Network Programmability Using POX Controller

Conference Paper · August 2014

DOI: 10.13140/RG.2.1.1950.6961

3 authors:

Sukhveer Kaur
INTI International University and Colleges
9 PUBLICATIONS **238** CITATIONS
SEE PROFILE

Japinder Singh
Shaheed Bhagat Singh State Technical Campus
12 PUBLICATIONS **321** CITATIONS
SEE PROFILE

Navtej Ghumman
Shaheed Bhagat Singh State University
27 PUBLICATIONS **341** CITATIONS
SEE PROFILE

Network Programmability Using POX Controller

Sukhveer Kaur[1], Japinder Singh[2] and Navtej Singh Ghumman[3]
[1,2,3]Department of Computer Science and Engineering,
SBS State Technical Campus, Ferozepur, India
E-mail: [1]bhullarsukh96@gmail.com, [2]japitaneja@gmail.com,
[3]navtejghumman@yahoo.com

***Abstract*—POX is a Python based open source OpenFlow/Software Defined Networking (SDN) Controller. POX is used for faster development and prototyping of new network applications. POX controller comes pre installed with the mininet virtual machine. Using POX controller you can turn dumb openflow devices into hub, switch, load balancer, firewall devices. The POX controller allows easy way to run OpenFlow/SDN experiments. POX can be passed different parameters according to real or experimental topologies, thus allowing you to run experiments on real hardware, testbeds or in mininet emulator. In this paper, first section will contain introduction about POX, OpenFlow and SDN, then discussion about relationship between POX and Mininet. Final Sections will be regarding creating and verifying behavior of network applications in POX.**

Keywords: POX, SDN, Open Flow, Mininet

I. Introduction

SDN separates the control plane of networking device (switch/ router) from its data plane, making it possible to control, monitor, and manage a network from a centralized controller.

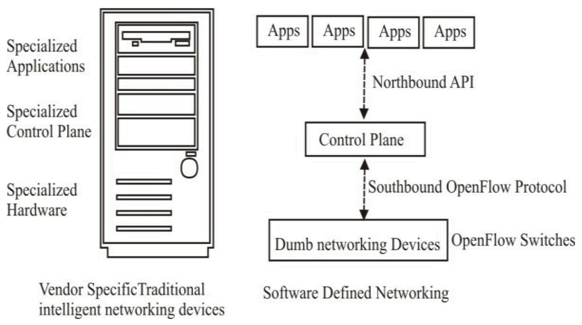

Specialized Applications
Specialized Control Plane
Specialized Hardware
Vendor Specific Traditional intelligent networking devices
Apps
Apps
Apps
Apps
Northbound API
Control Plane
Southbound OpenFlow Protocol
Dumb networking Devices
OpenFlow Switches
Software Defined Networking

Fig. 1 Decoupled Control and Data Plane

Software Defined Networking [1] tries to simplify the development of new applications by separating the data plane from control plane. Control plane is also called controller. This controller has a global view of the network and controls the flow through the network. Since most intelligence is now transferred to the controller, the switch only perform the actions that the controller requests. This makes the switches very simple and inexpensive. But in traditional networks (Fig. 1), each device has vendor-specific operating system to control the data plane. Additional applications can be implemented on top of this operating system.

POX [2] is an open source controller for developing SDN applications. POX controller provides an efficient way to implement the OpenFlow protocol which is the de facto communication protocol between the controllers and the switches. Using POX controller you can run different applications like hub, switch, load balancer, and firewall. Tcpdump packet capture tool can be used to capture and see the packets flowing between POX controller and OpenFlow devices.

Communication between the controller and the switches is carried by communication protocol such as OpenFlow [3], ForCES [4] (Fig. 2). OpenFlow is the most popular standard protocol used in SDN. OpenFlow switches behave as dumb forwarding devices. They are unable to perform any actions without programmed by the controller.

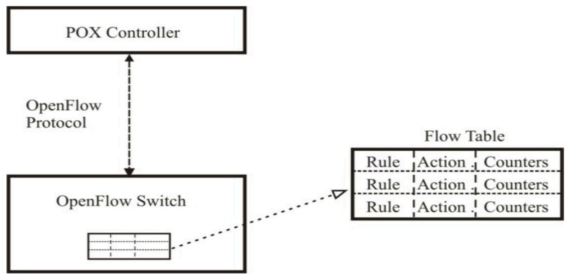


Fig. 2 POX Controller

When a switch is powered on, it will immediately connect to an OpenFlow controller. Initially, the flow table of the switches is empty. When a packet arrives at a switch, it does not know, how this packet is to be handled. Then it send packet-in message to the controller. To handle the packet, controller inserts a flow entries in flow table of switch. Flow entry in flow table contains three parts, rule(match field), action, counters. For each packet, that has to pass through a switch, a flow entry will have to be installed so that the switch can forward this traffic without further intervention of the controller [5]. Flow modification messages are sent to the switches to install the flow entries in flow table (Fig. 3). Once these are installed, traffic belonging to this flow will be handled by the switches themselves.

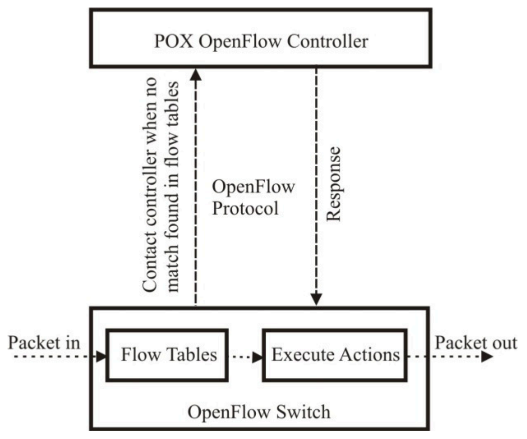


Fig. 3 SDN Architecture

II. POX AND MININET

Mininet is an emulation tool that allows running a number of virtual hosts, controllers, switches, and links. It uses container based virtualization to make a single system act as a complete network. It is a simple, robust and inexpensive network tool to develop and test OpenFlow based applications. Mininet [6] can create a complex network topology for testing purposes, without configuring the physical networks. It supports custom topologies. It supports simple and extensible Python API for network creation and testing.

Mininet combines the desirable features of simulators, testbeds and emulators. Mininet is cheaper, easily available, and quickly reconfigurable as compared to testbeds such as GENI [7], VINI [8], and Emulab [9]. It runs real, unmodified code as compared to simulators such as EstiNet [10], ns-3 [11]. The code that is to be developed in Mininet, can also run in real network without any modifications. It supports large scale networks containing large number of virtual hosts and switches. In short, Mininet's virtual hosts, switches, links, and controllers are just like the real thing. They are just created using software rather than hardware.

Mininet have built-in Controller classes to support different network controllers such as reference controller (controller), ovs-controller [12] and less used NOX Classic [13].

You can choose controller by invoking ‘mn’ command.

```
# mn --controller ref
# mn --controller ovsc
# mn --controller nox
```

Five most important open source controllers (Table I) that can be used by Mininet remotly are POX, Ryu [14], Trema [15], FloodLight [16], and OpenDaylight [17]. There are number of other SDN controllers like NOX (C++) , Jaxon (Java) [18], Beacon (Java) [19], Maestro (Java) [20] which are not considered because they are deprecated and poorly documented.

A. POX

The POX is a python based SDN controller that is inherited from the NOX controller.

B. Ryu

Ryu is a component-based SDN controller. Ryu has a collection of built-in components. These components can be changed, extended and composed for creating new customized controller applications. Any programming language can be used to develop a new component.

C. Trema

Trema is a framework for Ruby and C that builds software platform for OpenFlow developers. It is easy to use Open Source free software.

TABLE 1 DIFFERENT SDN CONTROLLERS

	POX	**Ryu**	**Trema**	**Floodlight**	**Open Day Light**
Language Support	Python	Python	C Ruby	Java	Java
OpenFlow Support	v1.0	v1.0 v1.2 v1.3	v1.0	v1.0	v1.0
OpenSource	Yes	Yes	Yes	Yes	Yes
GUI	Yes	Yes	No	Web GUI	Yes
REST API	No	Yes	No	Yes	Yes
Platform Support	Linux Mac Windows	Linux	Linux	Linux	Linux Mac Windows

D. Floodlight

The Floodlight Open SDN Controller is an Apache licensed, enterprise class, Java based OpenFlow controller. FloodLight controller contains a number of modules, where each module provides a service to the other modules and to the control logic application through simple Java API or a REST API.

E. OpenDayLight

OpenDayLight is an open source project. The goal of the project is to create robust code that covers major components of the SDN architecture, to gain acceptance among the vendors and users, and to have a growing community that contributes to the code and uses the code for commercial products.

To use POX controller, type the following command in terminal window.

```
# python pox.py log.level –DEBUG
```

Using this command POX controller runs in DEBUG mode. DEBUG mode allows display of additional messages exchanged with the switch. To launch Mininet with default topology of 1 switch and 2 hosts run the following command.

```
# mn
```

In this case switch will connect to the default ovs controller. If you want to use POX controller running on the same Mininet machine you need to run the 'mn' command with the controller option having the parameters set to 'remote'. Loopback address '127.0.0.1' will be used as ip address. The following command will connect the switches to remote POX controller running on another terminal.

```
# mn –controller=remote, ip=127.0.0.1
```

But if POX controller is on different machine (suppose 172.24.0.1), then run the following command

```
# mn --controller=remote, ip=172.24.0.1
```

You can create complex pre defined or custom defined topologies using Mininet. For example

```
# mn –mac --topo single,5 --switch ovsk --controller remote
```

This will create 5 hosts and 1 switch topology. The different options that can be used with 'mn' command are shown in Table II.

TABLE 2 MININET OPTIONS

Commands	Description
mn	run Mininet
--topo single, 5	create 1 switch with 5 hosts
--mac	makes mac address same as node number on hosts
--arp	install static ARP entries
--switch ovsk	use Open vSwitch
--controller remote	use remote controller
--ip	remote controller ip address

III. POX APPLICATIONS

There are various applications that can be created using POX. The network application could be a simple hub, switch, and router or could be sophisticated middle boxes such as firewall or load balancer. This section contains simple hub logic and application code.

A. *Hub Application*

If a flow entry in flow table contains action to flood the packet that arrives at specific port of forwarding device, then that device act like a hub. In the topology shown in Fig. 4, all hosts belong to the same network. When host h1 wants to send a packet to host h4, then it first sends a packet to forwarding device at port 1. When a packet arrives at port 1, then it matched against flow entry. When match is found, then it is flooded to all ports except the incoming port according to action specified in flow entry. If no match is found, then packet is forwarded to controller. In one terminal window, run the following command to create an experiment topology.

```
# mn --mac --topo single,5 --switch ovsk --controller remote
```

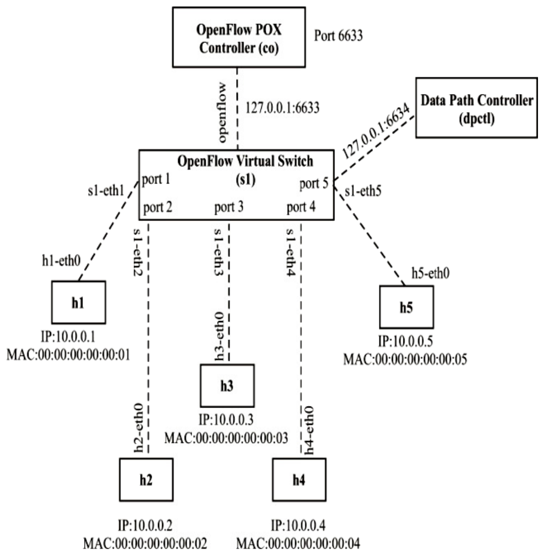


Fig. 4 Single Switch, 5 Hosts Topology

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpidToStr

log = core.getLogger()


def _handle_ConnectionUp (event):
  msg = of.ofp_flow_mod()
  msg.actions.append(of.ofp_action_output(port =
of.OFPP_FLOOD))
  event.connection.send(msg)
  log.info("Hubifying %s", dpidToStr(event.dpid))

def launch ():
  core.openflow.addListenerByName("ConnectionUp",
_handle_ConnectionUp)

  log.info("Hub running.")
```

Listing 1 Hub Application Code

This will launch Mininet network topology consisting of 1 OpenFlow switch, 1 OpenFlow controller and 5 hosts. The POX controller comes pre-installed with the provided VM image. From another terminal window, run the hub code (code file name is 'hub.py') shown in Listing 1 by using the following command.

```
# python pox.py log.level --DEBUG hub
```

This will launch the POX controller in verbose mode for debugging purposes and also run the hub application.

B. *Understanding Hub Application Code*

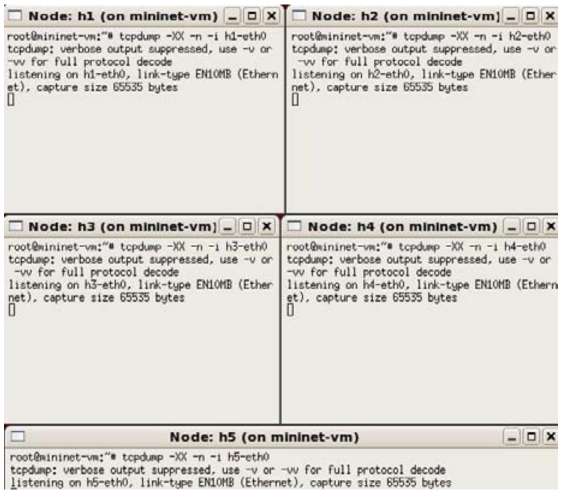
Before implement a hub application, first you need to import a core object that show a connection between modules in POX and OpenFlow library that is used for access a number of primitives.

1. ofp_action_output class: This class specifies a switch port, where you want to send the packet. There are various "special" port numbers. For example in the hub application, 'OFPP_FLOOD' which sends the packet to all ports except the incoming port.
2. ofp_flow_mod OpenFlow message: This message is send from controller to switch to insert flow table entry. Flow table entries will be matched against fields of incoming packets and then perform some actions on matching packets.
3. connection.send(...): Controller sends an OpenFlow message to a switch by using this function. A 'ConnectionUp' event is fired, When a connection to a switch starts. The above code call a '_handle_ConnectionUp ()' function that contains hub logic.
4. launch(): The launch() function is automatically called, when the application is started. The application registers all event listeners in this function.
5. dpid_to_str(): Each OpenFlow switch has a unique 64 bit datapath ID (DPID) and that is to be passed to controller from switch during handshaking. 48 bits are Ethernet address and 16 bits are implementation defined. It is a decimal number that is not easy to understand. POX define a pox.lib.util.dpid_to_str () function to format DPIDs.

IV. Verifying HUB Behavior

To verify hub behavior, Start a topology that contains single switch and 5 hosts and run it with POX controller. From host h1 sends icmp packets to the host h3. Here all the hosts see the same exact traffic which is the default behavior of hub. Launch an 'xterm' for each host and view the traffic simultaneously for each host by using 'tcpdump' (Fig. 5). For this purpose start 5 xterm terminals, one for each host.

The command for viewing traffic is "tcpdump". Pass the option '-XX' for verbose output, '-i' for specifying the interface for listening, '-n' for no name resolution.



```
Node: h1 (on mininet-vm)
root@mininet-vm:~# tcpdump -XX -n -i h1-eth0
tcpdump: verbose output suppressed, use -v or
-vv for full protocol decode
listening on h1-eth0, link-type EN10MB (Ethern
et), capture size 65535 bytes
```

```
Node: h2 (on mininet-vm)
root@mininet-vm:~# tcpdump -XX -n -i h2-eth0
tcpdump: verbose output suppressed, use -v or
 -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ether
net), capture size 65535 bytes
```

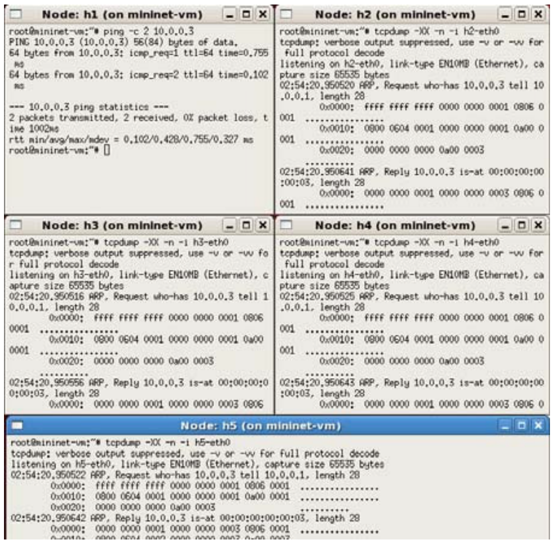
```
Node: h3 (on mininet-vm)
root@mininet-vm:~# tcpdump -XX -n -i h3-eth0
tcpdump: verbose output suppressed, use -v or
 -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ether
net), capture size 65535 bytes
```

```
Node: h4 (on mininet-vm)
root@mininet-vm:~# tcpdump -XX -n -i h4-eth0
tcpdump: verbose output suppressed, use -v or
-vv for full protocol decode
listening on h4-eth0, link-type EN10MB (Ethern
et), capture size 65535 bytes
```

```
Node: h5 (on mininet-vm)
root@mininet-vm:~# tcpdump -XX -n -i h5-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h5-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
```

Fig. 5 Tcpdump Outputbefore Running 'Ping' Utility

Now from host h1 ping to the host h3 at address 10.0.0.3. Ping packets will first go to the controller, which will then flood the packets to all hosts except the interface which sent the packet. You will see identical ICMP and ARP packets related to the ping in all the terminals. (Fig. 6) thus verifying the behavior of hub.



```
Node: h1 (on mininet-vm)
root@mininet-vm:~# ping -c 2 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_req=1 ttl=64 time=0.755
 ms
64 bytes from 10.0.0.3: icmp_req=2 ttl=64 time=0.102
 ms

--- 10.0.0.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, t
ime 1002ms
rtt min/avg/max/mdev = 0.102/0.428/0.755/0.327 ms
root@mininet-vm:~# 
```

```
Node: h2 (on mininet-vm)
root@mininet-vm:~# tcpdump -XX -n -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for
 full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), ca
pture size 65535 bytes
02:54:20.950520 ARP, Request who-has 10.0.0.3 tell 10
.0.0.1, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0
001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0
001  ................
        0x0020:  0000 0000 0000 0a00 0003
     ..........
02:54:20.950641 ARP, Reply 10.0.0.3 is-at 00:00:00:00
:00:03, length 28
        0x0000:  0000 0000 0001 0000 0000 0003 0806 0
001  ................
```

```
Node: h3 (on mininet-vm)
root@mininet-vm:~# tcpdump -XX -n -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv fo
r full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), c
apture size 65535 bytes
02:54:20.950516 ARP, Request who-has 10.0.0.3 tell 1
0.0.0.1, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806
0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00
0001  ................
        0x0020:  0000 0000 0000 0a00 0003
      ..........
02:54:20.950556 ARP, Reply 10.0.0.3 is-at 00:00:00:0
0:00:03, length 28
        0x0000:  0000 0000 0001 0000 0000 0003 0806
```

```
Node: h4 (on mininet-vm)
root@mininet-vm:~# tcpdump -XX -n -i h4-eth0
tcpdump: verbose output suppressed, use -v or -vv for
 full protocol decode
listening on h4-eth0, link-type EN10MB (Ethernet), ca
pture size 65535 bytes
02:54:20.950525 ARP, Request who-has 10.0.0.3 tell 10
.0.0.1, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0
001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0
001  ................
        0x0020:  0000 0000 0000 0a00 0003
     ..........
02:54:20.950643 ARP, Reply 10.0.0.3 is-at 00:00:00:00
:00:03, length 28
        0x0000:  0000 0000 0001 0000 0000 0003 0806 0
```

```
Node: h5 (on mininet-vm)
root@mininet-vm:~# tcpdump -XX -n -i h5-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h5-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
02:54:20.950522 ARP, Request who-has 10.0.0.3 tell 10.0.0.1, length 28
        0x0000:  ffff ffff ffff 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0000 0a00 0003                 ..........
02:54:20.950642 ARP, Reply 10.0.0.3 is-at 00:00:00:00:00:03, length 28
        0x0000:  0000 0000 0001 0000 0000 0003 0806 0001  ................
```

Fig. 6 Tcpdump Output after Running 'Ping' Utility

V. Conclusion

POX controller can be used to convert cheap, dumb merchant silicon devices into hub, switch, router or middleboxes such as firewall, load balancer. POX is also great tool for deploying and testing SDN applications. Its great strength lies in that it can be used with real hardware, in testbeds or with Mininet emulator. The POX controller has some great features but does not have GUI interface. Open Flow v1.0 is most widely used version. Open Flow version 1.3 will be the next version that is supposed to be widely implemented in products. POX supports only v1.0. So support for v1.3 could be future challenge area. The network applications created in POX controller can not be used with other controllers. Porting of POX network applications to other controllers can be another research area.

ACKNOWLEDGMENT

The authors would like to thank Mr. Vipin Gupta from U-Net Solutions, Moga, India for his valuable assistance for this work.

REFERENCES

[1] Nunes, B.; Mendonca, M.; Nguyen, X.; Obraczka, K.; Turletti, T., "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *Communications Surveys & Tutorials, IEEE* , vol.PP, no.99, pp.1,18.

[2] Fernandez, Marcial. "Evaluating OpenFlow controller paradigms." In *ICN 2013, The Twelfth International Conference on Networks*, pp. 151-157. 2013.

[3] Lara, Adrian, Anisha Kolasani, and Byrav Ramamurthy. "Network innovation using openflow: A survey." (2013): 1-20.

[4] Zhou, Lei, Ligang Dong, and Rong Jin. "Research on ForCES Configuration Management Based on NETCONF." *Information Technology Journal* 13, no. 5 (2014).

[5] Kim, Hyojoon, and Nick Feamster. "Improving network management with software defined networking." *Communications Magazine, IEEE* 51, no. 2 (2013): 114-119.

[6] Lantz, Bob, Brandon Heller, and Nick McKeown. "A network in a laptop: rapid prototyping for software-defined networks." In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, p. 19. ACM, 2010.

[7] GENI at http://www.geni.net/

[8] VINI at http://www.fp7-federica.eu/pres_eventi/20081014-vini-bavier.pdf.

[9] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau. Large-scale virtualization in the emulab network testbed. In USENIX 2008 Annual Technical Conference, pages 113-128. USENIX, 2008.

[10] Wang, Shie-Yuan, Chih-Liang Chou, and Chun-Ming Yang. "OpenFlow Controllers over EstiNet Network Simulator and Emulator: Functional Validation and Performance Evaluation."

[11] Henderson, Thomas R., Mathieu Lacage, George F. Riley, C. Dowell, and J. B. Kopena. "Network simulations with the ns-3 simulator." SIGCOMM demonstration (2008).

[12] OVS controller at http://yuba.stanford.edu/~casado/of-sw.html.

[13] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown,and S. Shenker. Nox: towards an operating system for networks. ACM SIGCOMM Computer Commun. Review, 38(3):105–110, 2008.

[14] Shalimov, Alexander, Dmitry Zuikov, Daria Zimarina, Vasily Pashkov, and Ruslan Smeliansky. "Advanced study of SDN/OpenFlow controllers." In *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, p. 1. ACM, 2013.

[15] Trema at https://github.com/trema/trema

[16] Kim, Hyojoon, and Nick Feamster. "Improving network management with software defined networking." *Communications Magazine, IEEE* 51, no. 2 (2013): 114-119.

[17] OpenDayLight at http://www.opendaylight.org/

[18] “Jaxon,” accessed 11-June-2013 at http://jaxon.onuos.org/

[19] Erickson, David. "The beacon openflow controller." In Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, pp. 13-18. ACM, 2013.

[20] EugeneNg, ZhengCai AlanL Cox TS. "Maestro: Balancing Fairness, Latency and Throughput in the OpenFlow Control Plane."