0493 | 405.9723 | 227.176 |

According to the average values taken from three experiments the network shows best performance with POX controller. The following figure (7) used for best visualizing the comparison results between Floodlight, POX and NOX controllers. From the results we can observe cleary that the POX controller achive higher throughput with increasing the time.

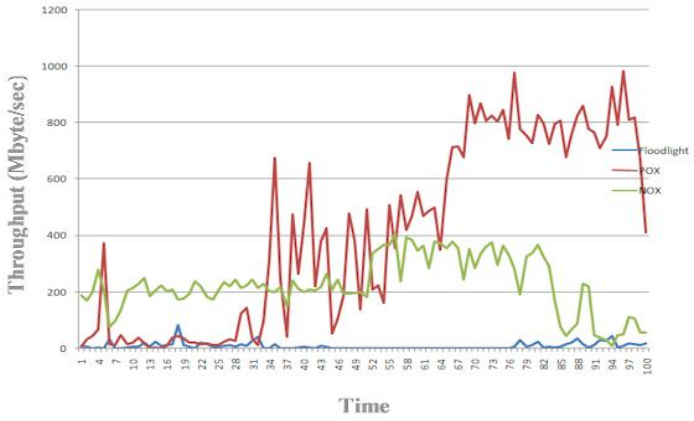



**Figure 7 Throughput - Linear Topology - Floodlight, POX and NOX**

## B) Tree Topology

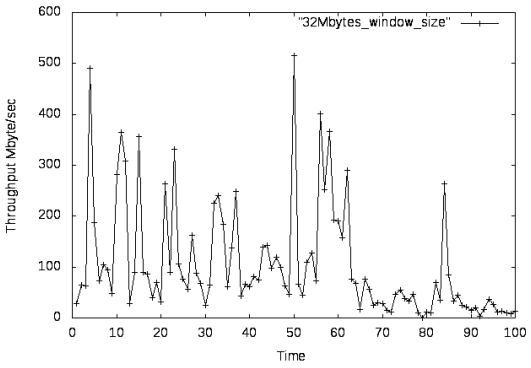
Mininet and iPerf used to emulate network and generate traffic, as previous experiment:

**$ mn --controlle=remote,ip=127.0.0.1,port=6633 –topo=tree,depth=3,fanout=2 –switch=ovs**

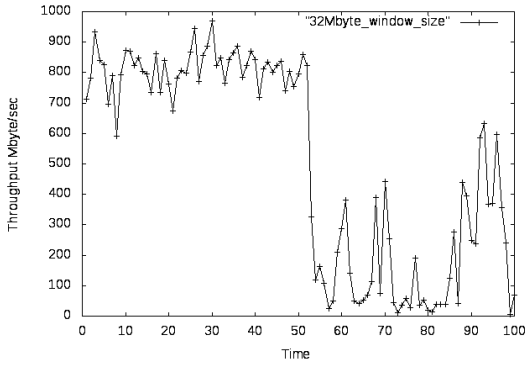
**- iperf -s -i 1 -t 100 -w 100M**

**- iperf -c 10.0.0.7 -i 1 -t 100 -w 100M s**

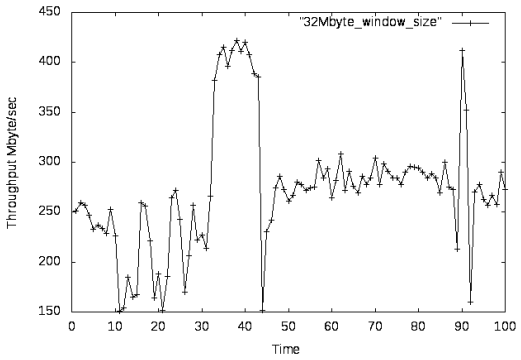
Table (3) contains average values for three experiments with different SDN controllers, Floodlight, POX and NOX.




(a)




(b)




(c)

**Figure 8 Throughput -Tree Topology – (a) Floodlight (b) POX (c) NOX**

**Table 3 average values for three experiments with different controllers, Floodlight, POX and NOX**

| Topology | Average | | |
|---|---|---|---|
| | Floodlight | POX | NOX |
| Tree | 105.5462 | 511.098 | 272.64 |

According to the average values taken from three experiments the network shows best performance with POX controller. The following figure (9) used for best visualizing the comparison result. Cleary observed that the POX controller provides higher throughput from startup to 48 time.

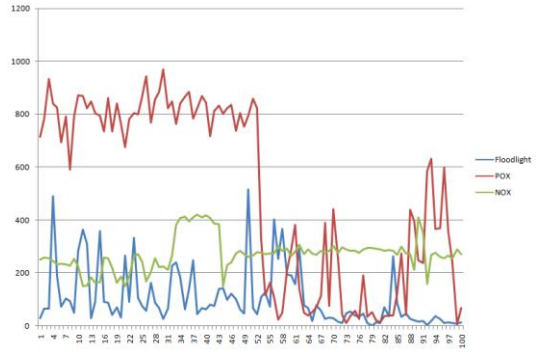



**Figure 9 Throughput - Tree Topology - Floodlight, POX and NOX**

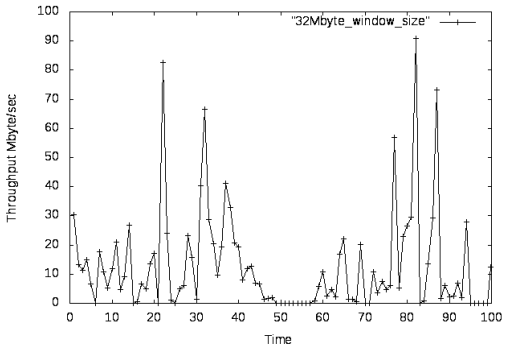
### C) Custom Topology

Miniedit and iPerf used to emulate network and generate traffic, as previous experiment. The pervious command used to invoke Miniedit graphical environment. Figure (10) describe the throughput results of the three controllers.

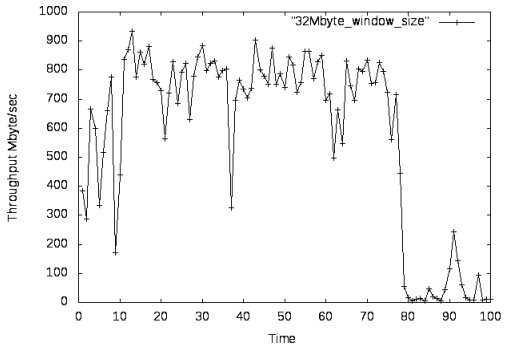
**$ sudo python expamples/miniedit.py**

**- iperf -s -i 1 -t 100 -w 100M**

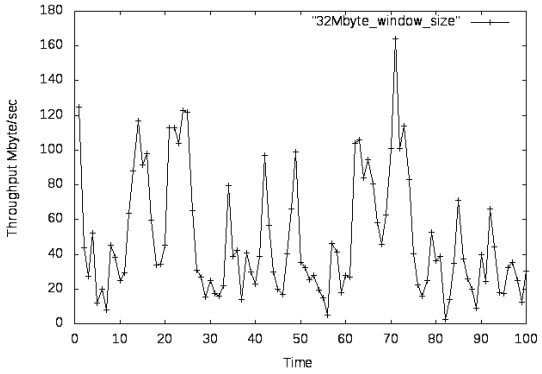
**- iperf -c 10.0.0.7 -i 1 -t 100 -w 100M**




(a)




(b)




(c)

**Figure 10 Throughput -Custom Topology – (a) Floodlight (b) POX (c) NOX**

The following table (4) contains average values for three experiments with different SDN controllers, Floodlight, POX and NOX.

**Table 4 Average values for three experiments with different controllers, Floodlight, POX and NOX**

| Topology | Average | | |
|---|---|---|---|
| | Floodlight | POX | NOX |
| Custom | 12.6635 | 574.9511 | 48.9107 |

According to the average values taken from three experiments the network shows best performance with POX controller. The following figure (11) used for best visualizing the comparison result:

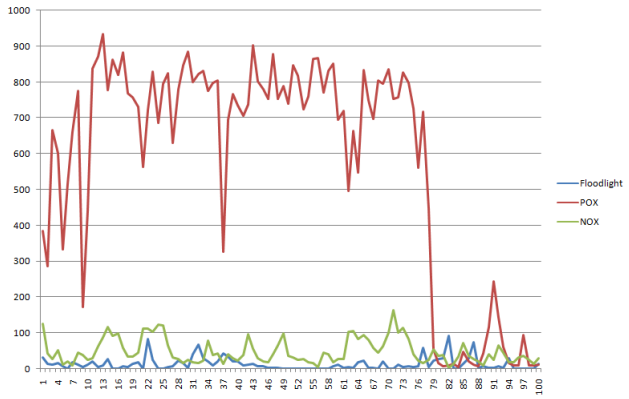
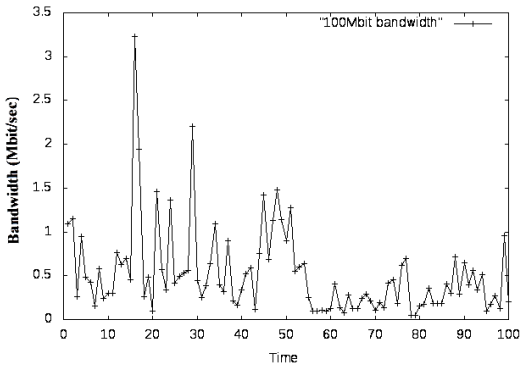



**Figure 11 Throughput - Custom Topology - Floodlight, POX and NOX**

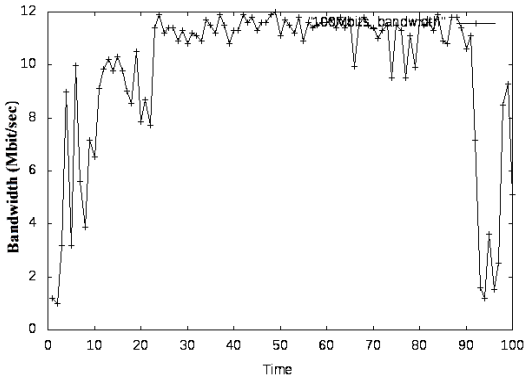
## 4.2 Average Bandwidth

### A) Linear Topology

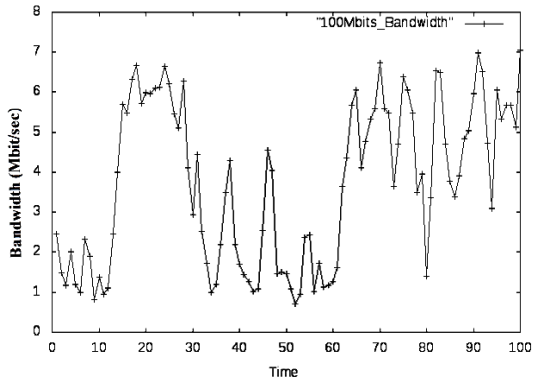
The SDN controllers used in this experiment is FloodLight, POX and NOX, controller runs for optimum performance. Mininet configured the network as the same as the previous experiment. The result came up with the following plotted in figure (12).




(a)




(b)




(c)

**Figure 12 Bandwidth -Linear Topology – (a) Floodlight (b) POX (c) NOX**

Table (6) contains average values for three experiments with different SDN controllers, Floodlight, POX and NOX.

**Table 5 Average values for three experiments with different SDN controllers, Floodlight, POX and NOX**

| Topology | Average | | |
|---|---|---|---|
| | Floodlight | POX | NOX |
| Linear | 0.393514 | 10.18568 | 3.260811 |

According to the average values taken from three experiments the network shows best performance with POX controller. The following figure (13) used for best visualizing the comparison result.

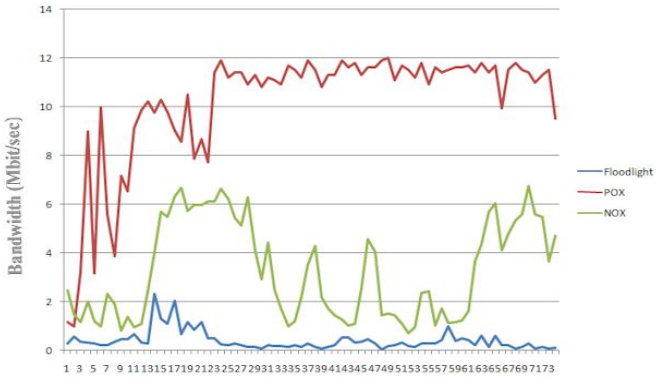
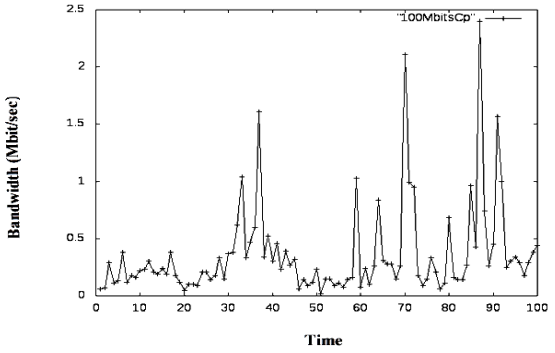


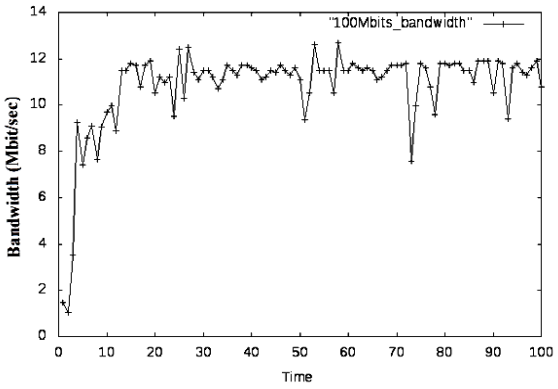

**Figure 13 Bandwidth - Linear Topology - Floodlight, POX and NOX**

### B) Tree Topology

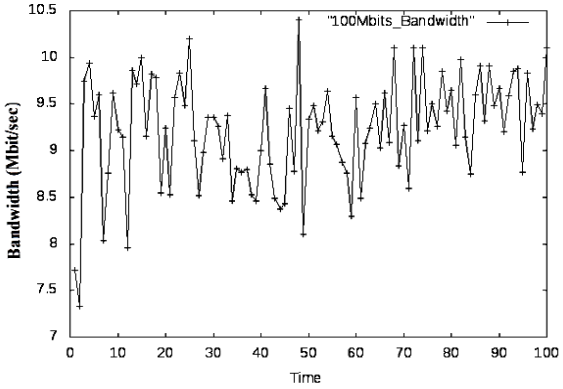
The result came up with the following plotted diagrams figurer (14).




(a)




(b)




(c)

**Figure 14 Bandwidth -Tree Topology – (a) Floodlight (b) POX (c) NOX**

The following table contains average values for three experiments with different SDN controllers, Floodlight, POX and NOX.

**Table 6 average values for three experiments with different controllers, Floodlight, POX and NOX**

| Topology | Average | | |
|---|---|---|---|
| | Floodlight | POX | NOX |
| Tree | 0.3579 | 10.8292 | 9.2242 |

According the average values taken from three experiments the network shows best performance with POX controller. Figure following diagram used for best visualizing the comparison results. Clearly observe that the POX and NOX are achieve higher bandwidth.

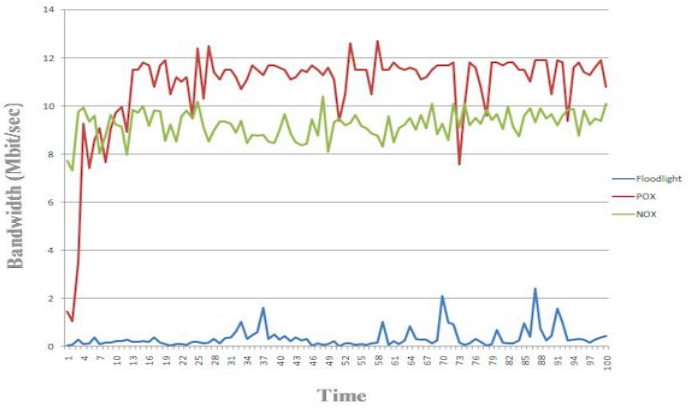
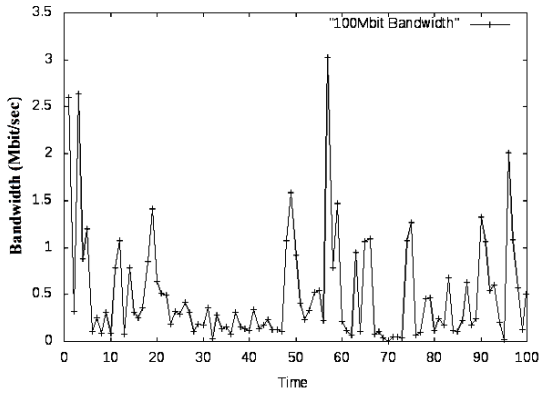



**Figure 15 Bandwidth - Tree Topology - Floodlight, POX and NOX**

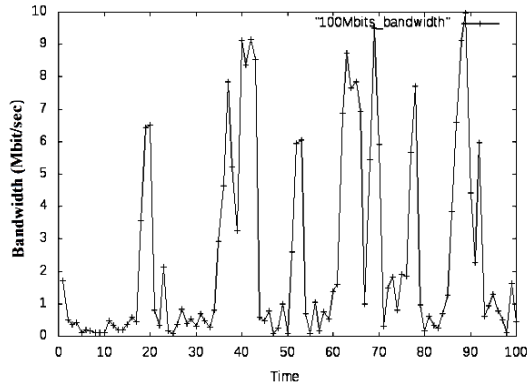
### C) Custom Topology

Miniedit used to configure the network as the same as the previous custom topology experiment for

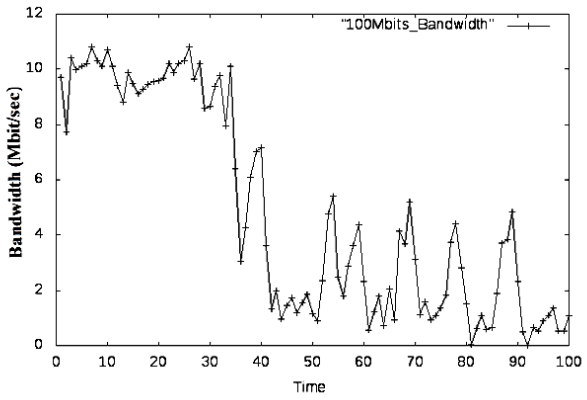
throughput. The result came up with the following plotted figure (16).




(a)




(b)




(c)

**Figure 16 Bandwidth - Custom Topology – (a) Floodlight (b) POX (c) NOX**

The following table (7) contains average values for three experiments with different SDN controllers, Floodlight, POX and NOX.

**Table 7 average values for three experiments with different controllers, Floodlight, POX and NOX**

| Topology | Average | | |
|---|---|---|---|
| | Floodlight | POX | NOX |
| Custom | 0.5025 | 2.482 | 4.8245 |

According the average values taken from three experiments the network shows best performance with NOX controller. The following figure (17) used for best visualizing the comparison results.

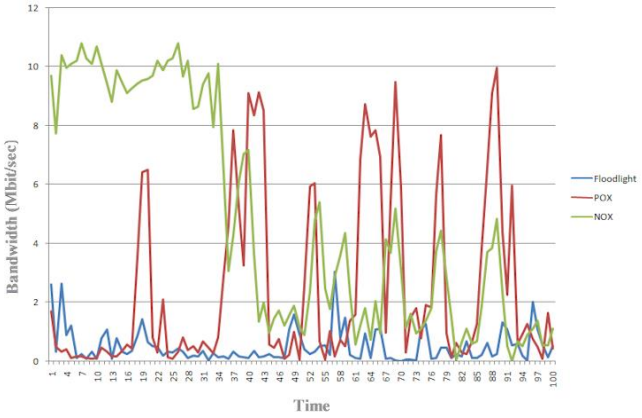



**Figure 17 Bandwidth - Custom Topology - Floodlight, POX and NOX**

- After sending the first quota of packets, the sender receives the acknowledgment, then progressively the number of packets can be send will increased (throughput), and that continued with each acknowledgment received, till congestion occurred, then the number of packet will be decreased to reach the amount of packets that can be accommodated by switch, this interpreting the zigzag pattern of the shape of the plot.
- The collapse shown above denotes the “Incast Problem” which occurred due to the burst of packets generated using iPerf (128KB for TCP) and the limitation of switch memory in order to buffer packets (congestion) [15], which affects the throughput dramatically.
- SDN controller handles congestion in switches throughout setting action table, this process is an original task for controller to be achieved, via application layer solutions through implementing efficient algorithms to handle congestion (which out of the scope of the research), however the controller capabilities of handling such situation will impact the throughput generally [16], One reason for

Floodlight low performance is congestion problem. The second observed reason for Floodlight controller performance is high rate of messages exchange, at least in the test's frame of time (100 seconds).

- Floodlight controller in linear topology experiment with window size 32 Mbyte shows lower throughput, unlike POX and NOX, we investigate Floodlight with different window sizes in the same topology. Figure (18) present the throughput comparison with 2, 20 and 32 Mbyte window size.

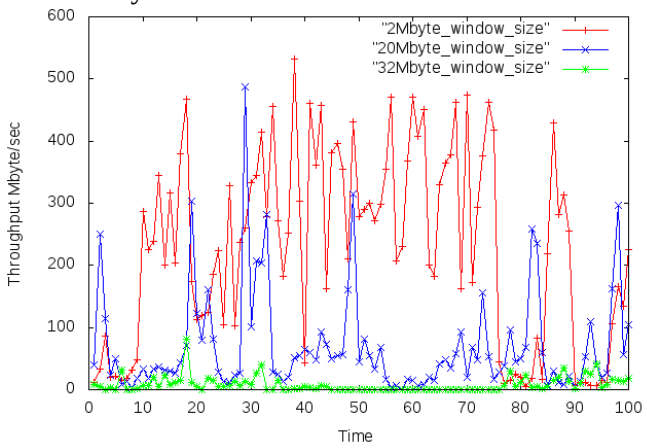



**Figure 18 Throughput with different window sizes 2, 20 and 32 Mbyte - Floodlight**

Floodlight show better performance with 2 Mbyte window sizes, which reveal Floodlight incapability of handling large window size.

**Table 8 Throughput Averages - 2, 20 and 32 Mbyte window size - Floodlight**

| Topology | Floodlight | | |
|---|---|---|---|
| | 2 Mbyte | 20 Mbyte | 32 Mbyte |
| Linear | 226.5677 | 70.0713 | 8.0493 |

- One cause of heavy traffic in networks is the discovery messages that broadcasting from controller to manipulate network by discovering connected device (switches).
- The SDN controller interfere in the process of transmitting data between end nodes, by setting the path for the data, through the modification of switch's flow table in case of table miss, this process required mechanisms to discover the topology in the first place, packet_in and packet_out messages exchanged between controller and switches in network to identify hosts, all examined SDN controller implement this message-exchanged topology discovery method[17], the process initiated by the switch when a packet with unknown destination received in his ingress port, the switch sends the packet (or part of it) to the controller to decide on it, same method of propagating LLDP messages by controller to maintain topology awareness and controller-switch channel establishment described in [18], the negotiation over the network consumes available bandwidth (to some extent) according to controller's specifications and behavior, such as time interval for sending discovery messages.
- Floodlight controller unable to handle congestion efficiency, no other factors in this controlled experimental environment can affect the performance of the network, in contrast with other investigated controllers.
- The high values of the plot describe the controller's max throughput can be achieved, on the other hand the low values describe the minimum throughput according the switch available memory.
- Standard deviation, show that the throughput did not affect by course of time.

## 4.3 Packet lost

SDN controllers investigated in this test are Floodlight, POX and NOX, as previous TCP/UDP test different topology constructed for testing the controllers, linear, tree and custom, Mininet and Miniedit used for constructing topologies, iperf used to generate traffic between end nodes.

One node configured as a server (10.00.0.7) and the other as a client (10.0.0.1), the dropped or lost packet observed according to the detailed report generated by iperf are listed in table (9).

Figure (19), figure (20) and figure (21) are describe the packet loss in three topologies. From the figures we can observe that the FloodLight has higher packet loss in linear and tree topology and NOX controller has lower packet loss in case of custom topology.

Congestion (as previous experiments) is the cause of dropping messages, due to the full memory state of the switch [8], Packets during their travelling through switches get lost or delayed, retransmission enforced when a packet timeout occurred. Protocols used to insure the transmission of packets such as TCP, in contrast UDP protocol do not maintain such assurance, which led to poor VoIP or video stream Floodlight in linear

topology with 32 Mb window size and 100 Mbits/sec bandwidths, shows highest dropped message among all controllers.

**Table 9 Received, send and lost packet**

| Controller | Topology | Received Packets | Send Packets | Lost percentage |
|---|---|---|---|---|
| **Floodlight** | Linear | 1397 | 4990 | 72 |
| | Tree | 3992 | 4990 | 20 |
| | Custom | 3470 | 4990 | 30.4409 |
| **POX** | Linear | 4984 | 4990 | 0.12024 |
| | Tree | 4990 | 4990 | 0 |
| | Custom | 3592 | 4990 | 28 |
| **NOX** | Linear | 4982 | 4990 | 0.16 |
| | Tree | 4990 | 4990 | 0 |
| | Custom | 4987 | 4990 | 0.05 |

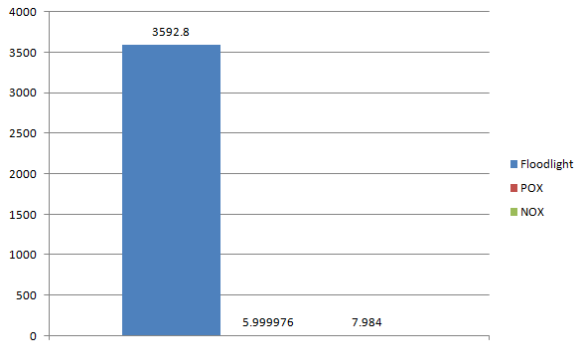



**Figure 19 Packet Lost - Linear Topology**

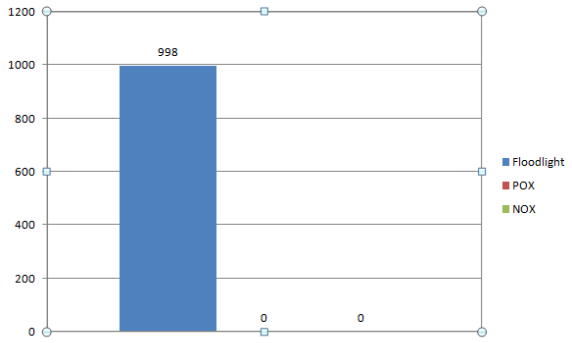



**Figure 20 Packet Lost - Tree Topology**

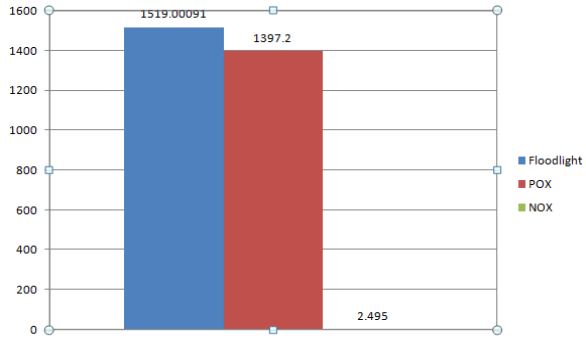



**Figure 21 Packet Lost - Custom Topology**

## 4.4 Latency

Latency estimated using high speed or burst of sending messages to a controller, the ability of handling received message is keystone for better performance, cbench emulate a fake network for testing purposes and generating an adequate report (at least form IT profession point of view). Packet_in and Packet_out is an openflow messages used to regulate and control the process of transferring data (in form of chopped messages "packets"), latency used some time to protect network form congestion or over-charge that may happened in high traffic networks, investigating controllers against latency or the delay may have happened in a controller is the goal of this test. The following table (11) shows the average responses per second, also the image describes the same values, higher value represents better response to OFPT_PACKET_IN sends to controller.

**Table 10 Average Response/sec.**

| Controller | Average Responses/sec |
|---|---|
| Floodlight | 1799.81 |
| POX | 5353.92 |
| NOX | 2223.83 |

Latency also denoted as response over time, Cbench used to measure latency in this experiment, the mechanism used by Cbench is generating traffic by sending messages from all switches connected in network to controller aggressively, which will cause congestion, and therefore will affect control channel with latency[8]. Figure (22) show that the POX controller has higher latency.

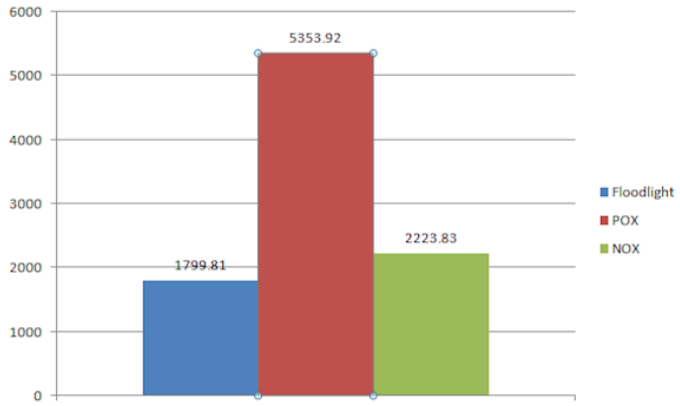



**Figure 22 Latency (response/sec)**

## 4.5 Topology Discovery Time

Mininet used for constructing emulated network. Iperf (as in other experiments) used as traffic generator, sending message (ICMP messages as matter of fact), the route for those message is unknown to controller, so to solve this problem, a switch used to broadcast a discovery message, this message received and handled by controller to a locate the desired route for message(s), the addressed node detected and switch sends a packet_out to controller. Table (12) describes the OpenFlow messages used for this purpose (discovery).

Figure 23 summarize discovery time for controllers, which can be obtained by calculating the difference between first and last discovery message exchanged between switch and controller.

The role played by Wireshark application is analyzing and capturing traffic on ports (test-bed ports), timestamp and message type used to identify required time for controller to discover the topology or addressed node. The number of packets handling discovery process depends on topology and active channels with switches [19].

**Table 11 Wireshark's captured OpenFlow packets – Floodlight, POX and NOX**

| # | Timestamp | Source | Destination | Protocol | Type | Packet Type | Topology | Controller | D. Time |
|---|---|---|---|---|---|---|---|---|---|
| 16018 | 91.642 | 5a:2d:36:8f:bc:e7 | Broadcast | OpenFlow | 128 | Type: OFPT_PACKET_IN | linear | floodlight | 0.008 |
| 16032 | 91.65 | localhost | localhost | OpenFlow | 92 | Type: OFPT_PACKET_OUT | | | |
| 39352 | 288.15 | 92:6e:db:bf:39:38 | Broadcast | OpenFlow | 128 | Type: OFPT_PACKET_IN | tree | floodlight | 0.096 |
| 39417 | 288.246 | localhost | localhost | OpenFlow | 172 | Type: OFPT_PACKET_OUT | | | |
| 262729 | 2314.993 | d6:7d:d9:17:56:22 | Broadcast | OpenFlow | 128 | Type: OFPT_PACKET_IN | custom | floodlight | 0.032 |
| 262770 | 2315.025 | localhost | localhost | OpenFlow | 172 | Type: OFPT_PACKET_OUT | | | |
| 2487 | 60.823 | 5a:23:95:2c:bc:a0 | Broadcast | OpenFlow | 128 | Type: OFPT_PACKET_IN | linear | pox | 0.077 |
| 2560 | 60.9 | localhost | localhost | OpenFlow | 172 | Type: OFPT_PACKET_OUT | | | |
| 21761 | 124.143 | b6:c6:cd:99:1d:13 | Broadcast | OpenFlow | 128 | Type: OFPT_PACKET_IN | tree | pox | 0.164 |
| 21846 | 124.307 | localhost | localhost | OpenFlow | 92 | Type: OFPT_PACKET_OUT | | | |
| 81292 | 761.396 | d2:46:35:26:ba:b6 | Broadcast | OpenFlow | 128 | Type: OFPT_PACKET_IN | custom | pox | 0.085 |
| 81361 | 761.481 | localhost | localhost | OpenFlow | 172 | Type: OFPT_PACKET_OUT | | | |

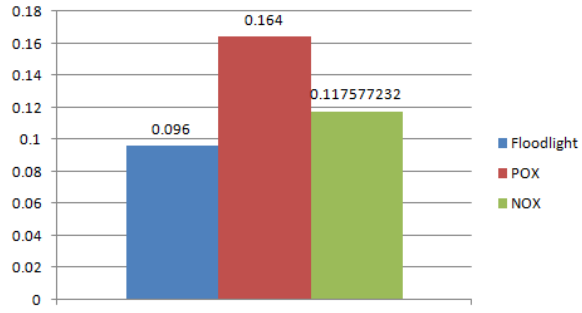
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 16796 | 53.10245 | 9e:50:ba:48:2b:b9 | Broadcast | OpenFlow | 128 | Type: OFPT_PACKET_IN | linear | nox | 0.098151 |
| 16864 | 53.2006 | 127.0.0.1 | 127.0.0.1 | OpenFlow | 172 | Type: OFPT_PACKET_OUT | | | |
| 37654 | 279.8228 | 96:e6:3a:9e:2c:ae | Broadcast | OpenFlow | 128 | Type: OFPT_PACKET_IN | tree | nox | 0.117577 |
| 37706 | 279.9404 | 127.0.0.1 | 127.0.0.1 | OpenFlow | 92 | Type: OFPT_PACKET_OUT | | | |
| 36996 | 241.0883 | 66:20:94:9c:57:b0 | Broadcast | OpenFlow | 128 | Type: OFPT_PACKET_IN | custom | nox | 0.141344 |
| 37069 | 241.2297 | 127.0.0.1 | 127.0.0.1 | OpenFlow | 172 | Type: OFPT_PACKET_OUT | | | |

## 4.6 Prediction Inspection

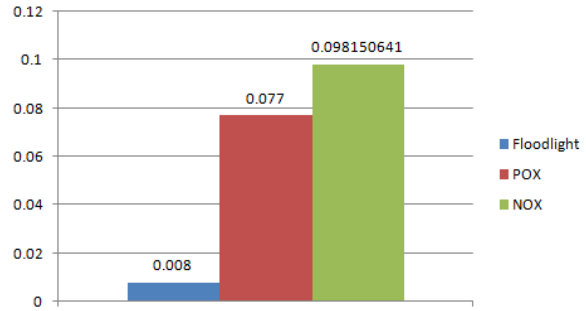
We use cbench benchmarking tool to collect data about throughput with different number of switches, which serve the mean of scalability. Cbench calculate average, standard deviation and other statistics to estimate the behavior of a controller, we use POX controller as an experimental subject. Test achieved using 8, 16, 24, 32, 40, 48 and 56 switches, the last value "56" used to test the correctness of line equation for predicting. The idea behind prediction is formulating an equation for each controller, which helps network administrator to estimate throughput and latency for future network scalability. Important point to recognize, the number of switches should not exceed the maximum number have served by SDN controller.

Table (13) contains data about throughput and latency, for 8, 16, 24, 32, 40, 48 and 56 switches.

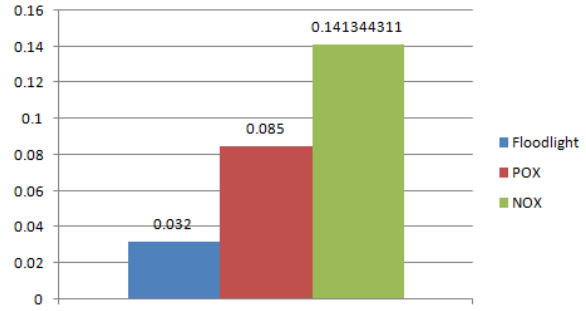
| Switches | Throughput - POX | | Latency - POX | |
|---|---|---|---|---|
| | average | stdev | average | stdev |
| 8 | 6278.04 | 328.35 | 5429.05 | 253.27 |
| 16 | 6069.85 | 720.75 | 5129.19 | 449.45 |
| 24 | 6152.86 | 540.59 | 5736.91 | 491.13 |
| 32 | 5726.17 | 832.30 | 5801.51 | 319.09 |
| 40 | 5774.27 | 1199.81 | 5918.37 | 19.84 |
| 48 | 5612.20 | 1205.37 | 5693.57 | 791.98 |
| 56 | 5586.11 | 1278.21 | 5531.57 | 1046.09 |




(a)




(b)




(c)

**Figure 23 Discovery Time – (a) Linear Topology (b) Tree Topology (c) Custom Topology**

**Table 12 Throughput and Latency of POX controller**

Figure (24) and figure (25) are represents trend line constructed according to the result output, for both throughput and latency for POX controller. The line equation for throughput and latency easily can be calculated using Excel, by choosing "Trend" for plotted scattered diagram. In this experiment, figure (24) shows the throughput of POX controller with different number of switches. The equation of best-line $y = (-16.58x + 6399)$, the negative value of the x parameter (beta) describes the negative relationship between throughput and the number of switches, controller's throughput decreased with each added switches.

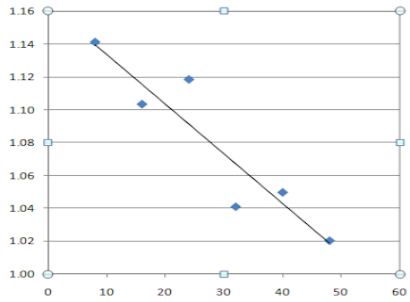



**Figure 24 Throughput vs. Switches – POX**

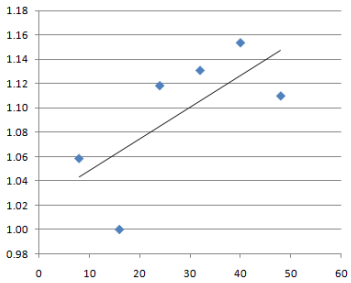



**Figure 25 Latency - POX**

By substituting the x value equal to 56 switchs, we get a good approximation for throughput equal to 5470.52, which is not far from cbench result equal to 5586.11 flows per ms as an average. Besides the computed correlation (-0.92876) represent very strong relationship between throughput and the number of switches. Unlike throughput in figure (25), latency test failed to predict behavior when tested with 56 switch, the result not close enough as preceding throughput result, we came with value equal to 6262.28 response per ms as an average, according to (y = 13.41x + 5242) line equation. The relation between latency and number of equation is positive and strong according to the computed correlation 0.693812977.

## 5 CONCLUSION AND RECOMMENDATION

In this experimental research, we tested several SDN Controllers such as Floodlight POX, NOX, experiments performed under different topologies linear, tree and custom. Applications used for benchmarking controller configured with different parameters such as bandwidth, socket buffer and different number of switches. Several tests for different performance measures considered, throughput, latency, packet lost, topology discovery time and UDP bandwidth utilization. We attempt to predict the behavior of controllers through regression or best line fit as a statistic for sample results drown from reports generated by benchmarking applications and analyzing tools.

The most important discovery in controllers evaluating revealed that the technology of SDN suffering from the lack of an adequate interface. Robustness also a great concern (Floodlight is an exception), which tends to fail during operating and conflicts with some operating system issues such as port allocating and de-allocating. The congestion forms the major restriction in the flow of packets across the network which requires more efficient algorithms to overcome this limitation. Once again, the aim of this research is to represent the strength of controllers rather than failing them. Studying SDN controller in a real hardware separated from emulated network will provide pest judgment, also running benchmarking tool in a separate machine increases efficiency of tests by overcome virtualization limitation and complications such as memory usage and port allocation. Designing test application for specific purposes such as network discovery will facilitate gaining more control in tests performance. Mimic realistic network configuration in an emulation application will provide close estimation for network performance. Designing real world network traffic such as UDP applications (audio and video streaming) for investigating capabilities of network and controller can provide results close to reality.

## REFERENCES

[1] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, "Software-defined networking (SDN): Layers and architecture terminology," in *RFC 7426*: IRTF, 2015.

[2] A. H. M. Hassan, A. M. Alhassan, and F. Izzeldean, "Performance Evaluation of SDN Controllers in Ofnet Emulation Environment," in *2019 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, 2019, pp. 1-6: IEEE.

[3] K. Kaur, J. Singh, and N. S. Ghumman, "Mininet as Software Defined Networking Testing Platform," presented at the International Conference on Communication, Computing & Systems (ICCCS–2014, 2014.

[4] C. Fernandez and J. Muñoz, "Software Defined Networking (SDN) with OpenFlow 1.3," *Open vSwitch and Ryu,(June 2010)*, vol. 183, 2016.

[5] (2/10/2019). *Wireshark. Available: https://www.wireshark.org*.

[6] V. Bhuvaneswaran, A. Basil, M. Tassinari, V. Manral, and S. Banks, "Benchmarking Methodology for Software-Defined Networking (SDN) Controller Performance," *Internet Engineering Task Force, Tech. Rep. RFC-8456*, 2018.

[7] Z. K. Khattak, M. Awais, and A. Iqbal, "Performance evaluation of OpenDaylight SDN controller," in *2014 20th IEEE international conference on parallel and distributed systems (ICPADS)*, 2014, pp. 671-676: IEEE.

[8] R. Kandoi and M. Antikainen, "Denial-of-service attacks in OpenFlow SDN networks," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015, pp. 1322-1326: IEEE.

[9] A. Tirumala, T. Dunigan, and L. Cottrell, "Measuring end-to-end bandwidth with Iperf using Web100," in *Presented at*, 2003, no. SLAC-PUB-9733.

[10] A. H. Eljack, A. H. M. Hassan, and H. H. Elamin, "Performance Analysis of ONOS and Floodlight SDN Controllers based on TCP and UDP Traffic," in *2019 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, 2019, pp. 1-6: IEEE.

[11] B. Xiong, K. Yang, J. Zhao, W. Li, and K. Li, "Performance evaluation of OpenFlow-based software-defined networks based on queueing model," *Computer Networks*, vol. 102, pp. 172-185, 2016.

[12] W. Braun and M. Menth, "Software-defined networking using OpenFlow: Protocols, applications and architectural design choices," *Future Internet*, vol. 6, no. 2, pp. 302-336, 2014.

[13] K. Naik and P. Tripathy, *Software testing and quality assurance: theory and practice*. John Wiley & Sons, 2011.

[14] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-art and Research Challenges," *IEEE Communications Surveys and Tutorials*, p. 9, 2015.

[15] A. Phanishayee *et al.*, "Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems," in *FAST*, 2008, vol. 8, pp. 1-14.

[16] A. Tirumala, L. Cottrell, and T. Dunigan, "Measuring end-to-end bandwidth with Iperf using Web100," p. 2.

[17] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska, "Efficient Topology Discovery in OpenFlow-based Software Defined Networks," ed, 2015.

[18] "OpenFlow Controller Benchmarking Methodologies," November 2016 ed: Open Networking Foundation, 2016, p. 11.

[19] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska, "Efficient topology discovery in OpenFlow-based software defined networks," *Computer Communications*, vol. 77, pp. 52-61, 2016.