Comparative Analysis of Software Defined Networking (SDN) Controllers – In Terms of Traffic Handling Capabilities

Abhishek Rastogi
Electronic Systems Engineering
University of Regina
Regina, Canada
rastogia@uregina.ca

Abdul Bais
Electronic Systems Engineering
University of Regina
Regina, Canada
abdul.bais@uregina.ca

Abstract— Software Defined Networking (SDN) is an emerging paradigm that allows network operators to have more control of their infrastructure. Control is achieved by decoupling control and data plane of the network. Controller is the place where all the intelligence of SDN resides; hence it is important to test the capabilities of the controller before deploying it in production networks. This paper presents an analysis of the capabilities of two open-source python based OpenFlow SDN controllers (POX and RYU). The analyzed indexes include performance of both the controllers when subjected to layer 1 and layer 2 switching. For testing purposes Mininet emulator and Distributed-Internet Traffic Generator (D-ITG) traffic generator are used. The results of the evaluation, when layer 1 switching is in question, show, that POX has better traffic handling capabilities, and, as a result, a better performance. But, as far as layer 2 switching is concerned, RYU came up with far better performance results.

Keywords—Networking, Programming, SDN, POX, RYU, Traffic Generation, Emulator

I. Introduction

Computer network architecture used today (traditional network architecture) is complex and difficult to manage. The complexity of the traditional network architecture stems from the fact that it is "inflexible" and "static" in nature. It is static because it cannot be programmed or configured dynamically as per the requirements. The lack of flexibility is due to the "closed box" networking equipment which are pre-installed with proprietary software. Proprietary software installed on these equipments helps them in learning the network topologies, routing data packets quickly and efficiently but at the same time, make them inflexible. Network equipment manufacturers did a commendable job in making an equipment smarter but left the person (administrator) administering them with less control. Whenever requirements change, administrators are forced to go to all those individual small boxes to push the change. Hence, in a nutshell, traditional network architecture give more importance to the hardware (switches, firewalls, routers) and each hardware has an autonomous software installed on it which basically controls the behavior of that hardware (distributed control plane). Software Defined Networking (SDN) on the other hand is an open architecture. It overcomes the shortcomings of traditional networks by giving more importance to software than hardware, having centralized control plane, and by giving more control to network administrators. The prime purpose of SDN is to provide administrator the power to dynamically push changes as per requirements and to replace much expensive "closed boxes" with cheaper "open boxes" that are flexible and easy to manage.

"Software Defined Networking (SDN) allows customization and optimization that enables invention and delivery of new types of network services"[1]. According to ONF, the SDN Architecture is [2]:

- Directly Programmable [18]
- Agile
- Centrally Managed
- Programmatically Configured
- Open-Standards based and vendor-neutral

SDN architecture highly depends on the efficiency of the control plane, where SDN controller resides. In order to make control plane most efficient, it is very important to have a controller that not only takes less time to make and push decisions onto the forwarding plane but also effectively handles the modern network traffic challenges. Advancement of SDN architecture has given birth to a number of SDN controllers, and, with so many controllers out there, it often becomes a difficult task to choose which controller to work with.

A comparison of two openflow controllers (RYU and NOX) is reported in [21], in which EstiNet (emulator) is used to compare various performance indexes, mainly, link restoration in both the controllers. In [22] seven openflow controllers (NOX, POX, Beacon, Floodlight, MuL, Maestro, and RYU) are compared in terms of scalability, reliability, and security. In [23] four controllers (NOX, NOX-MT, Beacon, and Maestro) are compared to measure different performance aspects like, controller response time. In [24] the architectural evaluation of four SDN controllers (NOX, Beacon, Maestro, and Floodlight) is performed. To best of our knowledge, none

978-1-5090-4300-2/16/$31.00 ©2016 IEEE

of the controllers are tested for performance like, throughput, latency, jitter, and packet rate for basic switching (layer 1 and layer 2) functions. We feel there is a need to compare the performance of various open source controllers based on network traffic generated by a realistic traffic generator, like iperf, D-ITG, etc.

The objective of this paper is to present a comparative analysis of two SDN controllers (POX and RYU) in terms of their traffic handling capabilities. As a proposed solution, MININET (emulator) is used to emulate virtualized custom topologies to monitor the traffic and performance. Distributed –Internet Traffic Generator (D-ITG) is used to generate TCP traffic through those custom topologies. We measure common characteristics of network performance such as throughput, delay, and jitter, for comparing the two controllers.

The organization of this paper is as follows: Section II discusses the evolution of SDN architecture, SDN controllers, and OpenFlow protocol. Section III discusses the Layer 1, and Layer 2 traffic analysis based on POX and RYU controllers. Section IV compares, contrasts, and provides results after analyzing performance obtained in section III. The paper is finally concluded in section V.

II. Background

Software-Defined Networking architecture is based on a new network representation that decouples control plane and data plane.

The architecture has been sub-divided into three layers: Application Layer, Control Layer, and Infrastructure Layer.

- Infrastructure Layer: This is the bottom most layer of the architecture and is mainly composed of all the physical devices like, switches, router, and firewalls.
- Control Layer: This is the middle layer of the architecture and is often represented as "intelligent layer" of the architecture. Network intelligence (Controller) resides in this layer that controls the behavior of physical elements present in infrastructure layer.
- Application Layer: This is the top most layer of the SDN architecture where network administrator can operate. The requirements of the whole network can be dynamically configured at this layer using customized SDN programs.

All three layers communicate with each other and this communication is carried out by Application Program Interfaces (APIs). These are commonly referred to as:

- Southbound API: Control layer talks to infrastructure layer using southbound API. Using these API control layer can dynamically instruct elements in the infrastructure layer on how to operate.
- Northbound API: Application layer talks to control layer using northbound API. Using these API administrators can dynamically instruct controller on how to make adjustments to the elements in the infrastructure layer.

A. Communication using OpenFlow – Operating Principles

This section goes through the concepts showcasing communication between OpenFlow switches and controller. Whenever an OpenFlow switch receives a packet that it has never seen before or which does not matches with the flow entries already present in flow table of the switch, it forwards that packet to the controller to handle [13]. The controller decides what to do with that particular packet. It can discard, drop, or send it back to the switch instructing where to send whenever similar type of packet arrives in future. Controller instructs the switch using OpenFlow protocol by pushing new entry in the flow table of the switch, which outlines where to send that kind of packet in future.

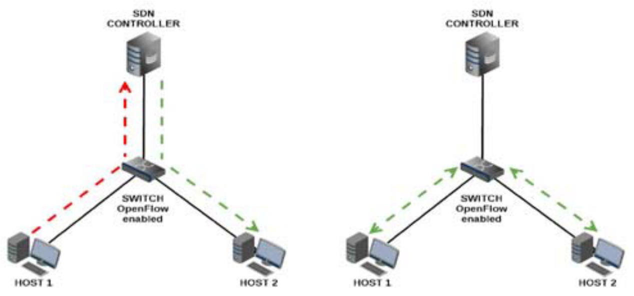

Figure 1: SDN Operating Principles [4]

Figure 1 illustrates the communication between SDN controller and OpenFlow switch. The figure shows one OpenFlow switch with two hosts connected to it (Host 1 and Host 2). Switch is also virtually connected to SDN controller which can communicate with switch using OpenFlow protocol. Now, if Host 1 wants to communicate with Host 2, it will send the connection request packet to the switch. As switch is not capable to determine what to do with this packet (assuming the flow table is empty) it will send this packet to the controller to take necessary actions (drop, discard, or route). The red dashed line shows that the packet was forwarded to controller rather than Host 2. Now, controller after inspecting the header, source, and destination of the packet, decided that this packet needs to be re-routed to the Host 2 connected to the same switch. Controller pushes down the flow entry into the flow table of the switch instructing it to forward the packet to Host 2. The green dashed line shows that controller sent the packet back to the switch which forwarded the packet to Host 2. Now, after sometime, if Host 1 again wants to communicate with Host 2 and sends connection request packet to the switch, this time packet will be forwarded to Host 2 without asking controller, as switch already have the flow entry (pushed earlier by the controller) in the flow table. This is shown by the green dashed line in the figure on the right hand side.

B. SDN Controllers

This section discusses different types of SDN controllers available as of today. SDN controllers are software applications that control the behavior of elements at the infrastructure layer level [20]. These applications are written in many different computer programming languages like, Java, Python, C, C++, and Ruby. These applications can be open source or license based. Most common SDN controllers present nowadays are OpenFlow based controllers i.e. which can support OpenFlow protocol and can communicate with lower layers over a secure medium [16]. Following are the most common and widely used OpenFlow controllers:

TABLE I. SDN CONTROLLERS

Controller Name	Programming Language
NOX [8]	C++
POX [7]	Python
Beacon [6]	Java
Floodlight [10]	Java
OpenDayLight [9]	Java
Ryu [5]	Python

III. ENVIRONMENT SETUP AND EVALUATION METHODOLOGY

This section describes the environment under which all the experiments are done. All the experiments are done on a virtual machine (VM) running on a laptop with the configuration shown in Table II.

TABLE II. EXPERIMENT SPECIFICATIONS

Type	Description
Laptop (Physical Machine)	Windows 10, HP G42 Notebook PC, Intel Core i3 processor, 3.0GB RAM
VMware (Virtual Machine on Laptop)	Linux, Ubuntu (15.04), 1 GB RAM, 64-bit OS.
Mininet [11] [19]	Used for emulating virtual SDN networks. Support Python programming language.
Distributed Internet Traffic Generator (Traffic Generator) [12]	Capable of generating traffic with real network traffic characteristic. Version 2.8.1
Cisco Packet Tracer (Student License)	Used to draw customized network topologies used in experiments. Ver: 6.2.0.0052

A. *Experiment Network Topologies*

This section describes different network topologies under test. There are many standard topologies available as of today, like, linear topology, bus topology, star topology, linear topology, and tree topologies. Real world topologies used nowadays are the combination of the above stated standard ones [17]. Topologies generally used in a data center type of a setting come from a tree like topology. Core switch is the root of the tree and is usually one in number, but can be more. Core switches then connect to number of aggregate switches that resembles to the trunk of the tree. Aggregate switches then connect to Hosts, which form the distributed structure (leaves) of the tree.

Network topologies used in this paper come from a data centre type of a setting but in a simplified form. In this project 3 different custom topologies are used to determine the performance of SDN controller controlling them.

1) Topology 1 –

The topology shown in figure 5, consists of 5 OpenFlow switches (S1, S2, S3, S4, S5), 12 hosts (H1 to H12), and an OpenFlow SDN controller (C0) controlling every switch.

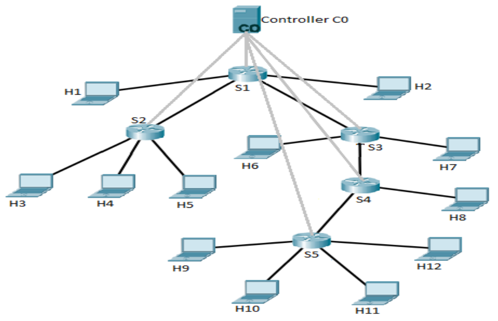

Figure 2: Topology 1 – Combination of standard (Linear and Tree) topology

2) Topology 2 –

The topology shown in figure 3, consists of 15 OpenFlow switches (S1 to S15), 15 hosts (H1 to H15), and an OpenFlow SDN controller (C0) controlling every switch. This is the extension of standard tree topology.

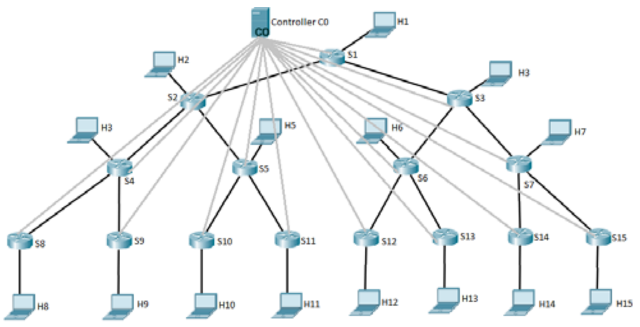

Figure 3: Topology 2 – Data centre type Tree topology

3) Topology 3 –

The topology shown in figure 3, consists of 6 OpenFlow switches (S1, S2, S3, S4, S5, S6), 12 hosts (H1 to H12), and an OpenFlow SDN controller (C0) controlling every switch. This is an example of fully connected standard topology.

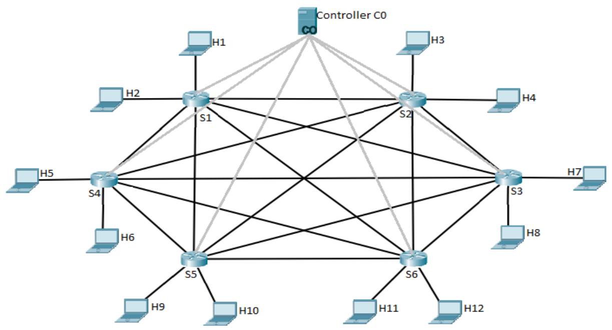

Figure 4: Topology 3 – Fully connected Mesh topology

B. *Layer 1 Switching*

Layer 1 switching basically deals with unintelligent devices like, hubs, and repeater which have no intelligence whatsoever preinstalled on it. The function of layer 1 devices is to flood every host connected to it. SDN switches can be tailored to behave like, hub or repeater.

In case of layer 1 switching, controller cannot push in flows as hubs and repeaters do not have the capability to store information. Hence, every time a packet comes in, it is forwarded to the controller and then controller decides to flood all the hosts connected to it. It is the time that it takes to make decision is important, because, the time taken in deciding where the packet needs to go is going to affect overall network bit-rate, hence, performance.

C. *Layer 2 Switching*

Layer 2 switching defines how data is assembled, in form of frames, for transmission. Layer 2 devices also provide connection interface between the layer 2 network devices and devices at physical forwarding plane. The function of layer 2 devices is to route data depending on the entries in the flow table.

In case of layer 2 switching, controller can push in flows as switches are more intelligent than hubs and can control the routing. Controller is responsible to instruct physical elements on how to handle incoming data packets. It is also responsible to pushing flows into the switch flow table. Switch has an ability to learn on how to handle packets coming to ingress port and exiting egress port. Whenever a packet arrives at the ingress port of the switch for the first time, it is forwarded to the controller. Flow table of a switch at the initial stage looks like shown in table III. Where "X" represents an unknown entity, S1 is switch 1 and S2 is switch 2 in a sample hypothetical topology.

Table III clearly shows behavior explained above. Every packet is forwarded to the controller as switch flow table is populated like that. When controller has the packet, it decides what to do and where to forward the packet and the same is communicated to the switch by inserting the flow entry into the switch flow table

TABLE III. SWITCH FLOW TABLE (INITIALLY)

Switch	Ingress Port	Source MAC	Dest. MAC	Priority	Action/Egress Port
S1	1	X	X	X	Forward to Controller
S2	2	X	X	X	Forward to Controller

When controller has the packet, it decides what to do and where to forward the packet and the same is communicated to the switch by inserting the flow entry into the switch flow table. Controller can push in flows using standard commands that tell the switch to forward the packets coming to ingress port 1 to egress port 2, having a priority of 35000 and vice-versa.

TABLE IV. SWITCH FLOW TABLE (AFTER CONTROLLER PUSHED THE ENTRY)

Switch	Ingress Port	Source MAC	Dest. MAC	Priority	Action/ Out Port
S1	1	X	X	35000	Forward /2
S2	2	X	X	35000	Forward /1

Every packet is now being forwarded as per the flow table IV. As of now, switch has not forwarded the packet yet. As soon as, switch forwards the packet to the host connected to port 1, switch learns the MAC address of that host and stores it in the flow table. Let's say, host connected to port 1 has MAC (1020304050) and host connected to port 2 has MAC (1050403020). Switch updates the flow table as shown in table V.

TABLE V. LEARNING SWITCH FLOW TABLE

Switch	In Port	Source MAC	Dest. MAC	Priority
S1	1	1020304050	1050403020	35000
S2	2	1050403020	1020304050	35000

D. *Experiment Traffic Generation*

Traffic is generated using Distributed Internet Traffic Generator (D-ITG). Single flow TCP traffic is being generated with a constant payload of 1000 bytes for 10 seconds while calculating round trip delay.

IV. SIMULATION RESULTS

In this section, discussion on results is done and a comparative conclusion is provided.

A. *Layer 1 switching – POX vs RYU*

Under this section, performance of both the controllers is discussed when they are subjected to layer 1 switching.

When topology 1, 2, and 3 were subject to layer 1 switching, we observed the following characteristics:

- RYU controller came up with less bit-rate than when POX controller was controlling the hub behavior of the network.
- RYU controller has higher average delay when compared to POX controller.
- RYU controller has higher average jitter when compared to POX controller.
- As bit-rate is less for RYU we can say that average packet rate will also be less for RYU when compared to POX.

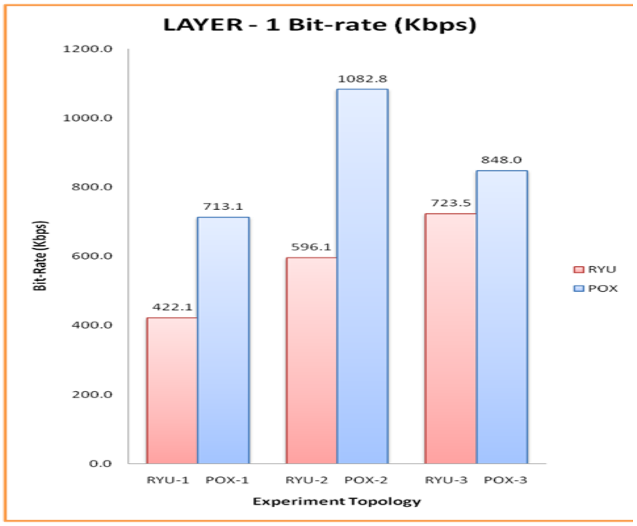

Figure 5: Layer 1 Bit rate comparison (POX vs RYU)

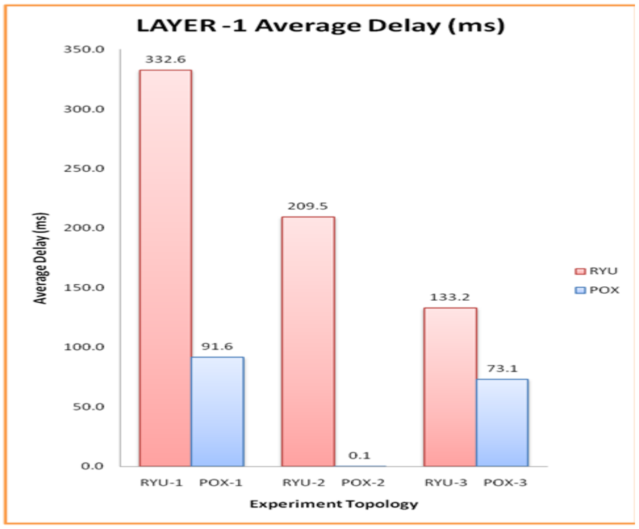

Figure 6: Layer 1 Average delay comparison (POX vs RYU)

Figure 5 and 6 represents a performance comparison between POX and RYU when subjected to all three topologies in question. Y-axis in figure 5 is a bit-rate (in kbps) while X-axis represents topology and controller (for e.g. Ryu-1 is representing RYU controller with Topology 1). In figure 6, Y axis is the Average Delay (in msec).

From the results above, it can be concluded that POX controller is better than RYU when layer 1 switching is in question. POX controller is better in handling scenarios like, broadcasting ARP requests in the network.

B. *Layer 2 switching – POX vs RYU*

Under this section, performance of both the controllers is discussed when they are subjected to layer 2 switching.

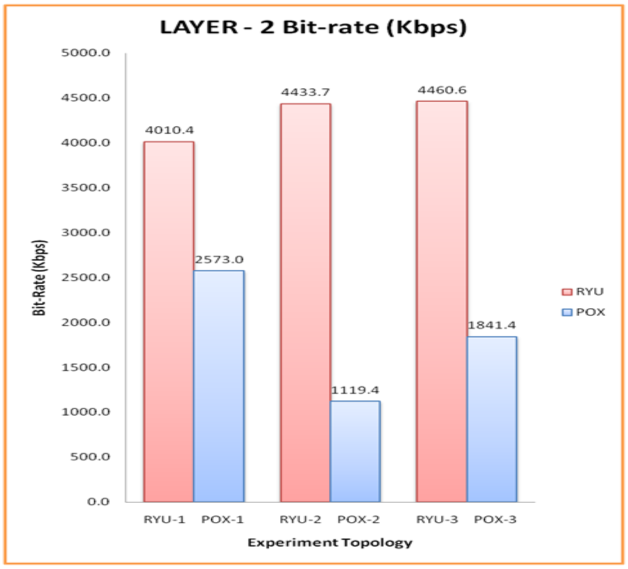

Figure 7: Layer 2 Bit rate comparison (POX v/s RYU)

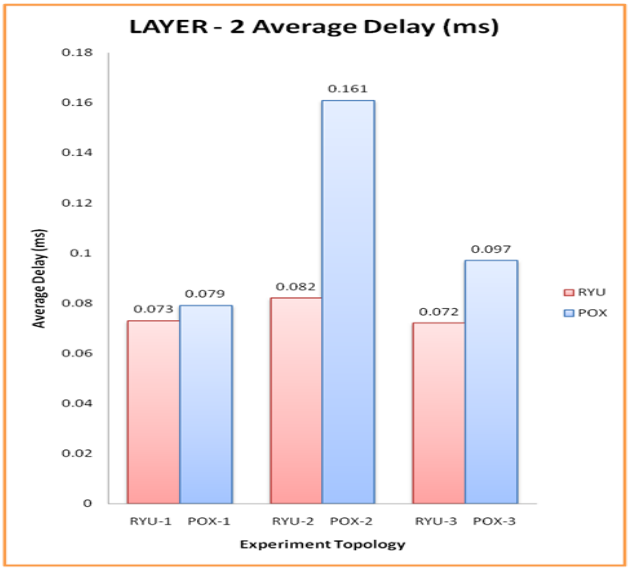

Figure 8: Layer 2 Average delay comparison (POX v/s RYU)

When topology 1, 2, and 3 were subject to layer 2 switching, results showcased in figure 7 and 8 explains the following:

- RYU controller came up with much higher bit-rate than when POX controller was controlling the layer 2 learning behavior of the network.
- RYU controller has comparable average delay when compared to POX controller.
- RYU controller has lesser average jitter when compared to POX controller.
- As bit-rate is higher for RYU we can say that average packet rate will also be higher for RYU when compared to POX.

From the results above, it can be concluded that RYU controller is better than POX when layer 2 switching is in question. RYU controller is better in handling scenarios like, routing new packets to their destination, pushing flows to the switch flow table, and instructing switches on how to handle repetitive packets in the network.

V. CONCLUSION AND FUTURE WORK

In this paper, performance of two open-source python based SDN controllers (POX and RYU) is analyzed. We selected POX and RYU because we wanted to have a fair comparison between the two. These two controllers have been analyzed in order to collect their traffic handling capabilities in terms of several network characteristics like, bit-rate, average delay, packet-rate, and average jitter. We found that, when layer 1 switching is in question, POX has better traffic handling capabilities (high bit-rate, less delay and jitter), and, as a result, a better performance. Meaning, POX controller is better in handling scenarios like, broadcasting ARP requests in the network. But, when layer 2 switching is in question, RYU came up with far better performance results. Meaning, RYU controller is better in handling scenarios like, routing new packets to their destination, pushing flows to the switch flow table, and instructing switches on how to handle repetitive packets in the network.

The study can be used as a template for SDN developers or researchers to help facilitate in the SDN controller choice process based on what their requirements are [20]. The future work would be to research the architecture of both the controllers to help understand the result pattern we got here.

REFERENCES

[1] Open Networking Summit. (2011, October 17-19). Retrieved October 15, 2015, from http://opennetsummit.org/archives/oct11/site/why.html

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, J. Turner, S. Shenker, (April 2008). OpenFlow: Enabling Innovation in Campus Networks. ACM SIGCOMM Computer Communication Review, (pp. 69-74). New York.

[3] What are SDN northbound APIs. (n.d.). Retrieved November 12, 2015, from SDN Central: https://www.sdxcentral.com/resources/sdn/north-bound-interfaces-api/

[4] F. Ongaro, A. Corradi, M. Gerla, E. Cerqueira, L. Foschini, (2013). Enhancing Quality of Service in Software-Defined Networks. Bologna.

[5] Build SDN Agilely. Retrieved November 11, 2015, from GitHub: https://osrg.github.io/ryu/

[6] D. Erickson. (2013). The Beacon OpenFlow Controller. HotSDN '13 Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, (pp. 13-18). New York, NY USA.

[7] S. Kaur, J. Singh, N. S. Ghumman. (2014). Network Programmability Using POX. International Conference on Communications, Computing and Systems, (pp. 134-138). Punjab, India.

[8] Rao, S. (2015, March 21). SDN Series Part 3: NOX, the Original OpenFlow Controller. Retrieved Nov 23, 2015, from http://thenewstack.io/sdn-series-part-iii-nox-the-original-openflow-controller/

[9] W. Braun, M. Menth. (2014). Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices. Future Internet , 302-336.

[10] Floodlight Controller. Retrieved December 13, 2015, from Project Floodlight: http://www.projectfloodlight.org/floodlight/

[11] An Instant Virtual Network on your Laptop. Retrieved September 30, 2015, from Mininet: http://mininet.org/

[12] S. Mishra, S. Sonavane, A. Gupta. (2015). Study of Traffic Generation Tools. International Journal of Advanced Research in Computer and Communication Engineering, (pp. 159-162).

[13] P. H. Isolani, J. A. Wickboldt, C. B. Both, & L. Z. Granville, (2015). Interactive monitoring, visualization, and configuration of OpenFlow-based SDN. 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM) (pp. 207-215). Ottawa, ON: IEEE.

[14] H. Kim, N. Feamster, (2013). Improving Network Management with Software Defined Networking. IEEE Communications Magazine (Volume:51 , Issue: 2) (pp. 114-119). IEEE.

[15] A. Yassine, H. Rahimi (2015). Software Defined Network Traffic Measurement: Current Trends and Challenges. IEEE Instrumentation & Measurement Magazine (Volume:18 , Issue: 2) (pp. 42-50). IEEE.

[16] M. P. Fernandez, (2013). Evaluating OpenFlow Controller Paradigms. ICN 2013 : The Twelfth International Conference on Networks, (pp. 151-157).

[17] Veena S, C. Pal, R. P. Rustagi, K. N. B. Murthy, (2014). A Framework for Implementing Realistic Custom Network Topology in Mininet. International Journal of Science and Research (IJSR) , 1316-1323.

[18] N. Feamster, J. Rexford, (2014). The Road to SDN: An Intellectual history of Programmable Networks. ACM SIGCOMM Computer Communication Review, Volume 44 Issue 2 (pp. 87-98). New York

[19] S. Kaur, J. Singh (2014). Mininet as Software Defined Networking Testing Platform . ICCS, (pp. 139-142). Punjab, India.

[20] R. Khondoker, (2014). Feature-based comparison and selection of Software Defined Networking (SDN) controllers. 2014 WCCAIS (pp. 1-7). Hammamet: IEEE.

[21] S. Y. Wang, H. W. Chiu, C. L. Chou, (2015). Comparisons of SDN OpenFlow Controllers over EstiNet: Ryu vs. NOX. The International Symposium on Advances in Software Defined Networks (pp. 1-6). Barcelona, Spain: IEEE

[22] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, R. Smeliansky, (2013). Advanced study of SDN/OpenFlow Controllers. CEE-SECR '13 Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia (Article No. 1). New York: IEEE

[23] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, R. Sherwood, (2012). On controller performance in software-defined networks. Proceeding Hot-ICE'12 Proceedings of the 2nd USENIX conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (pp. 10-15). Berkeley: ACM

[24] S. A. Shah, J. Faiz, M. Farooq, A. Shafi, S. A. Mehdi, (2013). An architectural evaluation of SDN controllers. 2013 IEEE International Conference on Communications (ICC) (pp. 3504-3508). Budapest:IEEE