# Building a Future in SDN with one Controller

[a]Aparicio Carranza, PhD, [a]Julio Tax and [a]Jose Reyes Alamo, PhD

[a]Computer Engineering Technology
New York City College of Technology of The City University of New York
186 Jay street Brooklyn, NY 11201
acarranza@citytech.cuny.edu,
Julio.Tax@mail.citytech.cuny.edu,
jreyesalamo@citytech.cuny.edu

## ABSTRACT

*There is an approach to computer networking that is changing everything we know about networking it is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications. Software Defined Networking (SDN) is a new approach to networking, complementing traditional network architectures. SDN aims to give the network operator granular programmatic control over network hardware in order to rapidly react to changes in policy, environment, costs, network conditions and other parameters. There are several differences between traditional and SDN networks such as being Symmetric vs Asymmetric, Floodless vs Flood-Based, Host-based vs Network - Centric. Therefore, We will explain how these differences work and how we applied them to our work building an SDN Lab in Virtual Machines using a POX or Floodlight Controller to manage the networks faster without using OpenFlow Hardware.*

**Keywords:** SDN, POX, Floodlight Controller, OpenvSwitch.

## 1. INTRODUCTION

The SDN architecture is directly programmable because it is decoupled from forwarding functions. It is agile, abstracting control from forwarding - this lets administrators dynamically adjust network-wide traffic to meet changing needs. Network intelligence is (*logically*) centralized in software-based SDN controllers that maintain a global view of the network, which appears to applications and policy engines as a single, logical switch [1]. SDN is also programmatically configured, which in turn lets network managers configure, manage, secure, and optimize network resources very quickly via dynamic, automated SDN programs, thus they can write themselves because the programs do not depend on proprietary software [2]. It is also Open standards-based and vendor-neutral which means that when implemented through open standards, SDN simplifies network design and operation because instructions are provided by SDN controllers instead of multiple, vendor-specific devices and protocols. The three main advantages of SDN are the following; Asymmetric, Flood Based and Network Centric [3]. First, in an Asymmetric model, SDN global information is centralized as much as possible, and edge driving is distributed as much as possible. The considerations behind such an approach are clear, centralization makes global consolidation a lot easier. Second, in a Flood-based model, a significant amount of the global information sharing is achieved using well known broadcast and multicast mechanisms, this can help make SDN models more symmetric and it leverages existing transparent bridging principles encapsulated dynamically in order to achieve global awareness and identity learning [3]. Third, in a Host-based model an assumption is made regarding use of SDN in data-centers with lots of Virtual Machines moving to enable elasticity. Under this assumption the SDN encapsulation processing is already done at the host Hypervisor on behalf of the local virtual machines. This design reduces SDN edge traffic pressures and uses "free" processing based on each host spare core capacity.

## 1.1 **OPENVSWITCH**

Open vSwitch sometimes abbreviated to OVS, is a production quality open source implementation of a distributed virtual multilayer switch. The main purpose of Open vSwitch is to provide a switching stack for hardware virtualization environments, while supporting multiple protocols and standards used in computer networks [4 - 5]. As a software implementation of a virtual multilayer network switch - Open vSwitch is designed to enable effective network automation through programmatic extensions, while still supporting standard management interfaces and protocols, including NetFlow, sFlow, SPAN, RSPAN, CLI, LACP and 802.1ag [5]. In addition, Open vSwitch is designed to support distribution across multiple physical servers by enabling creation of cross-server switches in a way that makes the underlying server architecture transparent, similar to VMware's vNetwork distributed vSwitch or Cisco's Nexus 1000v [6 - 8]. Therefore, in this work since Open vSwitch can operate both as a Software-based switch running within the virtual machines (VMs) hypervisors, and as the control stack for switching silicon; it has been ported to multiple virtualization platforms and switching chipsets. Thus, using a vSwitch for our Lab allowed set primitive abstraction for our work. Furthermore, by building our SDN lab, we have  installed OpenVswitch by performing the following steps:

```
$ apt-get purge network-manager
$ apt-get install openvswitch-datapath-source bridge-utils
$ module-assistant auto-install openvswitch-datapath
$ apt-get install openvswitch-brcompat openvswitch-common
```

Then we verified the installation by:

```
$ ovs-vsctl show ovs_version: "1.4.0+build0"
```

## **1.2 FLOODLIGHT (Java) SDN CONTROLLER.**

Floodlight java works with physical and virtual switches that speak the OpenFlow protocol. It is Apache licensed which means we can use floodlight for almost any purpose, because it is also open source. In addition floodlight is the core of a commercial controller product from Big Switch Networks and is actively tested and improved by a community of professional developers. It specifies a protocol through a well-defined "f*orwarding instruction set*". Floodlight is designed to work with the growing number of switches, routes, virtual switches, and access points that support the OpenFlow standard [10]. It offers a module loading system that make it simple to extend its capabilities to be enhanced. It also supports the  openStack cloud orchestration platform. Therefore, because of its flexibility to set up with minimal dependencies, we installed floodlight as our first controller for this work, by performing the following steps:

First, Install dependencies, ***apt-get*** for Ubuntu (UB) and ***yum*** for Red Hat (RH).

```
$ apt-get install build-essential default-jdk ant python-dev eclipse git
```

Second, Clone the Github project and build the jar and start the controller

```
$ git clone git://github.com/floodlight/floodlight.git
```

Third, cd into the floodlight directory just created.

```
$ cd floodlight
```

Fourth, Run ant to build a jar. It will be in the ***~/floodlight/target*** directory.

```
$ ant
```

Fifth, run the controller

```
$ java -jar target/floodlight.jar
```

By default it will bind to port 6633 and all ports e.g. 0.0.0.0/0.0.0.0:6633

## 1.3 POX (Python) SDN CONTROLLER.

POX is a networking software platform written in Python. POX started its life as an OpenFlow controller, but it can also function as an OpenFlow switch, and can be useful for writing networking software in general. POX officially requires Python 2.7 and currently communicates with OpenFlow 1.0 switches and includes special support for the Open vSwitch/Nicira extensions. POX can run anywhere it has topology discovery. Also, it's a platform for the rapid development and prototyping of network control software using Python [11]. Meaning, at a very basic level, it is one of the growing number of frameworks for helping to write an OpenFlow controller. As well as being a framework for interacting with OpenFlow switches – POX goes beyond these functions. We are using it as the basis for some of our continuing work to help us building the emerging discipline of software defined networking. We are using it to explore and prototype distribution, SDN debugging, network virtualization, controller design and programming models. The ultimate goal is to develop and archetypal, modern SDN controller. Thus, we installed POX as our second controller for this work by entering the following sequence of steps:

```
$ sudo apt-get install git
$ git clone http://github.com/noxrepo/pox
$ cd pox
$ ./pox.py forwarding.l2_learning
$ NOX&gt
```

By default it will bind to port 6633 and all ports e.g. 0.0.0.0/0.0.0.0:6633

Then we attach OpenVswitch to the controller.

```
$ ovs-vsctl set-controller br-int tcp:192.168.1.208:6633
```

## 2. IMPLEMENTATION

Our goal has been to build an SDN lab to show how this emerging technology is changing the playing field of network computing. We first installed KVM and then integrated into OVS by entering the following information:

```
$ apt-get install kvm uml-utilities
```

We also incorporated two scripts to bring up the KVM tap interfaces into the bridge from the CLI.

***<u>Script #1</u>***

```
#!/bin/sh switch='br-int'
  /sbin/ifconfig $1 0.0.0.0 up
  ovs-vsctl add-port ${switch} $1
```

***<u>Script #2</u>***

```
#!/bin/sh switch='br-int'
/sbin/ifconfig $1 0.0.0.0 down
ovs-vsctl del-port ${switch} $1
```

*Next, we boot these guest virtual machines*

**<u>Host 1</u>**

```
kvm -m 512 -net nic,macaddr=00:00:00:00:cc:10 -net
tap,script=/etc/ovs-ifup,downscript=/etc/ovs-ifdown -cdrom ubuntu-12.04-desktop-amd64.iso
```

**<u>Host 2</u>**

```
kvm -m 512 -net nic,macaddr=00:11:22:CC:CC:10 -net
tap,script=/etc/ovs-ifup,downscript=/etc/ovs-ifdown -cdrom ubuntu-12.04-desktop-amd64.iso
```

**<u>Host 3</u>**

```
kvm -m 512 -net nic,macaddr=22:22:22:00:cc:10 -net
tap,script=/etc/ovs-ifup,downscript=/etc/ovs-ifdown -cdrom ubuntu-12.04-desktop-amd64.iso
```

In each host, the process begins by loading from the ISO file. We just click “Try Ubuntu” when they are booting and just run them from disk – because, all we need are nodes that can test connectivity as we push static flows. If it is a more permanent test lab, we would install them on the disk. Once they are up and running, we assigned the IP addresses to each one of them by clicking in the top left of the Ubuntu window and type in ‘terminal’. Also, using ***ifconfig***, the I*P addresses can statically be assigned.*

Our ultimate goal has been to build a networking lab without networking equipment yet, another plus in the column of open software driven networks. Proofing and prototyping networks today are often done with other technologies such as NETFPGA or expensive vendor manufactured hardware. Since we are beginning to build primitives, APIs and abstraction layers the value of what the software community has done for two decades is becoming obvious to networking. A few of us are working on wrapping our heads around what the current and future state of Software Defined Networks (SDN) will look like. To build or modify modules and applications we need an appropriate lab setting to test our cases. Something we run into quite often is the idea that we need OpenFlow enabled hardware to build this lab. The good news is that we can absolutely put this requirement aside – By making use of vSwitch, it was all we ever need for modeling and prototyping our modules. Once performance benchmarks and metrics are sought, transitioning to hardware starts making sense.

## 3. CONCLUSION AND FUTURE WORK

Software-Defined Networking (SDN) is an emerging architecture that is dynamic, cost effective and manageable - making networking faster by allowing to transfer large amounts of data in a short time frame. Therefore, by doing this work, we decided to use two SDN controllers attached to OpenVswitch to exchange packet data between three virtual machines. We run multiple VM instances, each running Open vSwitch and using multiple bridges to create unique Data Path IDs (DPID) that essentially appear as different hosts. This approach conditions imposed less hardware requirements, but is a bit less flexible from seeing what the host is doing. That said we could spin up dozens of vSwitches under the control of single or separate OpenFlow SDN controllers and instantiate data paths. All we needed to do was to attach each bridge to a controller and now data paths were built by each OpenFlow controller rather than the OVS slow path. There are quite a few controllers out there. We primarily use POX (Python) and Floodlight (Java) since both have the best maintenance community based from what we have been exposed to along with easy modules to use, adjust or spin our own. The documented APIs are coming along as well - Floodlight probably has the slight edge there but we prefered Python so it was a wash for us. POX does not have a monetized product attached to it (BigSwitch) which would be attractive to some folks also. Thus, with both options we can integrate physical hardware which is essentially the SDN road map topology being proposed in data center solutions today, by making connections agile.

## REFERENCES

[1] "Prof. Scott Shenker - Gentle Introduction to Software-Defined Networking - Technion lecture". YouTube. 2012-06-26. Retrieved 2014-01-23.

[2] "Google's software-defined/OpenFlow backbone drives WAN links to 100% utilization". Networkworld.com. 2012-06-07. Retrieved 2014-01-23.

[3] Tom Nolle (2013-04-26). "Three models of SDN explained". Searchcloudprovider.techtarget.com. Retrieved 2014-01-23.

[4] "Complete list of Open vSwitch releases". openvswitch.org. Retrieved 2014-04-02.

[5] Thomas Graf (2013-04-24). "Underneath OpenStack Quantum: Software Defined Networking with Open vSwitch"(PDF). Red Hat. Retrieved 2014-04-09.

[6] Ralf Spenneberg. "Virtual switching with Open vSwitch". admin-magazine.com. Retrieved 2014-04-02.

[7] "ovs/INSTALL.NetBSD at master". *openvswitch/ovs*. github.com. 2014-01-11. Retrieved 2014-04-09

[8] Jesse Gross (September 2013). "Programmable Networking with Open vSwitch" (PDF). LinuxCon. Retrieved 2013-11-24.

[9] "FreshPorts – net/openvswitch". freshports.org. 2013-12-30. Retrieved 2014-04-02

[10] SDN Java Floodlight controller. 2014, Web. http://www.projectfloodlight.org/floodlight/

[11] POX Python SDN controller. 2014. Web https://github.com/noxrepo/pox