POX Controller and Open Flow Performance Evaluation in Software Defined Networks (SDN) Using Mininet Emulator

Haeeder Munther Noman¹, Mahdi Nsaif Jasim²
¹Software department, College of Information Technology, University of Babylon, Babylon, Iraq
²University of Information Technology and Communications, Baghdad, Iraq
Email: hydermu79@gmail.com

Abstract. Growing popularity of Internet demands for the agility as well as the flexibility of computer networks. Traditional networking system is unable to satisfy recent computing needs. Proprietary devices configured manually create an error-prone situation in addition they are incapable to fully utilize the capability of physical network infrastructure. This has resulted in the paradigm shift in the networking industry and it is known as Software defined networking. Advantages such as programmability, task virtualization and easy management of the network can be provided by employing SDN platform, on the other hand POX is defined as a Python based open source OpenFlow SDN Controller mainly used for faster development and prototyping of new network applications basically comes pre-installed with the mininet virtual machine. POX controller can turn dumb OpenFlow devices into hub, switch, load balancer and firewall devices. In this paper Performance metrics such as service Delay, utilized bandwidth, received packets and bytes were measured and recorded using network monitoring tools like iperf and D-ITG in order to analyze the functionality of the POX controller as well as to evaluate the operation performance of POX controller for SDN environment. The results of this research were the recommendation of using POX controller in for rapid development and prototyping of network control systems as well as being the framework for the interaction with Open Flow switches.

1. Introduction
The SDN controllers have been commonly classified according to the used programing language, and innovation purposes for the proposed research, in which the POX controller is an example of this environment. The northbound and southbound Application Programming Interfaces (APIs) conducts the communication between the upper and lower layers and POX controller respectively. The operating applications inside the SDN network and the controller can be communicated by using the Northbound APIs meanwhile the communications between the packet forwarding hardware and the adopted controller can be carried out by using the Southbound APIs. The APIs and software applications can be potentially developed with the use of API that gives the network the capability to be directly programmable and implementable[1] In SDN environment, the communication that takes place between the control layer and forwarding layer is named as the southbound communication. OpenFlow protocol can be defined as the standard mechanism that permits the control layer and forwarding layer to communicate with each other. Moreover, POX controller can have a direct access and manipulation capability to the forwarding devices of data plane in the presence of the OpenFlow protocol. SDN applications can be developed using POX open source controller[2]. The OpenFlow protocol that is a communication protocol between the switches and controllers can be implemented
effectively when the POX controller is used. Various software applications such as load balancer, switch, hub, and firewall can be run by using the POX controller. In this study, the OpenFlow version 1.0 was used because it can be supported by the POX controller as this was clearly reported in the previous works[3]. The separation of hardware installation or software configuration is required by each measurement technique in the conventional network. This could make the implementation technique as a complex task. On the other side, the OpenFlow networks are able to provide the needed interfaces in order to execute the majority of the common measuring methods economically and efficiently[4]. The measurements of the SDN Network are mostly performed on the application layer or network layer. In fact[5], the application layer measurements are designed to evaluate the performance of the utilized application. The measurements of the Network layer are designed to use infrastructure forwarding elements such as switches and routers[6]. This study aimed to analyze and study the OpenFlow protocol and the POX controller as well as the recognition of the exchanged messages between the POX controllers and the devices located in the network, as long as this protocol regarded as open source and POX controller could be programmed to implement the desired application in order to facilitate carrying possible improvements in order to increase utilization efficiency of POX controller in network world.

2. **Software Defined Network (SDN)**
SDN provides a new networking model that enables a better agility, a central management, open-standards architecture and the flexible support according to diverse requirements, by separating the controlling power from forwarding devices and establishing logically centralized control plane, named controller[7]. An SDN Controller is the brain of the SDN network, transmitting the required information to the specified switches/routers through southbound application programming interfaces (APIs) as well as the applications and business logic through northbound APIs. Different organizations are developing different well-defined APIs, which promises to dramatically improve the programmability, scalability resource utilization, security and real-time execution of SDN. The Open Networking Foundation (ONF) organization is taking the lead in developing the dominant OpenFlow protocol for SDN[8]

3. **SDN OpenFlow model**
Software defined networking has three models including the SDN OpenFlow model, SDN over relay model, and hybrid model. Among these models, the SDN OpenFlow model was chosen to be investigated in this study. Practically, in addition to that the SDN OpenFlow model provides network programming ability from the centralized view through the modern and extensible API; it can also utilize the standardized protocol between the controller and agent on the network element for the state of instantiating. Beside and most importantly, it separates the control and data planes. Open Flow protocol that provides a great deal of flow/traffic control.[9] Reported that the Stanford University firstly implemented and imagined Open Flow model as part of their network

4. **Mininet**
Mininet [10], is used to emulate the network. Mininet is perhaps a network emulation orchestration system which runs router, SDN switches, different end-hosts, and links between all the devices on a Linux kernel system. It is used for emulating all the SDN system, the hosts in mininet behaves nothing but like a real machine in which you can run different arbitrary programs. The SSH can be also used into the Mininet using putty and bridge the network to the hosts. The single system could appear as a completed network, running the same kernel, system, and user code when the lightweight virtualization is adopted. Furthermore, the virtual software-defined networks can be established by using the Mininet. These networks are consisting of an OpenFlow controller, flat Ethernet network of multiple OpenFlow-enabled Ethernet switches, and multiple hosts that connected to the specified switches. Built-in functions that support using different types of controllers and switches could be part of the Mininet. The Mininet Python API could strongly help in creating complex custom scenarios. Compared to any other network emulator mininet is easy to use and open source. Due to its’ many inbuilt features, it makes it favorable for research work in the field of SDN. We don’t have any other
emulator with the capability of developing python programs to implement the topology. After a survey to a number of emulators it is found that mininet is the most suitable for SDN development environment. The Mininet could reasonably perform on a single laptop by leveraging Linux features which are the processes and virtual Ethernet pairs in network namespaces.

5. POX Controller
There are many types of controllers. Some of the popular OpenFlow controllers are NOX, POX, Beacon, Floodlight, Ryu and Open Daylight, etc. POX is very popular for prototyping and for its simple structures. Others are used for the production network. POX is defined as an OpenFlow controller that was derived and developed through the use of python programming language that basically aim to provide an efficient and easy environment for performing research investigations and tests in SDN networks. POX relies on component-based model in which the whole network elements as well as activities are recognized as separate components that are able to be isolated and utilized every time and place the need is. The location of POX is specifically between network components at one side and the applications on the other side. Moreover, it is responsible for achieving any type of communications between applications and devices. SDN controllers mainly rely on protocols such as OpenFlow protocol were network devices are configured, optimal network path for traffic applications are selected and servers are permitted to inform the switches where to direct the packets. In addition of being a framework for the interaction with the OpenFlow switches, POX can be utilized as the foundation for some of ongoing work to assist building of the emerging discipline of SDN were it can be applied in various fields such as distribution prototyping and exploration, SDN debugging, network utilization, controller design and programming models. POX modules are in fact extra python programs that can be used in case POX is initiated from the command line prompt, these modules carry out network functionality in SDN.

6. D-ITG (Distributed Internet Traffic Generator)
The D-ITG tool can be defined as a traffic generator that remains to the designs where the time of the packets’ inter departure and size could be recognized[11]. In addition to the probability distribution such as Poisson and Gamma that could offer plenty of help, a large number of well-known protocols like: TCP, UDP, ICMP, Telnet and VoIP could also be supported. In fact, a packet flow details such as the Throughput Measurement, Round-Trip-Time (RTT), Jitter, and Packet loss could be obtained by the users. The D-ITG has a Distributed multi-component architecture as shown in Figure.1

![Figure 1. D-ITG Software Architecture](image-url)
7. Simulation model
The process of evaluation includes the creation of SDN topology using Mininet emulator with a different number of hosts and switches. The remote POX controller was launched and the SDN topology with different parameters was run on the remote POX controller. Furthermore, performance tests including the iperf tool to measure the bandwidth utilization in network as well as the D-ITG tool to generate traffic were carried on in order to enable users to obtain details of packets flow like delay, dropped packets and received bytes.

8. Network Topology
Network topology adopted in this research was a linear topology which consisted of 8 switches and nodes; switches employed were OpenFlow-enabled switches which support OpenFlow protocol and expected to be connected to the POX controller. Moreover, nodes perform the role of host computers in addition the links without any delay were created between nodes and switches on one hand and between switches and the POX controller on the other hand. Advance analysis, investigations and simulations from the network topology mentioned above were carried out by the use of Miniedit emulator which resulted in the network topology shown in Figure 2.

Figure 2. Mininet Network Topology

9. Implementation and Results:
The performance of the POX controller was measured and reported using the Mininet emulator in this study. Fundamentally, being the default controller in Mininet, POX is essentially utilized when the controller is called for any purpose. The whole switches adopted here can operate with the OpenFlow protocol which supports switches easily and eventually allowing them to communicate with the controller effectively in term
of the flow. The evaluation of the POX controller performance was tested, certain parameters were considered for testing purposes. Such parameters are:

1- Utilized bandwidth
2- Allocated CPU load
3- Loss of packets
4- Latency and throughput

The utilized topology was a linear network topology; the size of switches raises exponentially i.e. 8, 10, 15, 20 and 30 switches.

10. Bandwidth Utilization

Bandwidth utilization is considered an important factor for the purpose of performance analysis. In order to carry out a test for the bandwidth utilization, a linear topology network that consists of 8 switches and hosts was used as shown in Figure 2. Moreover, the 5Mbps of bandwidth was assigned before it was utilized by the switches, and then measured by using the iperf tool. After that, the assigned bandwidth was improved gradually to exactly 20Mbps, 30 Mbps, 40Mbps, 50Mbps till 100 Mbps which then was escalated to an exponential level till 500 Mbps. The tested results of the simulation were listed in Table 1.

| N Switch | BW Utilization 10Mbps | BW Utilization 20Mbps | BW Utilization 30Mbps | BW Utilization 40Mbps | BW Utilization 50Mbps | BW Utilization 100Mbps | BW Utilization 200Mbps | BW Utilization 300Mbps | BW Utilization 400Mbps | BW Utilization 500Mbps |
|----------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|------------------------|------------------------|------------------------|------------------------|------------------------|
| 8        | 10                    | 20                    | 30                    | 40                    | 50                    | 100                    | 200                    | 300                    | 400                    | 500                    |
| 10       | 10                    | 20                    | 30                    | 40                    | 49.98                 | 100                    | 200                    | 300                    | 400                    | 500                    |
| 15       | 10                    | 20                    | 29.9                  | 35.6                  | 49.1                  | 99.9                   | 198                    | 293                    | 398                    | 490                    |
| 20       | 10                    | 20                    | 30                    | 39.9                  | 49.8                  | 96                     | 196                    | 279                    | 396                    | 490                    |
| 30       | 10                    | 20                    | 30                    | 39.9                  | 49.8                  | 85.8                   | 176                    | 256                    | 367                    | 477                    |

Results shown in Table 1. Were represented in Figure 3. In which it can be clearly concluded that for the t Bandwidth of 10, 20, 30 Mbps the utilized bandwidth is constant. For 40 and 50 Mbps, the utilization was Minimum from 15 till 30 switches. For the bandwidth that ranges from 100 to 500 Mbps a slight utilization was noticed from 15 till 20 switches. However, in the case of 300Mbps, there was enormous bandwidth utilization. The effective utilization is recorded from 20 till 30 switches, which resulted to the fact that effective bandwidth utilization occurs when the range of bandwidth starts from 100Mbps and rising up to 500 Mbps. This could also cause significant bandwidth losses.
Figure 3. The Utilized Bandwidth (Mbps) Versus Number of Switches

11. CPU Load Allocation
POX controller efficiency during operation was under test through this procedure in which a linear network topology consisting of 8 switches and hosts were under consideration as shown in Figure 2. Multiple unidirectional flows from many senders towards many receivers were generated using the D-ITG (distributed internet traffic generator). In the beginning 10% of the CPU was assigned to each host in the network topology from h1 to h7 then by considering h1 as the sending host and h7 as the receiving host, h1 send a stream of packets with the size of 1000pkts/s where size of each packet is 512 bytes to h7. Furthermore, D-ITG was utilized to monitor, measure and record the average delay, Bytes received as well as average Bit/Rate and average packet/rate. Regularly the CPU load is increased by 10 to 20%, 30%, 40% until it reaches 100% of CPU load under the same assumptions, results were listed in Table 2.

| CPU allocation | Av. Delay (ms) | Bytes received (Kbyte) | Av. Bit/rate (Kbit/s) | Av. Pkt/ rate (Pkt/s) |
|----------------|---------------|------------------------|----------------------|----------------------|
| 10%            | 0.080         | 3608.576               | 2890                 | 705                  |
| 20%            | 0.055         | 3698.176               | 2961                 | 722                  |
| 30%            | 0.082         | 3617.792               | 2898                 | 707                  |
| 40%            | 0.049         | 3768.320               | 3017                 | 736                  |
| 50%            | 0.060         | 3670.016               | 2937                 | 717                  |
| 60%            | 0.074         | 3601.408               | 2882                 | 703                  |
| 70%            | 0.055         | 3750.400               | 3003                 | 733                  |
| 80%            | 0.040         | 4845.632               | 3080                 | 751                  |
| 90%            | 0.061         | 3693.056               | 2956                 | 721                  |
Results demonstrated in Table 2. Were represented in Figure 4. (a) And (b), at the beginning the minimum recorded rate of the received bytes, Average packet/rate and Average bit/rate took place when the CPU load is 10%. However, as the CPU load gradually increase up to 20%,30%,40%,50%,60% till 70%, then there was a light fluctuation of the received Bytes, Average Packet/Rate and Average Bit/Rate whereas average delay fluctuated heavily , when the CPU allocation between was between 70-80% for the sender hosts lead to achieve the highest received bytes, Average Packet/Rate, Average Bit/Rate and the lowest delay. This could lead to accomplishing the best performance case. Therefore, when largest CPU space was assigned to the Completely Fair Scheduler, the maximum packets are attained.

12. Packet loss
The last test was conducted to examine the Packet loss which actually played an important role when related to the efficiency of packets delivered. D-ITG tool was utilized and the same network linear topology was employed with 8 switches and hosts. In this scenario hosts h1 and h2 were considered as sender hosts meanwhile the host h8 was pointed out as the receiver, and h4 was made as the remote log host, as shown in Figure 9. Besides, the flows with a packet size that is varying from 64 to 2048 bytes were sent. Results were listed in Table 3.
### Table 3. Packet Losses According to the Packet Sizes

| Packet size (bytes) | Dropped packets (%) |
|---------------------|---------------------|
| 64                  | 3 (0.02%)           |
| 128                 | 17 (0.12%)          |
| 192                 | 9 (0.06%)           |
| 256                 | 6 (0.04%)           |
| 384                 | 11 (0.08%)          |
| 512                 | 2 (0.01%)           |
| 768                 | 7 (0.05%)           |
| 924                 | 11 (0.08%)          |
| 1200                | 7 (0.05%)           |
| 1536                | 12 (0.08%)          |
| 1792                | 26 (0.19%)          |
| 2048                | 13 (0.09%)          |

Results presented in Table 3. Were mapped in Figure 4. In which it could be seen that at 128 and 1792 bytes there was an acute growth in the dropped packets, later at 128, 192, 256, 384, 512, 924 and 1200 bytes the dropped packets were in a high fluctuation state (almost unstable), finally at 1200, 1536 bytes the dropped packets rate was raising exponentially until it reached the highest dropped packets at 1792 Bytes. Primarily at 64 bytes there was no dropped packets investigated.

![Figure 5. Dropped Packets vs. CPU allocation](image)

13. Conclusions
POX controller operation and performance metrics such as the bandwidth utilization, CPU load and packet loss where analyzed and evaluated in this work through the use of network software measuring tools such as Iperf and D-ITG. It was found that Effective bandwidth utilization happens when the bandwidth starts from 100Mbps and rises gradually to 500 Mbps specifically when the numbers of utilized switches range from 20 till 30 switches. Furthermore, when 70-80% of CPU load were allocated to the hosts in addition to the deployment of CFS scheduling algorithm then the minimum delay, and the maximum received bytes, average packet rates and average Bit rate were investigated. Dropped packets increased both at 128 and greater than 1200 bytes which mainly caused the specified POX controller to be minimal reliable after.
References

[1] R. Masoudi and A. Ghaffari, 2016 Software defined networks: A survey, *J. Netw. Comput. Appl.*, 67, pp. 1–25.

[2] F. Hu, “Network Innovation through Openflow and SDN,” *Новостной ресурс Techtarget*, 2014.

[3] C. Fancy and M. Pushpalatha, “Performance evaluation of SDN controllers POX and floodlight in mininet emulation environment,” in *2017 International Conference on Intelligent Sustainable Systems (ICISS)*, 2017, pp. 695–699.

[4] F. Benamrane and R. Benaini, 2015 Performances of OpenFlow-based software-defined networks: an overview, *J. Networks*, 10(6), p. 329.

[5] P. Goransson, C. Black, and T. Culver, 2016 *Software defined networks: a comprehensive approach*. Morgan Kaufmann.

[6] A. S. Dawood and M. N. Abdullah, 2014 A Survey and a Comparative Study on Software-Defined Networking, *Int. Res. J. Comput. Sci. ISSN*, pp. 2393–9842.

[7] N. Feamster, J. Rexford, and E. Zegura, 2014 The road to SDN: an intellectual history of programmable networks,” *ACM SIGCOMM Comput. Commun. Rev.*, 44(2), pp. 87–98.

[8] S. Azodolmolky, *Software defined networking with OpenFlow*. Packt Publishing Ltd, 2013.

[9] W. H. Muragaai, K. Seman, and M. F. Marhusin, 2017 A POX Controller Module to Prepare a List of Flow Header Information Extracted from SDN Traffic, *Int. J. Comput. Syst. Eng.*, 11(12), pp. 1305–08.

[10] V. V Shkarupylo et al., “Software defined networks and Internet of Things.”, 2019

[11] S. Srivastava, S. Anmulwar, A. M. Sapkal, T. Batra, A. K. Gupta, and V. Kumar, 2014, Comparative study of various traffic generator tools,” in *2014 Recent Advances in Engineering and Computational Sciences (RAECS)*, pp. 1–6.