CAPSTONE PROJECT: MINT 2017-2019

SDN: Controller Comparison with Implementation;

Security Concerns and Vulnerabilities

Under the guidance of:

Prof. Leonard Rogers

Submitted By:

Shveta Rai

MINT- 709

ABSTRACT

In this Capstone Project, I aimed at implementing a basic SDN structure with Mininet and two controllers i.e. POX and Floodlight, learning over the period, how it is an advantage over traditional network implementation and analyzing how the architecture of SDN is an enhancement.

After implementation, I learned various short-comings and challenges of SDN, exploring their effects, concerns and their solutions, some of them namely:

1. **OpenFlow and related concerns**: One of the main concerns pertaining OF protocol I would incorporate is the scalability (TCP SYN flooding) and fault-tolerance issues in the design. Another issue would the communication bottleneck between data and control plane which can be manipulated in attacks related to traffic and how it is currently dealt with and what possible solutions could be there in the future.

2. **Controller Vulnerabilities**: Controller/Network Hypervisor is the first step in a virtualized SDN environment and a security consideration as well since it defines data flow that occurs in the Data Plane. Prominent weakness of the controller that I would like to study, analyze its possible solutions and how they are currently used in the industry.

3. **Large network vulnerabilities like DDoS attack**: Since SDN consists of three layers, namely infrastructure layer, control layer and application layer, the potential DDoS attack can be launched on any of the three. During the Capstone Project, I would research and analyze the threats via the mentioned mediums and their defense and mitigation approaches currently used and possible future solutions.

Via this Capstone Project, I have gained knowledge about SDN, its implementation, the security vectors involved with addition to what possible concerns, shortcomings and challenges this highly anticipated technology faces with the wide spread implementation.

ACKNOWLEDGMENT

It has been a great learning experience, this Capstone Project, not only in terms of knowledge but devoting time and following the decided schedule with dedication.

For the experience, I want to thank the Program Coordinator for introducing Capstone Project as a course in the MINT program. Also, a huge thank you to Prof. Mike Macgregor who has been a great force behind MINT as a program.

Prof. Leonard Rogers has been instrumental not only in MINT 712, teaching us beyond the quintessential aspects of security but answering our questions concerning our projects and otherwise with great depth and zeal.

I'm indebted to my parents for giving me the push to pursue Masters in Canada, far away from my home in India. And last but not the least my friends **Kunjal Pundeer**, for ensuring I was keeping up with my set deadlines and **Vaibhavi Kadam**, for accompanying me in my study sessions.

INDEX

Table of Contents

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

In today's expansively digital inter-connected world gathering information isn't a herculean task. To further extend on it, we have a burst of mobile devices, cloud, data center amongst many others ensuring the transition, storage and accessibility of services and data is as feasible as it could be.

Starting with the traditional Client-Server architecture which comprised of two or more clients accessing the services the server was programmed to allow, we, today, can work with masses of cross-server platforms and fetch whatever required at the ease of our homes.

But this intensive east-west machine-to-machine traffic needs only the requisites but also the quality of all aspects of communications.

In this capstone project, I have put in efforts to learn where and how the traditional network architecture lacks and how SDN covers up the shortcomings of the former, while having its own vulnerabilities. In the process, I also go through basic components of SDN, trying to implement Mininet based SDN environment using two controllers namely, Floodlight and POX.

In the end, I venture into the vulnerabilities of SDN and what measures are currently in place to tackle the issues and what prospects the future holds for Software Defined- Networking.

1.1 TRADITIONAL NETWORK INFRASTRUCTURE

In traditional network infrastructure, the control plane and data plane were both integrated together, making it a dedicated appliance, be it within a switch/router etc. Ethernet switches are the best example of a traditional unit consisting of a combined control and data plane i.e. a dedicated hardware.

The ports serve the inbound and outbound traffic and then the controller inbuilt processes the control logic to forward the packets to their destination. The ARP table has the MAC addresses mapped to corresponding ports. Based on the functionality needed, an Ethernet switch can perform MAC filtering, device monitoring etc.

Traditional networks, although feasible, are tedious and complex to handle the enormous inflow and outflow of data generated today. Manual configuring of networks, setting and defining policies, ensuring proper routes and over-seeing the security of each route becomes a nightmare for the network administration. Add on top, the number of devices connected to the internet seeking information and manual configuration of each individual component, being as accurate as vendor specific commands.

In addition to complexity, the traditional environment also must endure the faults and ensure proper dynamic nature to adjust to load at any time. Authentication, access control lists, VLANs, firewall rules, topologies and QoS are some other part of the same problem arising while handling changes in the daily traffic. And since automation in traditional setup is almost non-existent, this makes the entire process even more challenging.

One of the main characteristics of this architecture was that the dedicated hardware is mostly proprietary and needs to be configured individually and made compatible with each other. This can be utterly time consuming and frustrating, in addition to the probability of being error prone. Also, the evolution of the involved appliances could be slow since it is under the control of the manufacturer.

Explicitly speaking about distributed control and transport layer network protocols, the routers and switches are vertically integrated i.e. data plane and control plane are integrated within the devices which further reduces flexibility when it comes to introducing new abstractions to the evolution of the infrastructure, making the current implementation static.

1.2 MOTIVATION

For the above stated reasons and more, there is a need of an element of dynamic nature for automation.

Elastic computing comes to the rescue. Virtualisation, being one of the most successful enablers of the same, ensures connectivity amongst distributed nodes providing differentiated QoS for plethora of applications, keeping provisioning of resources dynamic.

But it still requires manual labour to converge multiple networks, which results in inability to dynamically adapt to changes in application traffic and user requirements.

One of the possible solutions currently in consideration is Software Defined Networking (SDN) which aims at manipulating the basic structure of networks to overcome some of the previously mentioned limitations.

2. SOFTWARE DEFINED NETWORKING

Software Defined Networking is an emerging architectural approach aiming to put in automated processing of the network by segregating its control logic i.e. control plane from the underlying hardware (routers and switches) forwarding the traffic (data plane), thus more closely binding the interaction amongst applications and network devices with the services being offered.

It centralises the logic decision taking component in the architecture. It is a set of manageable networks hatched together with virtualisation being one of the prime foundations, at present.

According to one of IEEE's paper, SDN could be defined as a network architecture with the following four pillars:

1) A network architecture where the control and data planes are decoupled thus removing the control functionality from network devices which now will simply become forwarding elements.

2) Forwarding decisions are flow-based, instead of destination-based.

3) Control logic is moved to an external entity called SDN controller or Network Operating System (NOS).

4) The network is programmable through software applications running on top of the NOS that interacts with the underlying data plane devices.

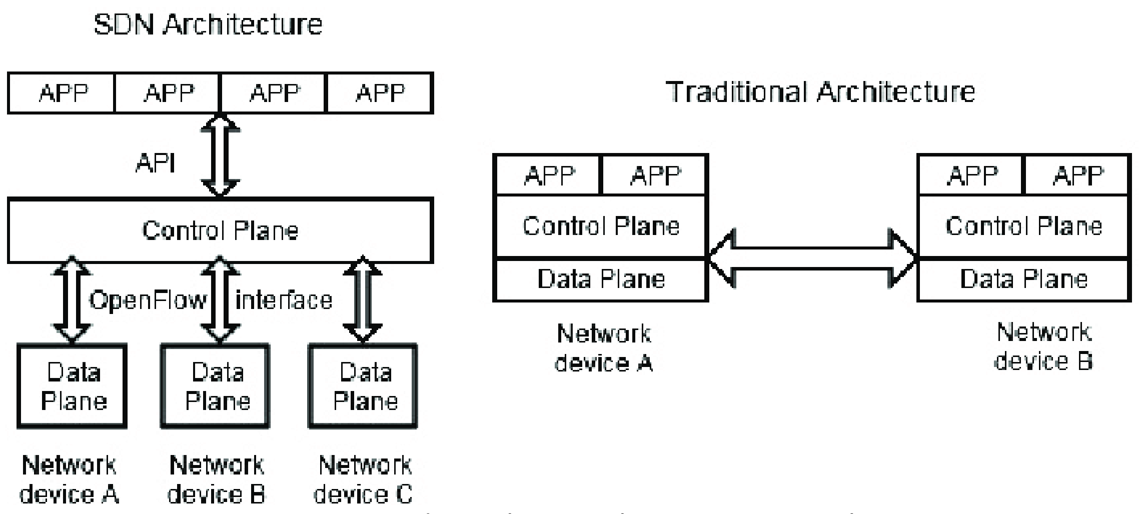


Figure 1: Traditional Networks vs SDN Networks

2.1 VIRTUALISATION AS AN ENABLER

Virtualisation is the logical abstraction of physical assets i.e. translating hardware into either firmware or software or emulated software-based objects. It is one of the key enablers for technologies like Cloud Computing, Network Function Virtualisation and even Software Defined-Network.

It multiplexes the physical interface and creates multiple virtual objects from that interface, with aggregation, it can create one virtual object from multiple physical objects and with emulation, it can create a virtual object from different types of physical entities.

a hypervisor is the main entity responsible for the instantiation of virtual objects. It is a small specialised operating system running on physical server that partitions and provisions physical resources as virtual resources. It is also responsible for maintaining isolation amongst all instances.

The main benefit of virtualisation is the maximum utilisation of resources, in addition to cost reduction. It allows for elastic and scalable resource provisioning while sharing amongst many users. Multi-tenacity is one of the biggest benefits of virtualisation.

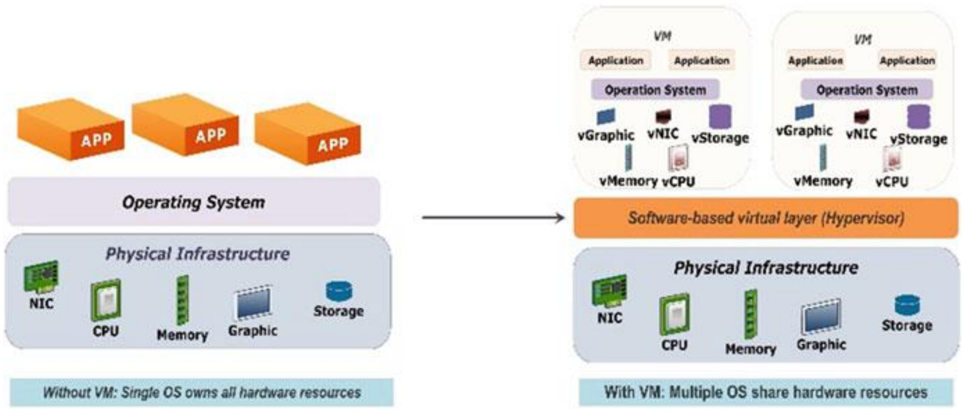


Figure 2: Virtualisation

2.2 <u>SDN ARCHITECTURE</u>

The SDN architecture can be divided into:

<u>Control plane</u>: It deals with the functionalities of changes in topology and service provisioning, amongst others. It establishes a local data set, also called ROUTING INFORMATION BASE (RIB) to form forwarding table entries which stores network topology and is used by the Data Plane to direct inbound and outbound traffic. It has direct control over the network's data plane via APIs such as OpenFlow.

The Control Plane has another component in the form of a Management Plane dealing with functions such as monitoring, configuring and management service provisioning to layers of network stack and the rest of the system.

Benefits:

- ✓ Security measures are put on top of the controller and this makes it easy to dynamically add/cut off devices at various places in the network thus helping in effective network monitoring.
- ✓ Also, since we can keep an eye on the status of devices, any device susceptible to attack can be filtered and removed in the initial stages itself. Example: A DDoS attack can be detected and mitigated quickly by isolating the outbound and inbound traffic of the malfunctioned device.

<u>Infrastructure plane</u>: Like in traditional networks, it is comprised of networking equipment. The main difference is the equipment are just forwarding devices, without any embedded control or ability to take in decisions as the logic intelligence is separated to form the controller.

It includes the **Data Plane**. The Data Plane, also called the "Forwarding Plane", deals with the forwarding of network user traffic, built on the rules put in the system by the control plane. It handles the incoming traffic and performs basic checks on the receiving packets such as packet measurement, packet filtering, packet buffering amongst many.

Authenticated datagrams are then processed by lookups in the FIB table which is formed from a well- established and stable RIB table in the control plane. In addition to FIB tables, the Data Plane also implements certain small services such as Access Control Lists, QoS and policies

<u>Application Plane</u>: It has the network behavior definition in the form of applications and services offered. It hosts SDN applications while communicating with the controller through APIs on the Northbound interface.

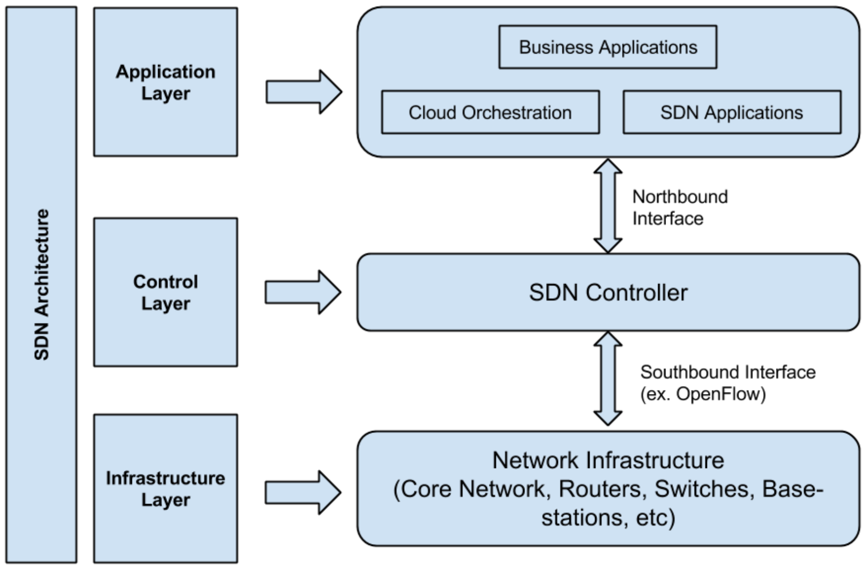


Figure 3: SDN Architecture

2.3 SDN COMPONENTS

Components of SDN based infrastructure would include:

- ✓ **Forwarding Devices**: These are the data plane network devices capable of receiving and sending data packets on its ports and they could be switches, routers and even firewalls. These switches can be hardware(physical), software or virtual. The main function of switches in SDN is to forward and process data.
- ✓ **Controller**: Controller is a logical entity in the infrastructure that is responsible for receiving instructions/pre-requisites from the application layer and further relays them to the underlayer networking components. In remains in contact with the infrastructure via the Southbound APIs, adding/updating or deleting flow entries and with the application layer via the Northbound APIs.
- ✓ **Southbound Interface**: Southbound APIs help in facilitating dynamic control over the network to meet the real-time demands. OPENFLOW is one of the most well-known southbound interface and industry standard defining the interaction of SDN controller with the Data plane. Other southbound APIs include Lisp and NetConf, amongst others.
- ✓ **Northbound Interface**: It is the mode of communication between application layer and the control layer. Based on the developer, it could be implemented in languages like Python, java and C++. Currently, there is no standard protocol that exists for the Northbound APIs. It is used to develop vendor independent applications, for load balancing and monitoring of the applications implemented.
- ✓ **Network Applications**: These are programs and applications communicating with SDN via APIs to provide services to the end user.
- ✓ **OpenFlow**: Industry standard for Southbound APIs, it defines the communication between switching hardware and network controller. The OpenFlow switches are basic forwarding elements accessible by the OpenFlow protocol and interface. These switches consist of one or more flow tables, having header fields, that performs packet lookups and forwarding and can perform as either a router, switch, firewall or

other roles as instructed by the controller. So, when the packet arrives, the header is extracted to match it with the table and if found corresponding action is taken.

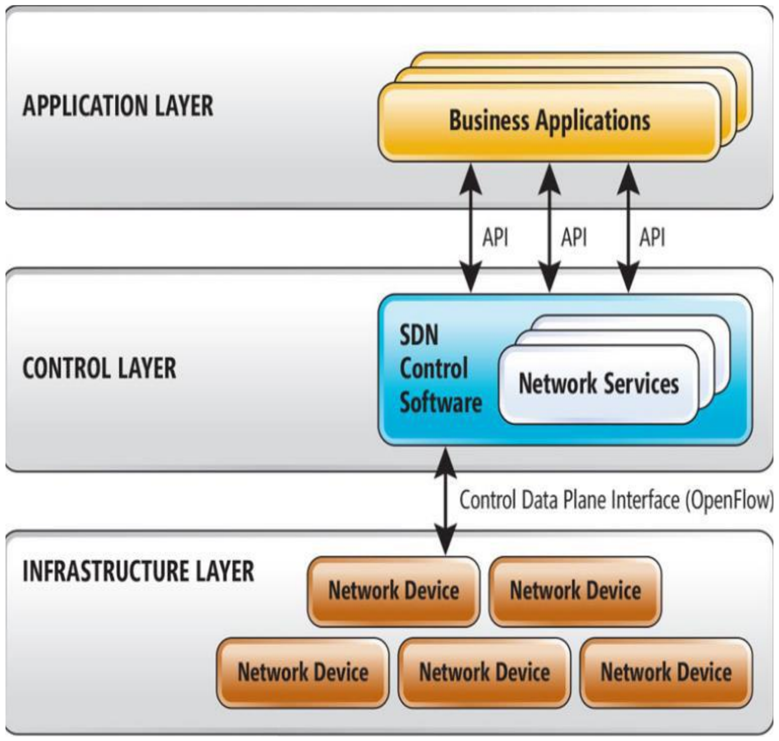


Figure 4: SDN Planes

2.4 ADVANTAGES OF SDN

a. **Directly Programmable**: It is based on active networking which brings in programmable functions in the network thus lowering the barrier to innovation. One can implement new networking protocols in something as simple as a virtual machine.

b. **Accessibility**: It is vendor neutral since it is open standards-based. The network managers can use features of SDN by writing their own programs that aren't based on proprietary standards.

c. **Centrally managed**: Having a logical centralised control plane has benefits including:

   ✓ Scalability and dynamic volatility of network is well adjusted and supported with each managed device.

  - ✓ High availability is ensured.
  - ✓ Geographically speaking, since the control plane is logical, it is easier to manage.

d. **OPEX**: Operational efficiency and reduction in cost.

e. **Agile**: The logical centralization has several benefits including:

  (i) Simpler and less error-prone to modifications

  (ii) A control program can react to changes in the network state (traffic etc.) and maintain the policies, working it all dynamically.

  (iii) Simplification of development of sophisticated network services, applications and functions.

  (iv) It brings in Network Virtualisation and thus the ability to demultiplex software programs based on packet headers.

2.5 OPPORTUNITIES FOR SDN

- ✓ To support and enhance the movement for dynamic networks and ease the replication and virtual resource allocation.
- ✓ To ease the administration responsibilities for the configuration and functionality provisioning, involving something as tricky and sensitive as security with more effectiveness.
- ✓ To bring in the concept of easy and scalable network deployment and functioning.
- ✓ To utilise network resources in a better way.
- ✓ To significantly reduce Operational Expenditure (OPEX) and complexity.
- ✓ Enabling user applications for dynamic service requesting from the network.

2.6 CHALLENGES FOR SDN

- ✓ **Addressing dynamic change with accuracy**: SDN, even though can automate provisioning of newly converged network in less time but need of the hour is a performance monitoring solution (with open APIs) which can enhance integration as they can listen on event bus, look for new devices and instantly do the needful changes.

- ✓ **Addressing rapid on-demand growth:** Rapid increase in connected devices can pose a risk to monitoring platforms in current SDN scenario. SDN needs extra performance management capacity which can help spin up additional virtual appliances as the demand increases, not affecting the performance or resource allocation.
- ✓ **Security:** SDN being different than traditional network, brings along plethora of new vulnerabilities. Moreover, SDN still being in R&D phase has many loopholes to be considered before its wide spread implementation.

3. OPENFLOW

Considered one of the first SDN standards, it is a communication protocol mostly used as a southbound API, enabling SDN controller to directly interact with the Data plane. The Controller uses this interface to implement changes to the FIB (Forwarding Information Base) to efficiently manage the traffic, implement new rules and or control flows for optimal performance.

It was originally started to allow the creation and testing of experimental protocols for research purposes at the Stanford University. According to Open Networking Foundation (ONF), OpenFlow provides network programmability from a centralised view.

As a set of protocols and an API, it is divided into two parts, namely: Wire Protocol and Configuration & Management protocol.

Wire Protocol: Is used for establishing a control session, for defining a message structure used to exchange flow-mods and collect statistics while defining the fundamental structure of a switch.

Configuration and Management Protocol: Is used to allocate physical switch ports, define high availability and response on controller failure.

The OpenFlow API handles L2-L4 network flow but to handle L5-L7 flow, it was extended.

Open vSwitches are one of the switches the OpenFlow protocol can use and the tables in the switch consists of **header fields, counters and actions**. Header fields are matched with the FIB and if a match is found, the counter is updated, and the concerned action is taken. Else, a PACKET-IN message is sent to the controller over a TLS secured channel to notify about the packet.

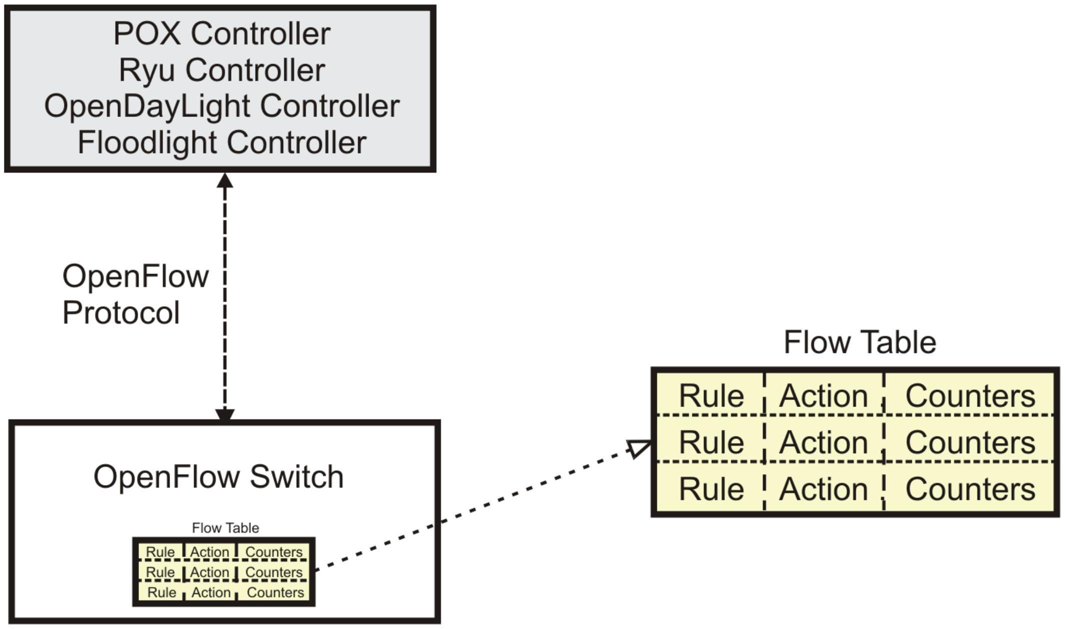


Figure 5: OpenFlow in SDN

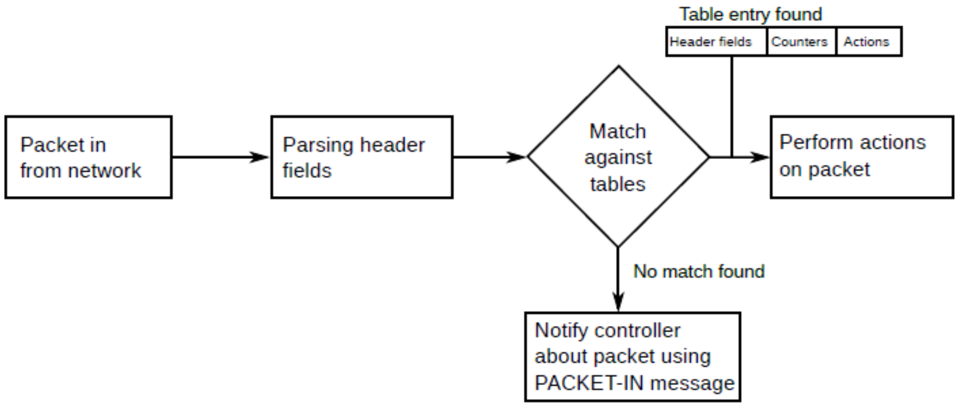


Figure 6: Packet Forwarding in Open vSwitch in OpenFlow

Benefits of OpenFlow include:

- It is an enabler for innovation and accelerates new features and services.
- It has simplified provisioning, performance optimising.
- It has helped in abstraction by decoupling control and data planes.

3.1 <u>OpenFlow Architecture</u>

The OpenFlow protocol has four interconnected layers, namely: message layer, state machine, system interface and configuration.

- **Message Layer**: It is used to define the semantics and syntax of messages shared and supports manipulating messages to get the desired outputs.
- **State Machine**: It works at core low-level to attain actions as negotiations, flow control, delivery etc.
- **System Interface**: It sets up instructions defining how OpenFlow identifies interfaces, enabling interaction with the outside environment.
- **Configuration**: It ensures the configuration aspect such as buffer sizes, reply intervals etc.

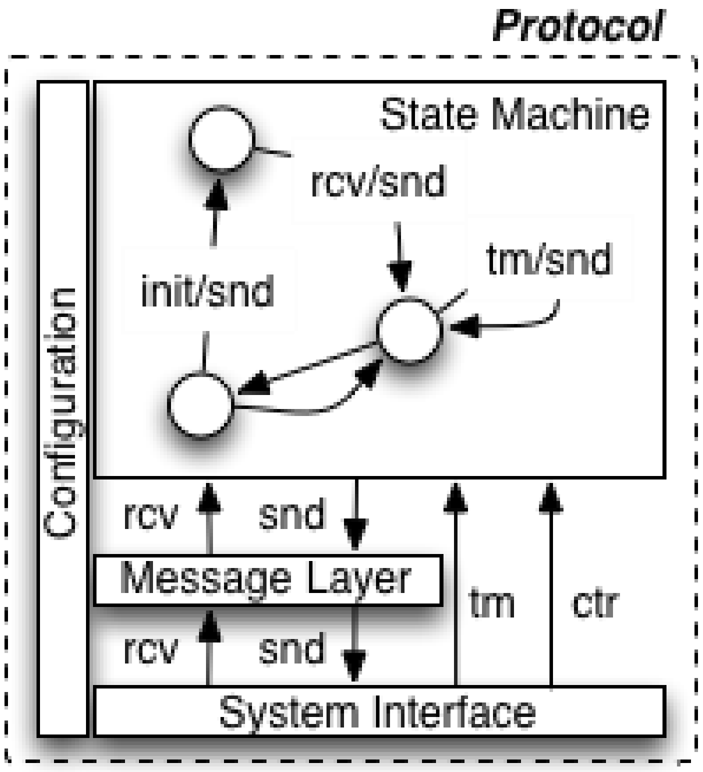


Figure 7: OpenFlow Layers

3.2 OpenFlow Messages

The messages exchanged by the OpenFlow implementing controller and OpenFlow switch can be of three types:

- Symmetric messages: Bidirectional; sent without solicitation.
- Asynchronous messages: Sent via switch without controller asking them.
- Controller messages: Initiated by controller to control/view switch's state.

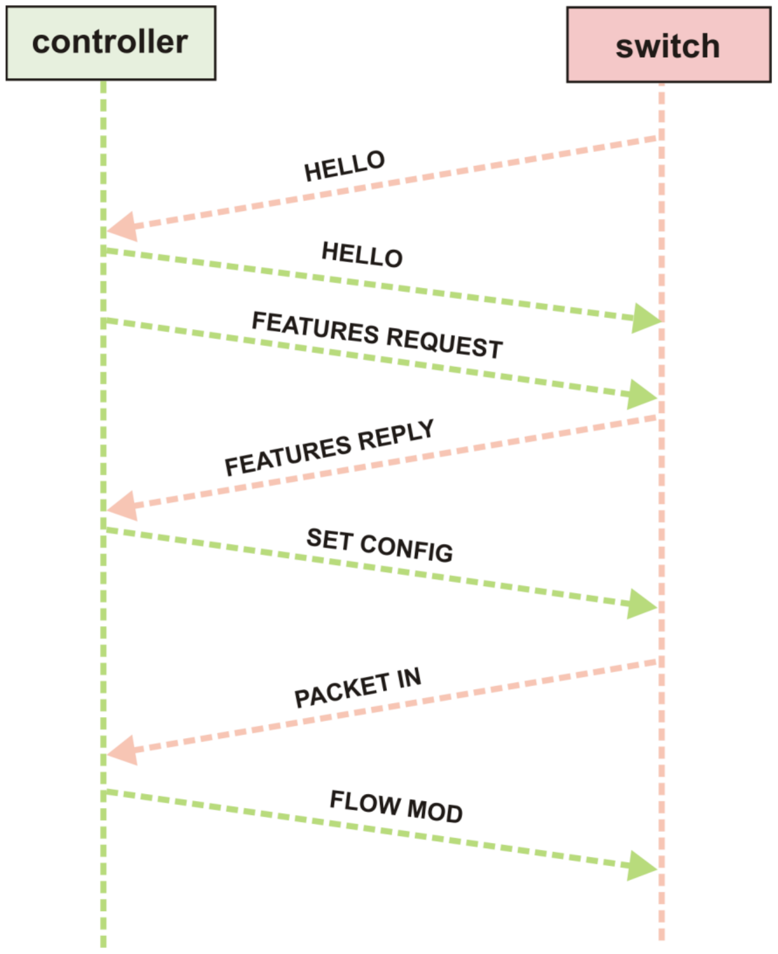


Figure 8: Flow of Messages in OpenFlow

- Hello (Controller -> Switch): after the TCP handshake, controller sends its version number to switch.
- Hello (Switch -> Controller): switch replies to previous Hello message with its supported version number.
- Features Request (Controller -> Switch): Controller requests for available ports.

- Features Reply (Switch -> Controller): Switch replies with the list of available ports, port speeds and supported actions and tables.
- Set Config (Controller -> Switch): Controller asks the switch to send flow expirations.
- Packet-In (Switch -> Controller): received packet didn’t match any entry in switch’s flow table and hence was sent to the controller.
- Flow-Mod (Controller -> Switch): controller instructs the switch to add the packet entry to the flow table.

4. Open vSwitch

Often abbreviated as OVS, Open vSwitch is an open-source virtual multilayer switch. For Virtualisation environments, it provides a switching stack whilst supporting multiple network protocols and standards. They sit below the OpenFlow interface. It abstracts out underlying server architecture and allows creation of cross-server switches, thus enabling transparent distribution across multiple platforms.

Written majorly in platform independent C, it has a Linux kernel implementation, providing easy portability to multiple environments.

Ovs-ofctl is a utility tagged along with Open vSwitch allowing to monitor and control a single switch's flow table, proving helpful for the debugging process. It helps examining OVS's kernel flow cache, which is a subset of full OpenFlow flow table.

Characteristics of Open vSwitch:

- **Mobility**: Open vSwitch supports both configuration and migration of slow as well as fast network state between VM instances.
- **Network Dynamics**: Open vSwitch supports features which allow network control system to adapt as quickly as the environment changes. Open vSwitch Database (OVSDB) is one of the many features that supports remote triggers.
- **Maintaining Logical tags**: Tags are useful to uniquely identify VMs or hold some relevant context in the logical domain. And Open vSwitch has multiple methods to specify and maintain tagging rules, making it accessible to VMs at demand.
- **Hardware integration**: Open vSwitch is capable of offloading packet processing to hardware chipsets which allows it to be able to control both hardware and software switch and compatibility.

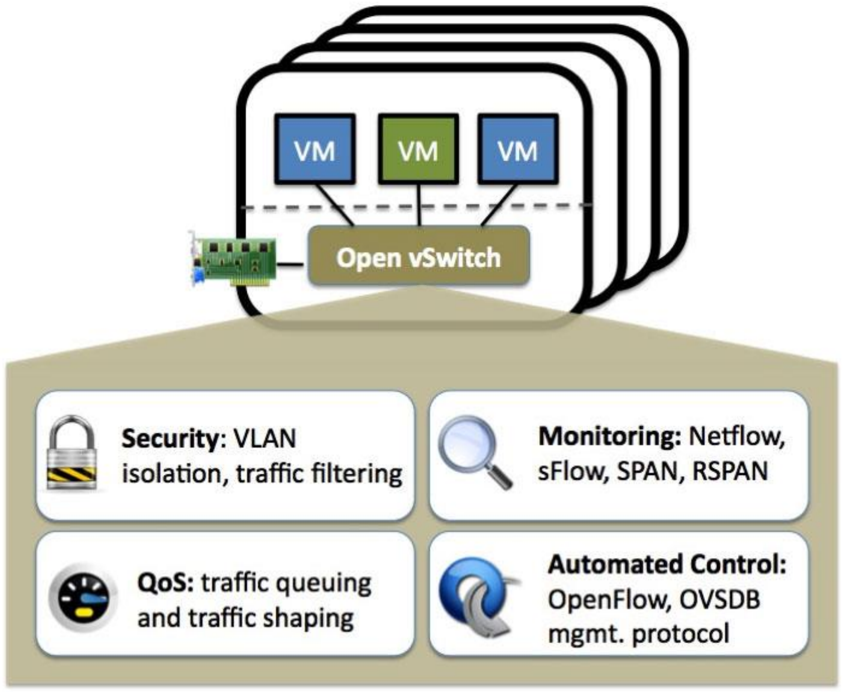

VM
VM
VM
Open vSwitch
Security: VLAN isolation, traffic filtering
Monitoring: Netflow, sFlow, SPAN, RSPAN
QoS: traffic queuing and traffic shaping
Automated Control: OpenFlow, OVSDB mgmt. protocol

Figure 9: Open vSwitch: Features

5. MININET

5.1 INTRODUCTION

Since SDN is the R&D phase, emulation tools are of utmost help while recreating deployment scenarios, making it possible to have different performance metrics evaluated but without the financial constraint and the complexity of an actual deployment.

Simulators, an alternative, could model a scenario on software to have performance evaluated, being as close as possible to actual implementation, in addition to having full knowledge of all factors, on and off stage, involved.

Mininet, a single Linux kernel-based system, is one of the network emulation orchestration systems widely used for research on SDNs. It creates a realistic virtual network instantly, on a single machine with a real kernel, switch and application running code.

The process-based lightweight virtualisation and network namespaces helps Mininet create virtual networks easily and is the reason why the hosts have their own private network interface and can only see their own processes due to isolation provided by virtualisation instances.

The switches in Mininet are software based Open vSwitches or the OpenFlow reference switches, giving a real feel of the OpenFlow protocol to the user. The links are live in Linux and connect the emulated switches to the emulated processes that work as hosts.

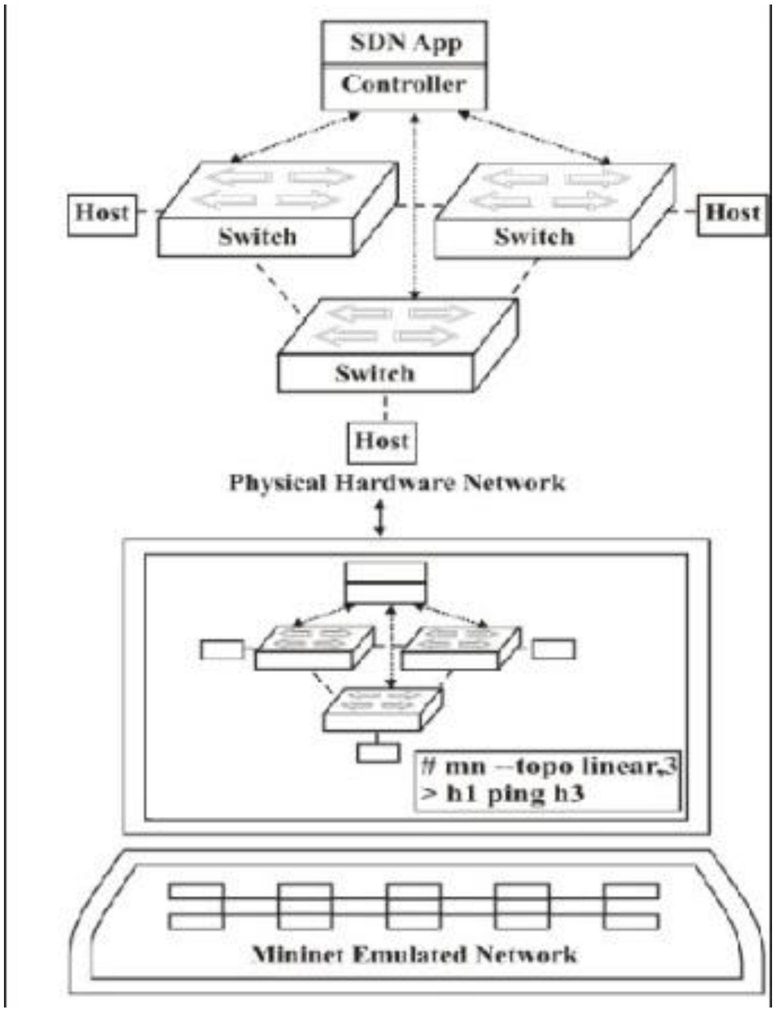


Figure 10: Mininet Emulated Network

5.2 COMPONENTS

- ✓ **Isolated Hosts**: The user-level processes(hosts) in Linux kernel are moved into a network namespace and since process groups have exclusive ownership of components like ports, interfaces, routing tables etc., each host is isolated.
- ✓ **Emulated Links**: Linux Traffic Control manages the data rate of each link in the emulated network. A virtual ethernet acts like a connecting wire for two or more virtual interfaces.
- ✓ **Emulated Switches**: Mininet amongst available resources, uses either default Linux bridges or Open vSwitches which are responsible for handling inbound and outbound

traffic through the interfaces available to Mininet, whether on VM or host.

5.3 FEATURES

- ✓ It is a command-line instantiated network platform where we can run real programs.
- ✓ Being a python API, we can easily create networks of varying sizes and topologies and the processing is fast.
- ✓ Open source and supported and developed by BSD Open Source License.

5.4 LIMITATIONS

- ✓ Resource limitations as when we run multiple instances on a single machine, the resources need to be balanced and shared amongst the hosts the switches (virtual).
- ✓ Mininet is a single Linux kernel and hence software vendor limitation can arise.
- ✓ Mininet doesn’t come with a pre-written OpenFlow controller. One must develop their own controller with the custom features.
- ✓ By default, Mininet network is isolated from the LAN and from internet but we can use NAT options to connect them to the internet or LAN.

6. SETUP MININET

Mininet can be installed either on an Ubuntu machine, from scratch or we can import the already built Mininet VM from the Mininet GitHub.

6.1 MININET FROM GIT

Install Ubuntu ISO on a VirtualBox Machine and update the system using:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get dist-upgrade
```

To get the Mininet source code from Git, we install Git:

```
$ sudo apt-get install git
```

Install Mininet from source code:

```
$ sudo git clone git://github.com/mininet/mininet
```

We can tag all released versions of git and choose whichever version we want to install. In our case, we have installed 2.2.0b3 version.

```
$sudo git checkout -b 2.2.0b3
```

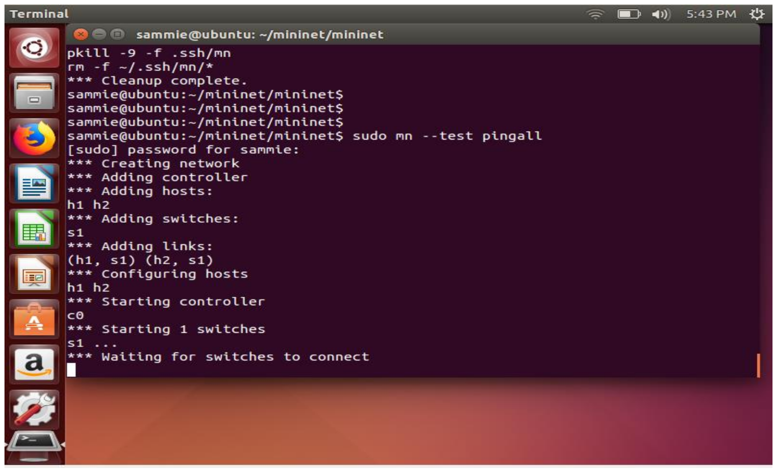
Mininet project has an install script, so we run the script that would install Mininet 2.2

```
$ ~/mininet/util/install.sh -a
```

After the script runs completely, we can test the if the installation was successful. So, we run:

```
$ sudo mn –test pingall
```

Mininet is up and running.



```
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
sammie@ubuntu:~/mininet/mininet$
sammie@ubuntu:~/mininet/mininet$
sammie@ubuntu:~/mininet/mininet$
sammie@ubuntu:~/mininet/mininet$ sudo mn --test pingall
[sudo] password for sammie:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
```

Figure 11: Starting Mininet

7. SDN CONTROLLERS

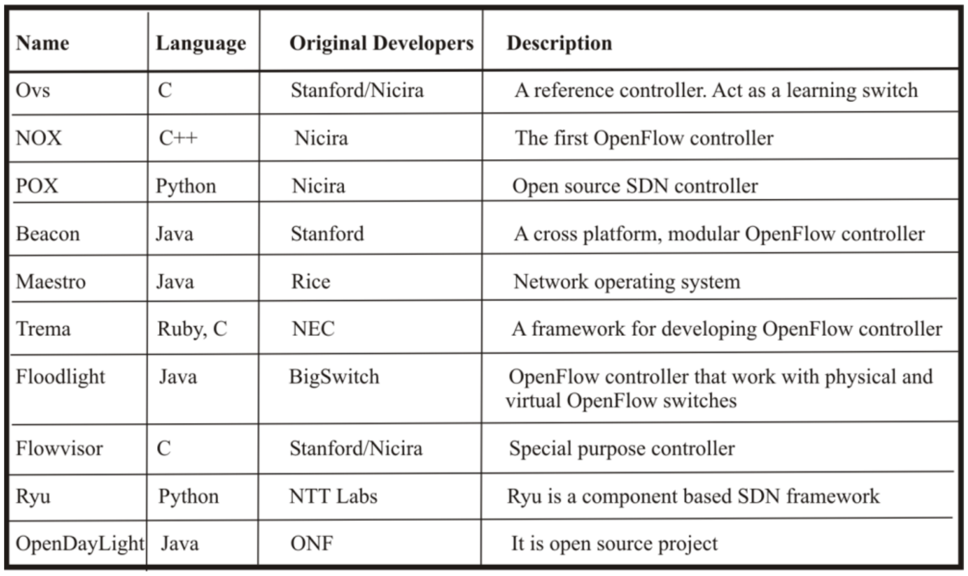
7.1 INTRODUCTION

SDN controllers are the main brain of the architecture. They are master behind configuring network devices and instructing switches, via Southbound Interfaces, of their actions and feeds information to the application through the northbound interface.

Controller can be viewed as the centralized management in Software defined-networking. It handles the flow tables in a switch. One of the two things can happen with a packet i.e. it's either in the flow table which can then proceed with the action associated with it or it isn't in the flow table which then controller can instruct the switch to add an entry via flow mod option.

7.2 ATTRIBUTES OF CONTROLLER

- ✓ Centralised management and distribution of the network state to the switches and network devices connected alongside.
- ✓ It is a high-level data model providing a top-level view of relation between resources, policies and other services offered.
- ✓ They can support further scalability via multi-controller environment.



Name	Language	Original Developers	Description
Ovs	C	Stanford/Nicira	A reference controller. Act as a learning switch
NOX	C++	Nicira	The first OpenFlow controller
POX	Python	Nicira	Open source SDN controller
Beacon	Java	Stanford	A cross platform, modular OpenFlow controller
Maestro	Java	Rice	Network operating system
Trema	Ruby, C	NEC	A framework for developing OpenFlow controller
Floodlight	Java	BigSwitch	OpenFlow controller that work with physical and virtual OpenFlow switches
Flowvisor	C	Stanford/Nicira	Special purpose controller
Ryu	Python	NTT Labs	Ryu is a component based SDN framework
OpenDayLight	Java	ONF	It is open source project

Figure 12: SDN Controllers

7.3 POX: Introduction

POX is an OpenFlow controller which now is also extending to function as an OpenFlow switch. A networking software platform, it is written in Python and can run on any platform that has Python 2.7 and above installed on it.

The booting process for POX requires pox.py file that takes a list of module names on command line, locates those modules and calls their launch function to put them through their up state for functioning.

Features of POX:

- ✓ It is a python based OpenFlow interface.
- ✓ It has reusable sample components for many services like path selection, topology discovery etc.
- ✓ It works on major platforms like Windows, Linux, Mac OS.

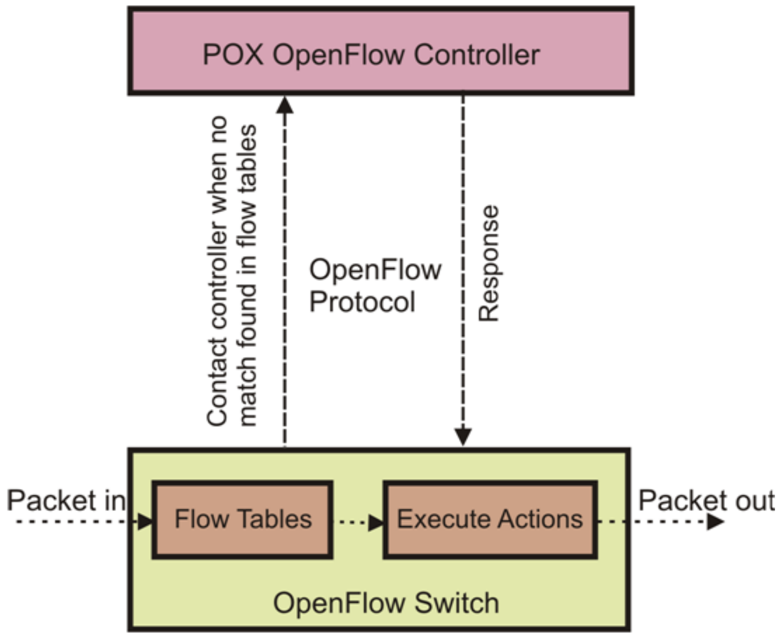


Figure 13: POX Controller Functioning

7.3.1 SETUP FROM GIT

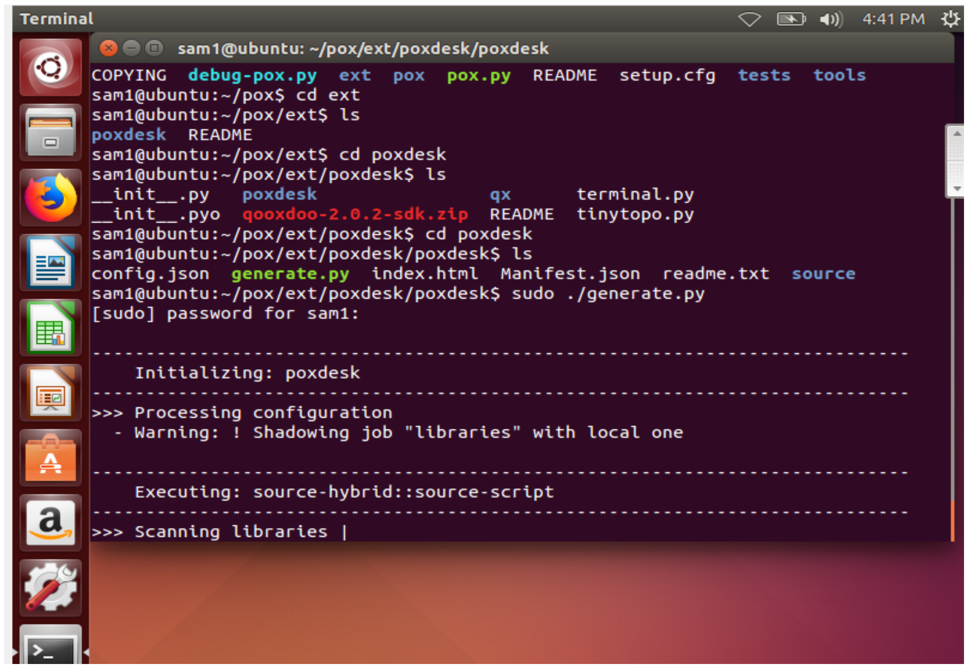
Generally, POX comes installed along with Mininet, but we can also install POX on another Host.

We first pull the POX repository

```
$ sudo git clone https://github.com/noxrepo/pox
```

Then we install POXDesk which is a web-based GUI for POX and makes it convenient to monitor the switches and the network. Steps for the same are:

```
$ cd pox
$ sudo git checkout betta (we're choosing to branch out to betta)
$ cd ext
$ sudo git clone https://github.com/MurphyMc/poxdesk
$ cd poxdesk
$ sudo wget http://downloads.sourceforge.net/qooxdoo-2.0.2 sdk.zip
$ sudo unzip qooxdoo-2.0.2-sdk qx
$ cd poxdesk
$ sudo ./generate.py
```



```
COPYING  debug-pox.py  ext  pox  pox.py  README  setup.cfg  tests  tools
sam1@ubuntu:~/pox$ cd ext
sam1@ubuntu:~/pox/ext$ ls
poxdesk  README
sam1@ubuntu:~/pox/ext$ cd poxdesk
sam1@ubuntu:~/pox/ext/poxdesk$ ls
__init__.py   poxdesk                qx      terminal.py
__init__.pyo  qooxdoo-2.0.2-sdk.zip  README  tinytopo.py
sam1@ubuntu:~/pox/ext/poxdesk$ cd poxdesk
sam1@ubuntu:~/pox/ext/poxdesk/poxdesk$ ls
config.json  generate.py  index.html  Manifest.json  readme.txt  source
sam1@ubuntu:~/pox/ext/poxdesk/poxdesk$ sudo ./generate.py
[sudo] password for sam1:

----------------------------------------------------------------------------
    Initializing: poxdesk
----------------------------------------------------------------------------
>>> Processing configuration
  - Warning: ! Shadowing job "libraries" with local one

----------------------------------------------------------------------------
    Executing: source-hybrid::source-script
----------------------------------------------------------------------------
>>> Scanning libraries |
```

Figure 14: Installing POX

```
$ cd ../ ../ ../
$ sudo ./pox.py samples.pretty_log web messenger messenger.log_service
messenger.ajax_trasnsport openflow.of_service poxdesk
```

Now we can access POX controller on POXDesk at http://127.0.0.1:8000/poxdesk/source

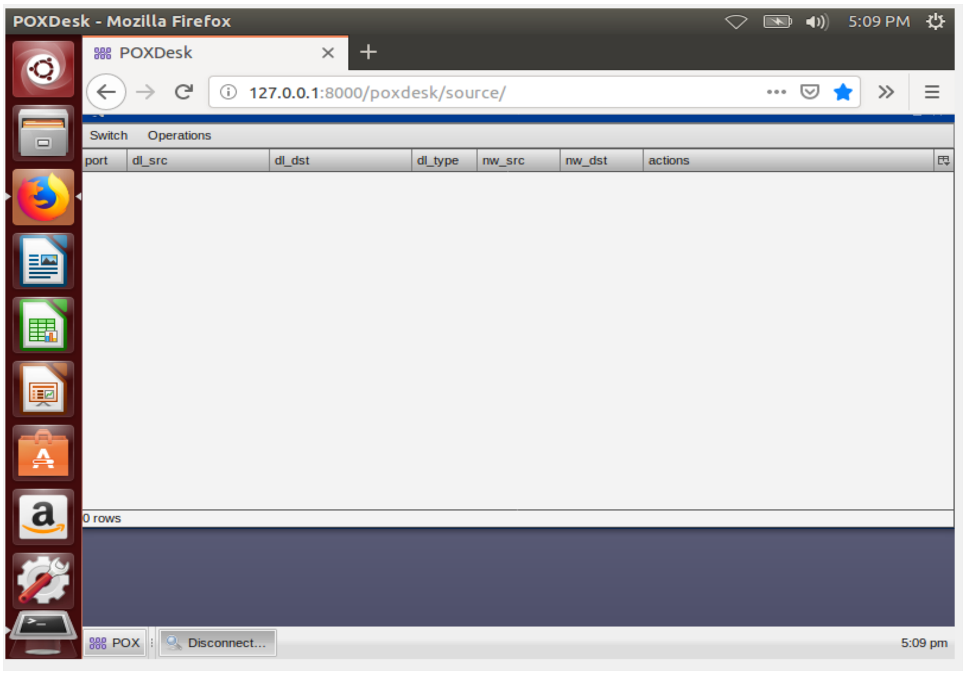
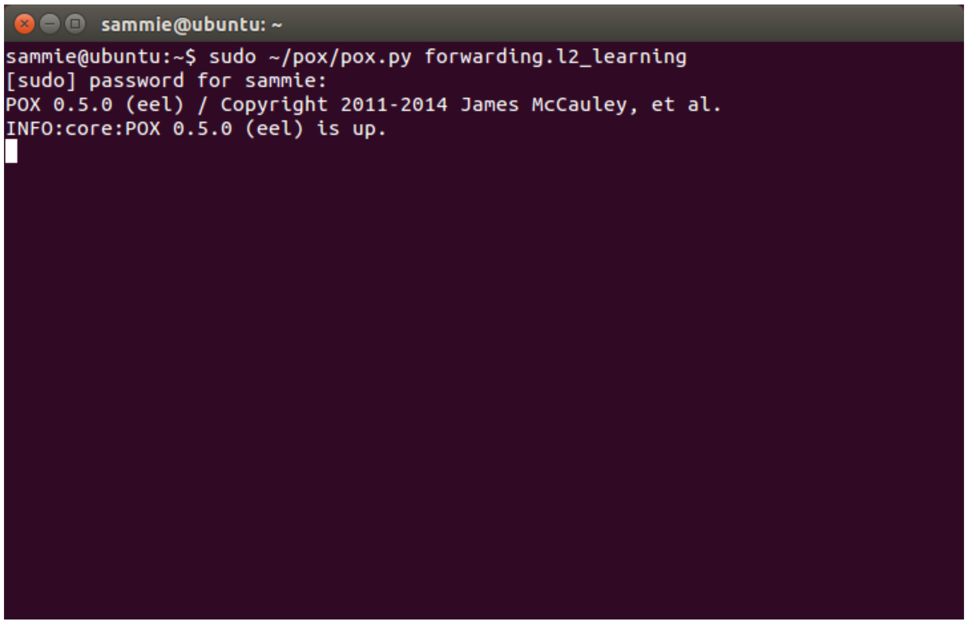


Figure 15: POXDesk

7.3.2 CONNECTING MININET & POX

✓ POX uses the forwarding.l2_learning component in learning like a layer 2 device i.e. switch. We run POX by running pox.py script and specifying the 'forwarding.l2_learning' component.

```
$ sudo ~/pox/pox.py forwarding.l2_learning
```

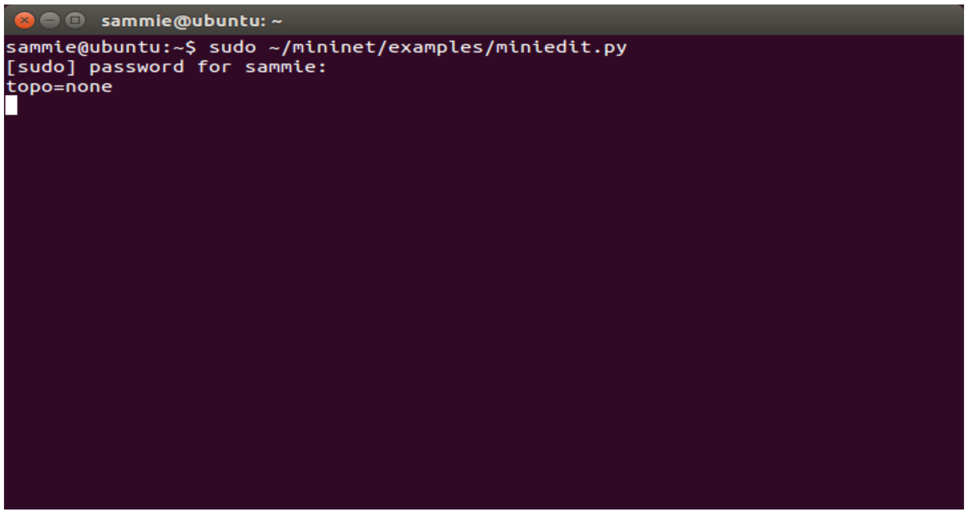


```
sammie@ubuntu: ~
sammie@ubuntu:~$ sudo ~/pox/pox.py forwarding.l2_learning
[sudo] password for sammie:
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.5.0 (eel) is up.
```

Figure 16: POX forwarding.l2

- ✓ Next, we can either use Miniedit, which is a graphical component for building the topology or use the Mininet command line to build up a topology to work with. Another method could be writing a script in python that could implement the topology along with routes and the scenario that needs to be implemented. Miniedit version:

  $ sudo ~/mininet/examples/miniedit.py



```
sammie@ubuntu: ~
sammie@ubuntu:~$ sudo ~/mininet/examples/miniedit.py
[sudo] password for sammie:
topo=none
```

Figure 17: MiniEdit

✓ And then we can build up a topology. In this scenario, I've built up a 3 switch (s1, s2 and s3), one controller(c0) and four host (h1, h2, h3, h4) tree structure as topology.

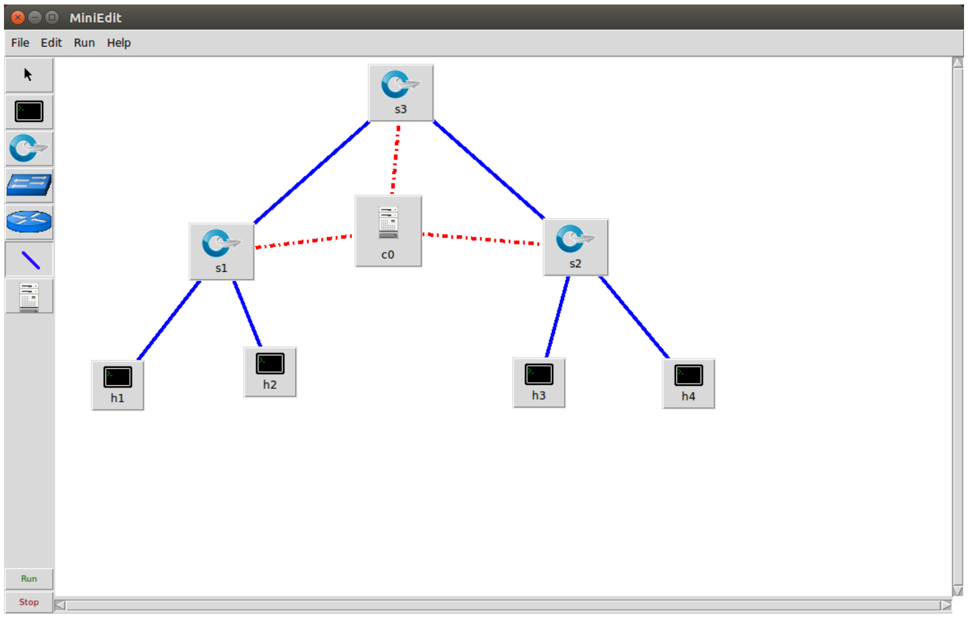
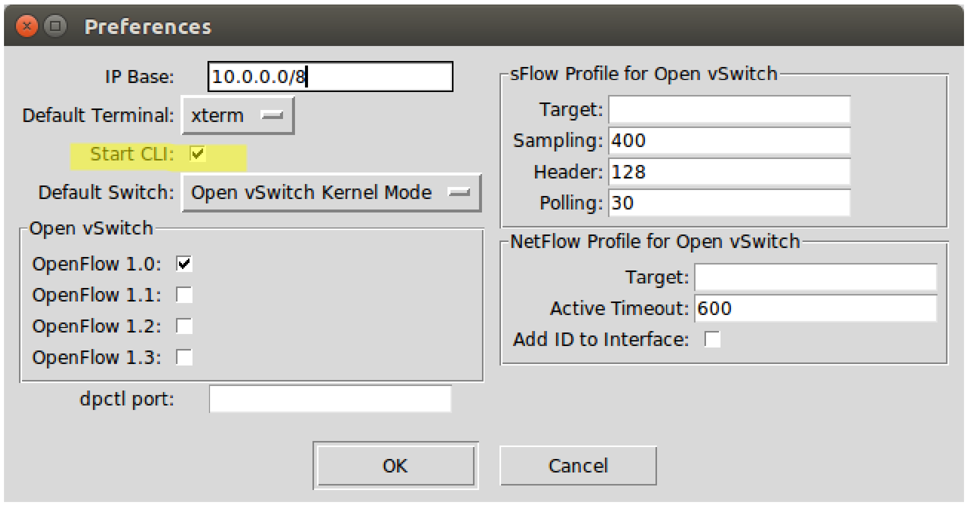


Figure 18: MiniEdit Topology

✓ We edit preferences in the Miniedit so we can access and make changes to topology from the CLI. We can also select the version of switch we wish to use.



IP Base: 10.0.0.0/8
Default Terminal: xterm
Start CLI: ✓
Default Switch: Open vSwitch Kernel Mode

Open vSwitch
OpenFlow 1.0: ✓
OpenFlow 1.1: ☐
OpenFlow 1.2: ☐
OpenFlow 1.3: ☐

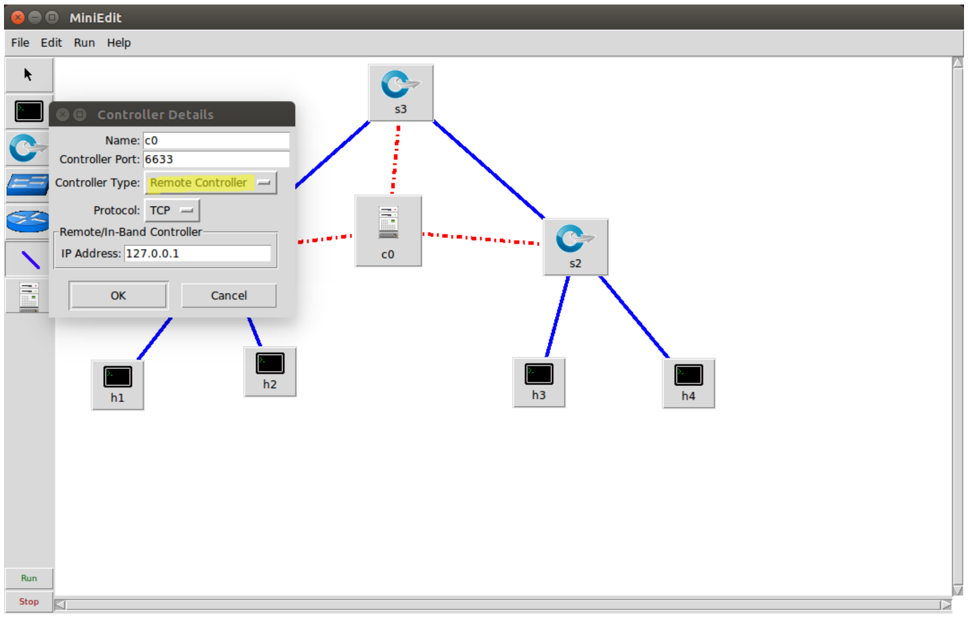
dpctl port:

sFlow Profile for Open vSwitch
Target:
Sampling: 400
Header: 128
Polling: 30

NetFlow Profile for Open vSwitch
Target:
Active Timeout: 600
Add ID to Interface: ☐

Figure 19: Controller Preference MiniEdit

- ✓ Also, since we're using POX as the controller, we select Remote controller as the option for the controller by going to Controller properties.

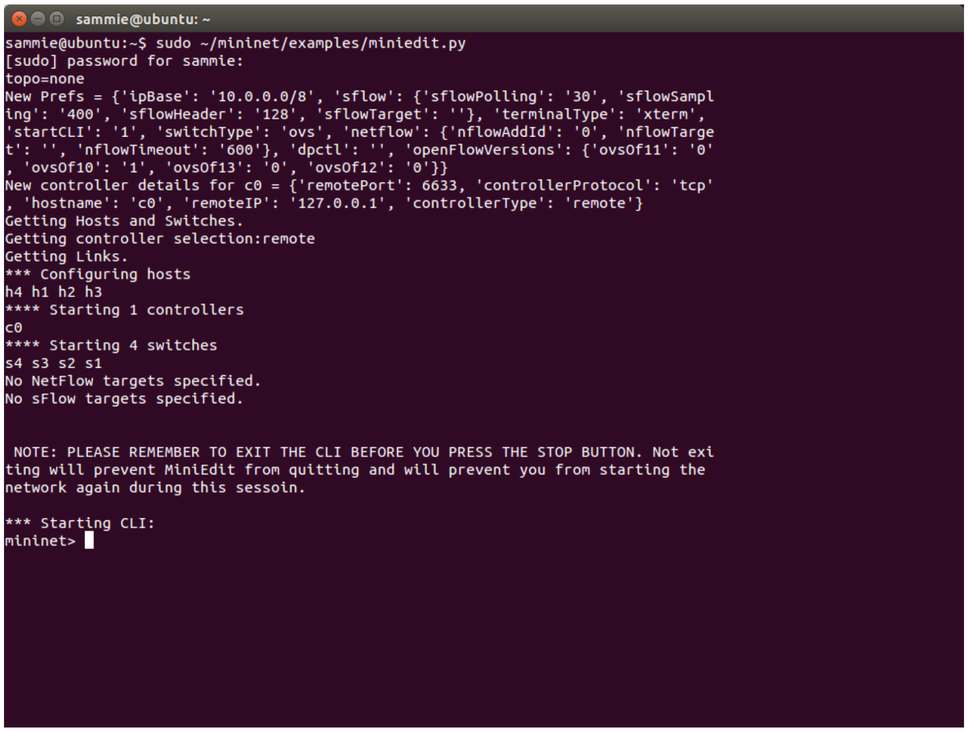


Name: c0
Controller Port: 6633
Controller Type: Remote Controller
Protocol: TCP
Remote/In-Band Controller
IP Address: 127.0.0.1

Figure 20: MiniEdit Controller Preference

- ✓ In this implementation, I ran POX controller in the same VM as the switches and hosts in Mininet, so the switches communicate with the remote POX controller using host system's loopback address and default OpenFlow port number (6633).

✓ Next, we RUN the topology.



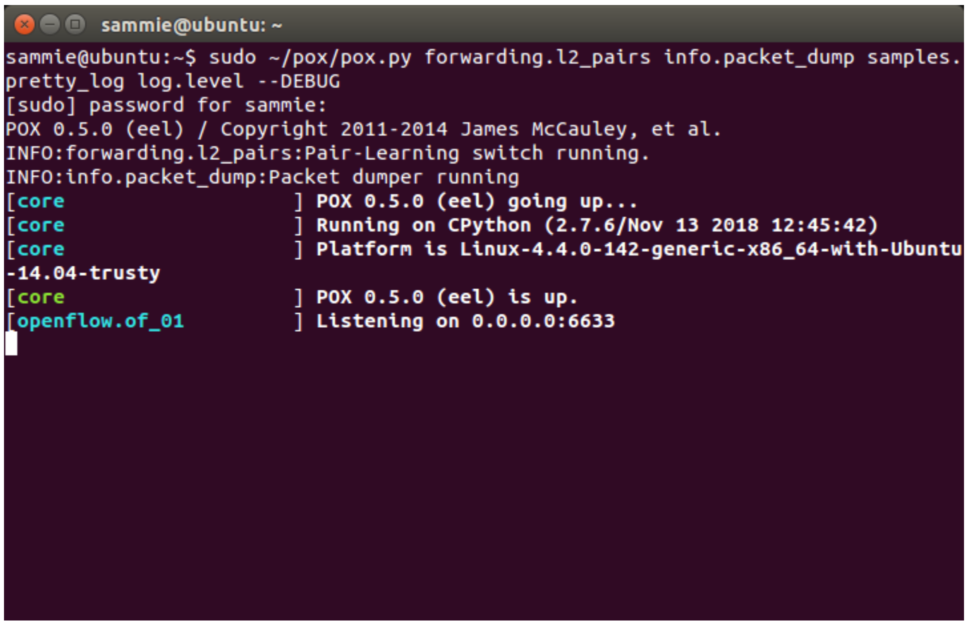
```
sammie@ubuntu:~$ sudo ~/mininet/examples/miniedit.py
[sudo] password for sammie:
topo=none
New Prefs = {'ipBase': '10.0.0.0/8', 'sflow': {'sflowPolling': '30', 'sflowSampl
ing': '400', 'sflowHeader': '128', 'sflowTarget': ''}, 'terminalType': 'xterm',
'startCLI': '1', 'switchType': 'ovs', 'netflow': {'nflowAddId': '0', 'nflowTarge
t': '', 'nflowTimeout': '600'}, 'dpctl': '', 'openFlowVersions': {'ovsOf11': '0'
, 'ovsOf10': '1', 'ovsOf13': '0', 'ovsOf12': '0'}}
New controller details for c0 = {'remotePort': 6633, 'controllerProtocol': 'tcp'
, 'hostname': 'c0', 'remoteIP': '127.0.0.1', 'controllerType': 'remote'}
Getting Hosts and Switches.
Getting controller selection:remote
Getting Links.
*** Configuring hosts
h4 h1 h2 h3
**** Starting 1 controllers
c0
**** Starting 4 switches
s4 s3 s2 s1
No NetFlow targets specified.
No sFlow targets specified.


 NOTE: PLEASE REMEMBER TO EXIT THE CLI BEFORE YOU PRESS THE STOP BUTTON. Not exi
ting will prevent MiniEdit from quitting and will prevent you from starting the
network again during this sessoin.

*** Starting CLI:
mininet> 
```

Figure 21: Running MiniEdit Topology

✓ We now need to start the POX controller.

```
$ sudo ~/pox/pox.py forwarding.l2_pairs info.packet_dump
samples.pretty_log log.level -DEBUG
```

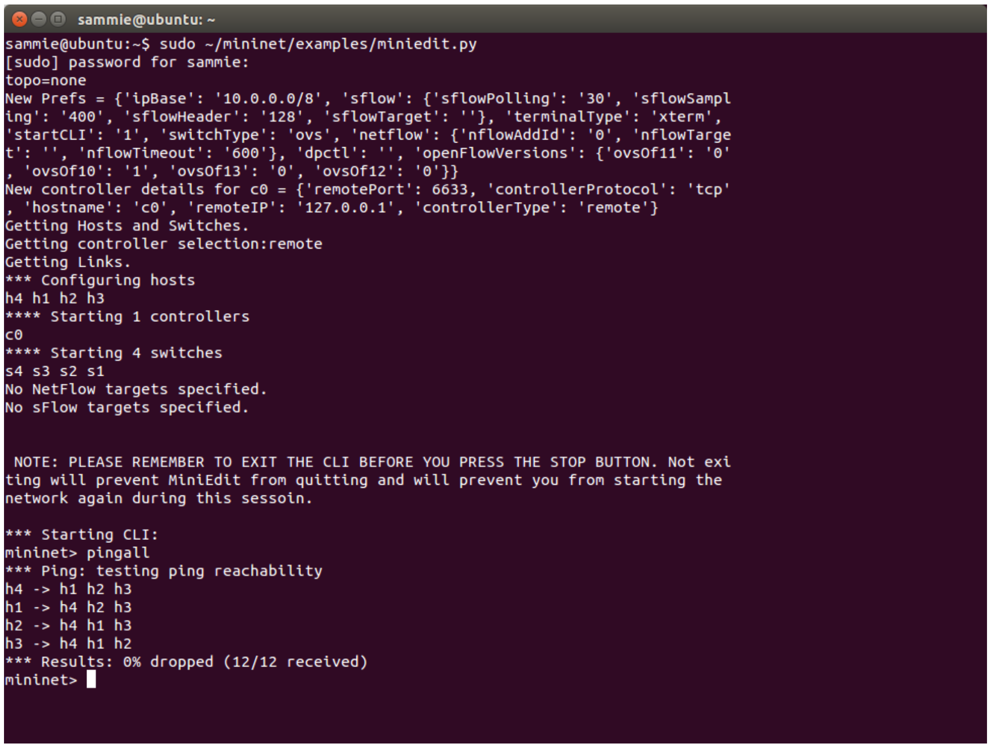
This starts to show the logs as the controller starts and connects to the switches we set in the topology.



```
sammie@ubuntu:~$ sudo ~/pox/pox.py forwarding.l2_pairs info.packet_dump samples.
pretty_log log.level --DEBUG
[sudo] password for sammie:
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:forwarding.l2_pairs:Pair-Learning switch running.
INFO:info.packet_dump:Packet dumper running
[core                    ] POX 0.5.0 (eel) going up...
[core                    ] Running on CPython (2.7.6/Nov 13 2018 12:45:42)
[core                    ] Platform is Linux-4.4.0-142-generic-x86_64-with-Ubuntu
-14.04-trusty
[core                    ] POX 0.5.0 (eel) is up.
[openflow.of_01          ] Listening on 0.0.0.0:6633
```

Figure 22: Starting POX Controller

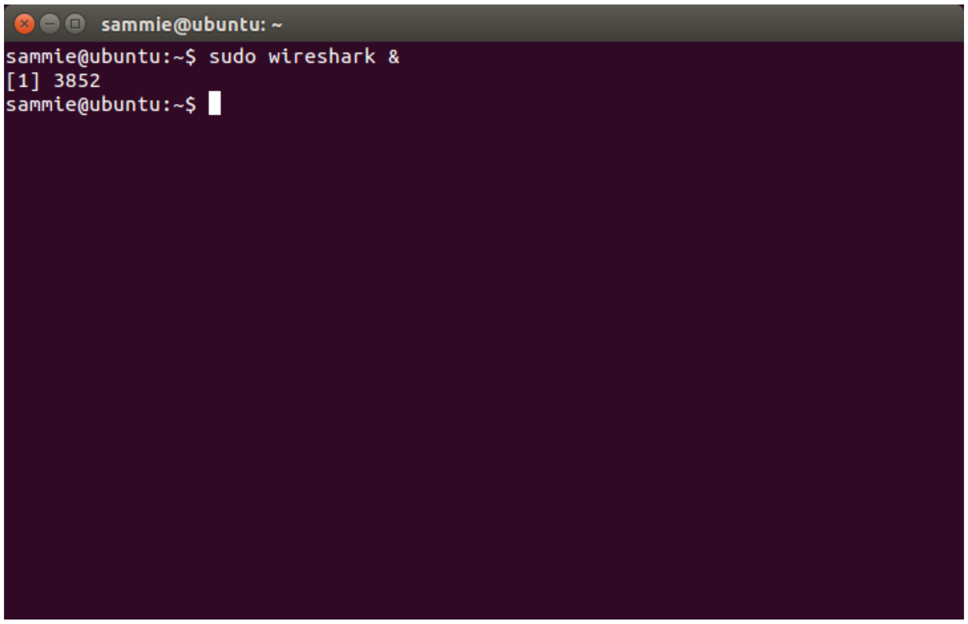
✓ Next, we try the connectivity amongst the nodes via ‘`pingall`’ command.



```
sammie@ubuntu:~$ sudo ~/mininet/examples/miniedit.py
[sudo] password for sammie:
topo=none
New Prefs = {'ipBase': '10.0.0.0/8', 'sflow': {'sflowPolling': '30', 'sflowSampl
ing': '400', 'sflowHeader': '128', 'sflowTarget': ''}, 'terminalType': 'xterm',
'startCLI': '1', 'switchType': 'ovs', 'netflow': {'nflowAddId': '0', 'nflowTarge
t': '', 'nflowTimeout': '600'}, 'dpctl': '', 'openFlowVersions': {'ovsOf11': '0'
, 'ovsOf10': '1', 'ovsOf13': '0', 'ovsOf12': '0'}}
New controller details for c0 = {'remotePort': 6633, 'controllerProtocol': 'tcp'
, 'hostname': 'c0', 'remoteIP': '127.0.0.1', 'controllerType': 'remote'}
Getting Hosts and Switches.
Getting controller selection:remote
Getting Links.
*** Configuring hosts
h4 h1 h2 h3
**** Starting 1 controllers
c0
**** Starting 4 switches
s4 s3 s2 s1
No NetFlow targets specified.
No sFlow targets specified.


 NOTE: PLEASE REMEMBER TO EXIT THE CLI BEFORE YOU PRESS THE STOP BUTTON. Not exi
ting will prevent MiniEdit from quitting and will prevent you from starting the
network again during this sessoin.

*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h4 -> h1 h2 h3
h1 -> h4 h2 h3
h2 -> h4 h1 h3
h3 -> h4 h1 h2
*** Results: 0% dropped (12/12 received)
mininet>
```

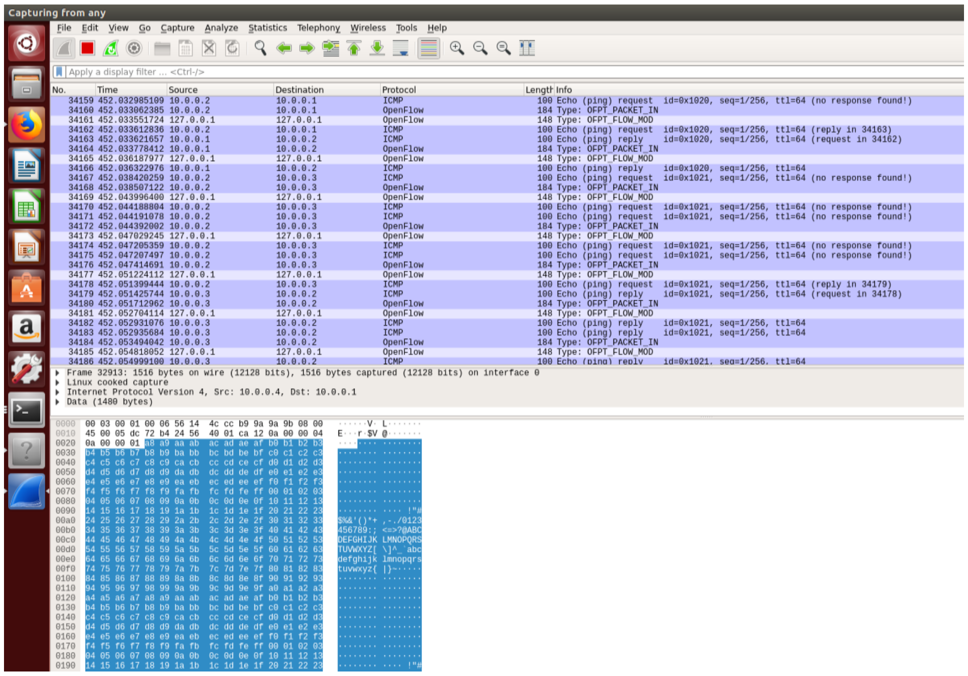
Figure 23: Testing Connectivity amongst all nodes

✓ We start Wireshark to see what packets are being captured in terms of traffic.



```
sammie@ubuntu:~$ sudo wireshark &
[1] 3852
sammie@ubuntu:~$ 
```

Figure 24: Starting Wireshark to capture packets


Capturing from any
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

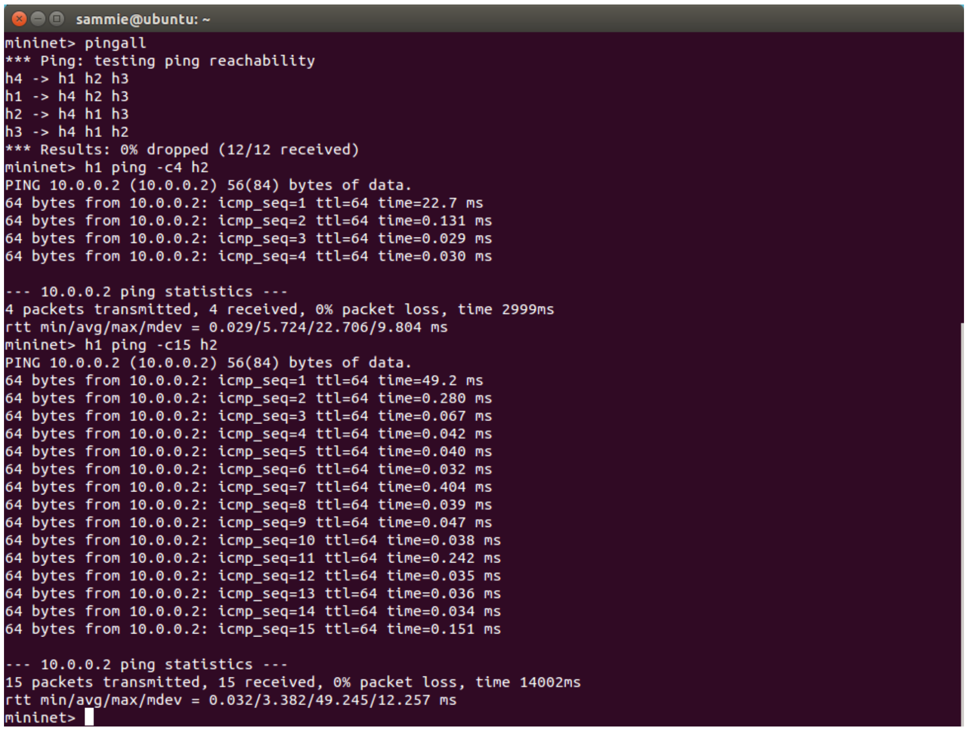
```
No.   Time          Source     Destination Protocol Length Info
34159 452.032985109 10.0.0.2   10.0.0.1    ICMP     100 Echo (ping) request  id=0x1020, seq=1/256, ttl=64 (no response found!)
34160 452.033062385 10.0.0.2   10.0.0.1    OpenFlow 184 Type: OFPT_PACKET_IN
34161 452.033551724 127.0.0.1  127.0.0.1   OpenFlow 148 Type: OFPT_FLOW_MOD
34162 452.033612836 10.0.0.2   10.0.0.1    ICMP     100 Echo (ping) request  id=0x1020, seq=1/256, ttl=64 (reply in 34163)
34163 452.033621657 10.0.0.1   10.0.0.2    ICMP     100 Echo (ping) reply    id=0x1020, seq=1/256, ttl=64 (request in 34162)
34164 452.033778412 10.0.0.1   10.0.0.2    OpenFlow 184 Type: OFPT_PACKET_IN
34165 452.036187977 127.0.0.1  127.0.0.1   OpenFlow 148 Type: OFPT_FLOW_MOD
34166 452.036322976 10.0.0.1   10.0.0.2    ICMP     100 Echo (ping) reply    id=0x1020, seq=1/256, ttl=64
34167 452.038420259 10.0.0.2   10.0.0.3    ICMP     100 Echo (ping) request  id=0x1021, seq=1/256, ttl=64 (no response found!)
34168 452.038507122 10.0.0.2   10.0.0.3    OpenFlow 184 Type: OFPT_PACKET_IN
34169 452.043996400 127.0.0.1  127.0.0.1   OpenFlow 148 Type: OFPT_FLOW_MOD
34170 452.044188804 10.0.0.2   10.0.0.3    ICMP     100 Echo (ping) request  id=0x1021, seq=1/256, ttl=64 (no response found!)
34171 452.044191078 10.0.0.2   10.0.0.3    ICMP     100 Echo (ping) request  id=0x1021, seq=1/256, ttl=64 (no response found!)
34172 452.044392002 10.0.0.2   10.0.0.3    OpenFlow 184 Type: OFPT_PACKET_IN
34173 452.047029245 127.0.0.1  127.0.0.1   OpenFlow 148 Type: OFPT_FLOW_MOD
34174 452.047205359 10.0.0.2   10.0.0.3    ICMP     100 Echo (ping) request  id=0x1021, seq=1/256, ttl=64 (no response found!)
34175 452.047207497 10.0.0.2   10.0.0.3    ICMP     100 Echo (ping) request  id=0x1021, seq=1/256, ttl=64 (no response found!)
34176 452.047414691 10.0.0.2   10.0.0.3    OpenFlow 184 Type: OFPT_PACKET_IN
34177 452.051224112 127.0.0.1  127.0.0.1   OpenFlow 148 Type: OFPT_FLOW_MOD
34178 452.051399444 10.0.0.2   10.0.0.3    ICMP     100 Echo (ping) request  id=0x1021, seq=1/256, ttl=64 (reply in 34179)
34179 452.051425744 10.0.0.3   10.0.0.2    ICMP     100 Echo (ping) reply    id=0x1021, seq=1/256, ttl=64 (request in 34178)
34180 452.051712962 10.0.0.3   10.0.0.2    OpenFlow 184 Type: OFPT_PACKET_IN
34181 452.052704114 127.0.0.1  127.0.0.1   OpenFlow 148 Type: OFPT_FLOW_MOD
34182 452.052931076 10.0.0.3   10.0.0.2    ICMP     100 Echo (ping) reply    id=0x1021, seq=1/256, ttl=64
34183 452.052935684 10.0.0.3   10.0.0.2    ICMP     100 Echo (ping) reply    id=0x1021, seq=1/256, ttl=64
34184 452.053494042 10.0.0.3   10.0.0.2    OpenFlow 184 Type: OFPT_PACKET_IN
34185 452.054818052 127.0.0.1  127.0.0.1   OpenFlow 148 Type: OFPT_FLOW_MOD
34186 452.054999100 10.0.0.3   10.0.0.2    ICMP     100 Echo (ping) reply    id=0x1021, seq=1/256, ttl=64
```

```
▸ Frame 32913: 1516 bytes on wire (12128 bits), 1516 bytes captured (12128 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 10.0.0.4, Dst: 10.0.0.1
▸ Data (1480 bytes)
```

```
0000  00 03 00 01 00 06 56 14  4c cc b9 9a 9a 9b 08 00   ······V· L·······
0010  45 00 05 dc 72 b4 24 56  40 01 ca 12 0a 00 00 04   E···r·$V @·······
0020  0a 00 00 01 a8 a9 aa ab  ac ad ae af b0 b1 b2 b3   ········ ········
0030  b4 b5 b6 b7 b8 b9 ba bb  bc bd be bf c0 c1 c2 c3   ········ ········
0040  c4 c5 c6 c7 c8 c9 ca cb  cc cd ce cf d0 d1 d2 d3   ········ ········
0050  d4 d5 d6 d7 d8 d9 da db  dc dd de df e0 e1 e2 e3   ········ ········
0060  e4 e5 e6 e7 e8 e9 ea eb  ec ed ee ef f0 f1 f2 f3   ········ ········
0070  f4 f5 f6 f7 f8 f9 fa fb  fc fd fe ff 00 01 02 03   ········ ········
0080  04 05 06 07 08 09 0a 0b  0c 0d 0e 0f 10 11 12 13   ········ ········
0090  14 15 16 17 18 19 1a 1b  1c 1d 1e 1f 20 21 22 23   ········ ···· !"#
00a0  24 25 26 27 28 29 2a 2b  2c 2d 2e 2f 30 31 32 33   $%&'()*+ ,-./0123
00b0  34 35 36 37 38 39 3a 3b  3c 3d 3e 3f 40 41 42 43   456789:; <=>?@ABC
00c0  44 45 46 47 48 49 4a 4b  4c 4d 4e 4f 50 51 52 53   DEFGHIJK LMNOPQRS
00d0  54 55 56 57 58 59 5a 5b  5c 5d 5e 5f 60 61 62 63   TUVWXYZ[ \]^_`abc
00e0  64 65 66 67 68 69 6a 6b  6c 6d 6e 6f 70 71 72 73   defghijk lmnopqrs
00f0  74 75 76 77 78 79 7a 7b  7c 7d 7e 7f 80 81 82 83   tuvwxyz{ |}~·····
0100  84 85 86 87 88 89 8a 8b  8c 8d 8e 8f 90 91 92 93   ········ ········
0110  94 95 96 97 98 99 9a 9b  9c 9d 9e 9f a0 a1 a2 a3   ········ ········
0120  a4 a5 a6 a7 a8 a9 aa ab  ac ad ae af b0 b1 b2 b3   ········ ········
0130  b4 b5 b6 b7 b8 b9 ba bb  bc bd be bf c0 c1 c2 c3   ········ ········
0140  c4 c5 c6 c7 c8 c9 ca cb  cc cd ce cf d0 d1 d2 d3   ········ ········
0150  d4 d5 d6 d7 d8 d9 da db  dc dd de df e0 e1 e2 e3   ········ ········
0160  e4 e5 e6 e7 e8 e9 ea eb  ec ed ee ef f0 f1 f2 f3   ········ ········
0170  f4 f5 f6 f7 f8 f9 fa fb  fc fd fe ff 00 01 02 03   ········ ········
0180  04 05 06 07 08 09 0a 0b  0c 0d 0e 0f 10 11 12 13   ········ ········
0190  14 15 16 17 18 19 1a 1b  1c 1d 1e 1f 20 21 22 23   ········ ···· !"#
```

Figure 25: Wireshark Capture

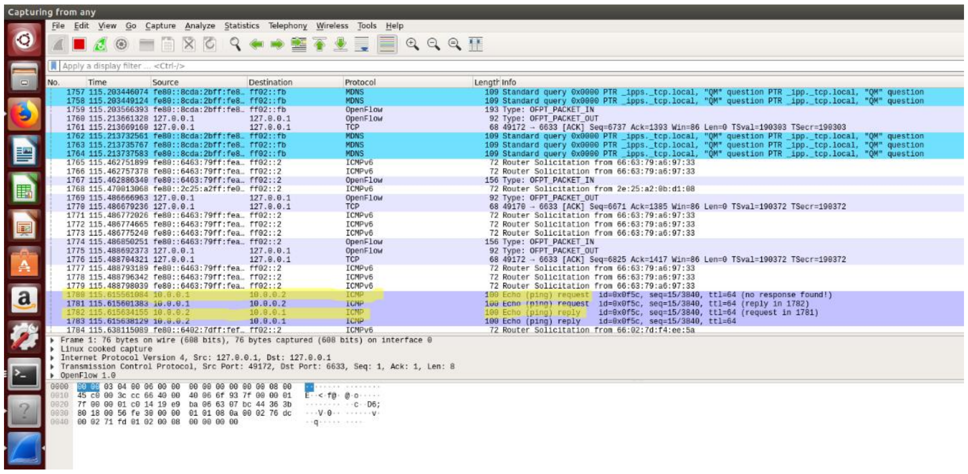
✓ I tried pinging between nodes taking different packet sizes. Standard Packet size of 64 bytes



```
mininet> pingall
*** Ping: testing ping reachability
h4 -> h1 h2 h3
h1 -> h4 h2 h3
h2 -> h4 h1 h3
h3 -> h4 h1 h2
*** Results: 0% dropped (12/12 received)
mininet> h1 ping -c4 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=22.7 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.131 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.029 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.030 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.029/5.724/22.706/9.804 ms
mininet> h1 ping -c15 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=49.2 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.280 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.067 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.042 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.040 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.032 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.404 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.039 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.038 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.242 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.035 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=0.036 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=0.034 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=0.151 ms

--- 10.0.0.2 ping statistics ---
15 packets transmitted, 15 received, 0% packet loss, time 14002ms
rtt min/avg/max/mdev = 0.032/3.382/49.245/12.257 ms
mininet> 
```

Figure 26: Pingall ; Standard Ping

✓ And the Wireshark captured traffic is as follows:

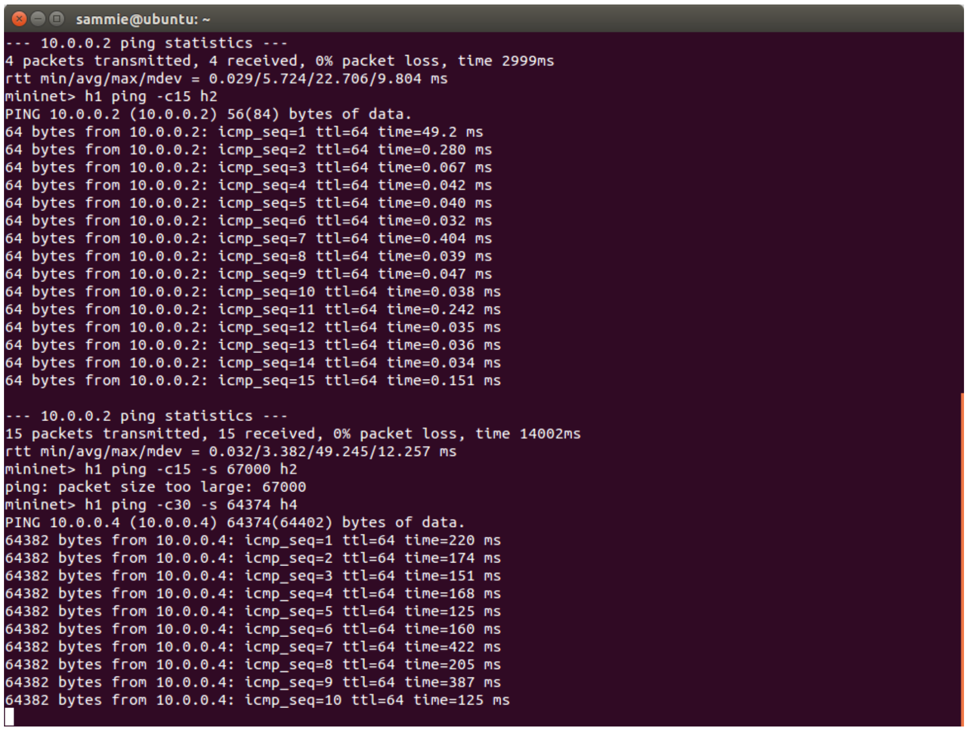


Capturing from any
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.  Time           Source                      Destination   Protocol  Length Info
1757 115.203446074  fe80::8cda:2bff:fe8…        ff02::fb      MDNS      109 Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question
1758 115.203449124  fe80::8cda:2bff:fe8…        ff02::fb      MDNS      109 Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question
1759 115.203566393  fe80::8cda:2bff:fe8…        ff02::fb      OpenFlow  193 Type: OFPT_PACKET_IN
1760 115.213661328  127.0.0.1                   127.0.0.1     OpenFlow   92 Type: OFPT_PACKET_OUT
1761 115.213669160  127.0.0.1                   127.0.0.1     TCP        68 49172 → 6633 [ACK] Seq=6737 Ack=1393 Win=86 Len=0 TSval=190303 TSecr=190303
1762 115.213732561  fe80::8cda:2bff:fe8…        ff02::fb      MDNS      109 Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question
1763 115.213735767  fe80::8cda:2bff:fe8…        ff02::fb      MDNS      109 Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question
1764 115.213737583  fe80::8cda:2bff:fe8…        ff02::fb      MDNS      109 Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question
1765 115.462751899  fe80::6463:79ff:fea…        ff02::2       ICMPv6     72 Router Solicitation from 66:63:79:a6:97:33
1766 115.462757378  fe80::6463:79ff:fea…        ff02::2       ICMPv6     72 Router Solicitation from 66:63:79:a6:97:33
1767 115.462886340  fe80::6463:79ff:fea…        ff02::2       OpenFlow  156 Type: OFPT_PACKET_IN
1768 115.470013068  fe80::2c25:a2ff:fe0…        ff02::2       ICMPv6     72 Router Solicitation from 2e:25:a2:0b:d1:08
1769 115.486666963  127.0.0.1                   127.0.0.1     OpenFlow   92 Type: OFPT_PACKET_OUT
1770 115.486679236  127.0.0.1                   127.0.0.1     TCP        68 49170 → 6633 [ACK] Seq=6671 Ack=1385 Win=86 Len=0 TSval=190372 TSecr=190372
1771 115.486772026  fe80::6463:79ff:fea…        ff02::2       ICMPv6     72 Router Solicitation from 66:63:79:a6:97:33
1772 115.486774665  fe80::6463:79ff:fea…        ff02::2       ICMPv6     72 Router Solicitation from 66:63:79:a6:97:33
1773 115.486775240  fe80::6463:79ff:fea…        ff02::2       ICMPv6     72 Router Solicitation from 66:63:79:a6:97:33
1774 115.486850251  fe80::6463:79ff:fea…        ff02::2       OpenFlow  156 Type: OFPT_PACKET_IN
1775 115.488692373  127.0.0.1                   127.0.0.1     OpenFlow   92 Type: OFPT_PACKET_OUT
1776 115.488704321  127.0.0.1                   127.0.0.1     TCP        68 49172 → 6633 [ACK] Seq=6825 Ack=1417 Win=86 Len=0 TSval=190372 TSecr=190372
1777 115.488793189  fe80::6463:79ff:fea…        ff02::2       ICMPv6     72 Router Solicitation from 66:63:79:a6:97:33
1778 115.488796342  fe80::6463:79ff:fea…        ff02::2       ICMPv6     72 Router Solicitation from 66:63:79:a6:97:33
1779 115.488798039  fe80::6463:79ff:fea…        ff02::2       ICMPv6     72 Router Solicitation from 66:63:79:a6:97:33
1780 115.615561084  10.0.0.1                    10.0.0.2      ICMP      100 Echo (ping) request  id=0x0f5c, seq=15/3840, ttl=64 (no response found!)
1781 115.615601383  10.0.0.1                    10.0.0.2      ICMP      100 Echo (ping) request  id=0x0f5c, seq=15/3840, ttl=64 (reply in 1782)
1782 115.615634155  10.0.0.2                    10.0.0.1      ICMP      100 Echo (ping) reply    id=0x0f5c, seq=15/3840, ttl=64 (request in 1781)
1783 115.615638129  10.0.0.2                    10.0.0.1      ICMP      100 Echo (ping) reply    id=0x0f5c, seq=15/3840, ttl=64
1784 115.638115089  fe80::6402:7dff:fef…        ff02::2       ICMPv6     72 Router Solicitation from 66:02:7d:f4:ee:5a

▸ Frame 1: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 49172, Dst Port: 6633, Seq: 1, Ack: 1, Len: 8
▸ OpenFlow 1.0

0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 08 00   ········ ········
0010  45 c0 00 3c cc 66 40 00  40 06 6f 93 7f 00 00 01   E··<·f@· @·o·····
0020  7f 00 00 01 c0 14 19 e9  ba 06 63 07 bc 44 36 3b   ········ ··c··D6;
0030  80 18 00 56 fe 30 00 00  01 01 08 0a 00 02 76 dc   ···V·0·· ······v·
0040  00 02 71 fd 01 02 00 08  00 00 00 00               ··q····· ····
```

Figure 27: Wireshark Capture for standard ping

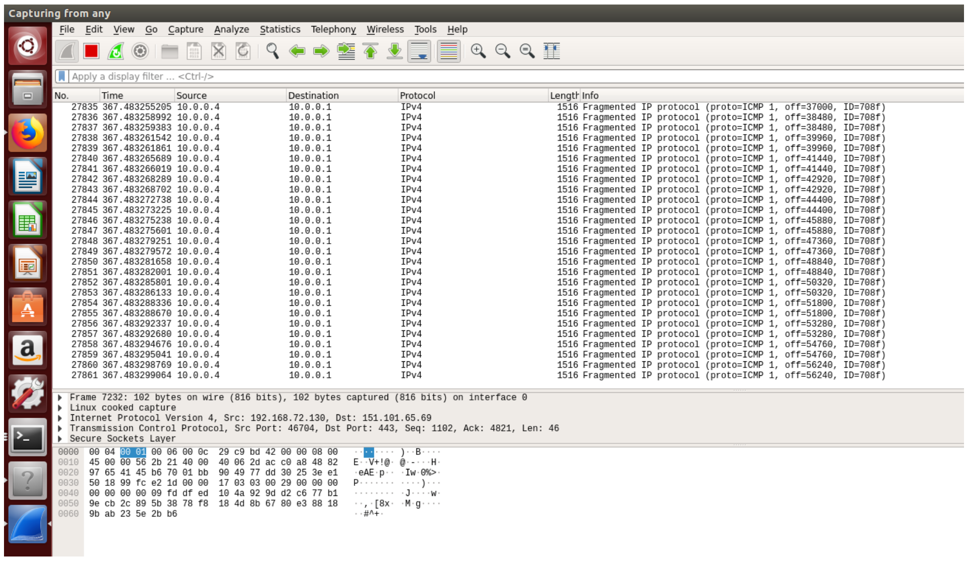
✓ Packet size = 64382 bytes



```
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.029/5.724/22.706/9.804 ms
mininet> h1 ping -c15 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=49.2 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.280 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.067 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.042 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.040 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.032 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.404 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.039 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.038 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.242 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.035 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=0.036 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=0.034 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=0.151 ms

--- 10.0.0.2 ping statistics ---
15 packets transmitted, 15 received, 0% packet loss, time 14002ms
rtt min/avg/max/mdev = 0.032/3.382/49.245/12.257 ms
mininet> h1 ping -c15 -s 67000 h2
ping: packet size too large: 67000
mininet> h1 ping -c30 -s 64374 h4
PING 10.0.0.4 (10.0.0.4) 64374(64402) bytes of data.
64382 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=220 ms
64382 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=174 ms
64382 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=151 ms
64382 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=168 ms
64382 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=125 ms
64382 bytes from 10.0.0.4: icmp_seq=6 ttl=64 time=160 ms
64382 bytes from 10.0.0.4: icmp_seq=7 ttl=64 time=422 ms
64382 bytes from 10.0.0.4: icmp_seq=8 ttl=64 time=205 ms
64382 bytes from 10.0.0.4: icmp_seq=9 ttl=64 time=387 ms
64382 bytes from 10.0.0.4: icmp_seq=10 ttl=64 time=125 ms
```

Figure 28: Ping for packet size greater than standard

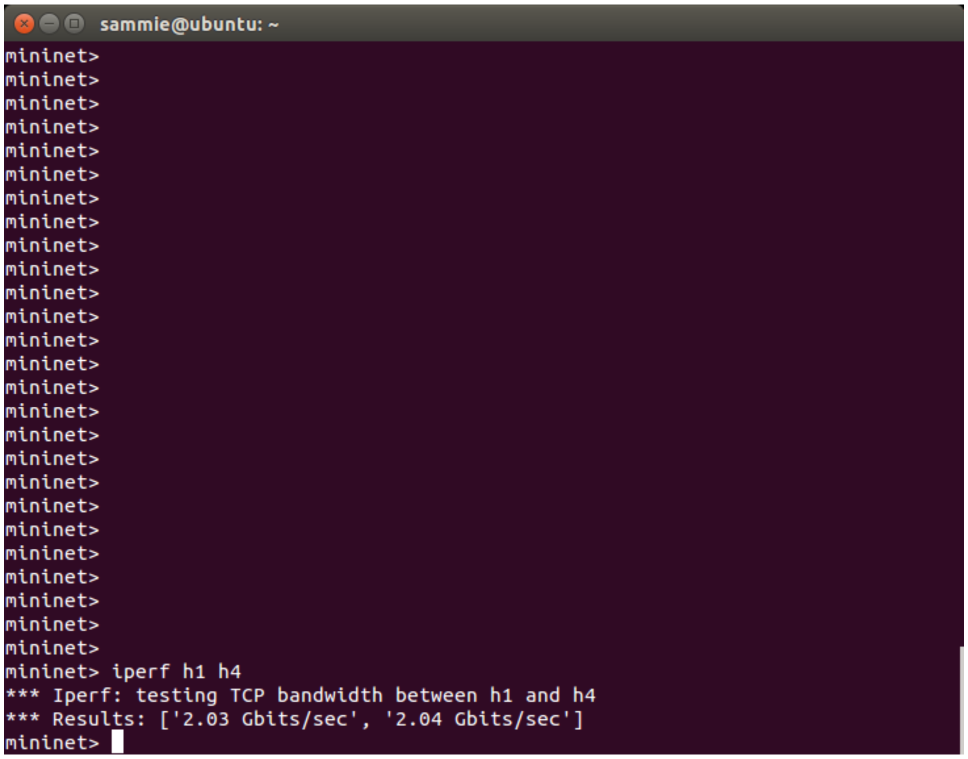
✓ The Wireshark traffic shows that the packets were fragmented.



```
No.   Time          Source      Destination  Protocol  Length Info
27835 367.483255205 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=37000, ID=708f)
27836 367.483258992 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=38480, ID=708f)
27837 367.483259383 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=38480, ID=708f)
27838 367.483261542 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=39960, ID=708f)
27839 367.483261861 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=39960, ID=708f)
27840 367.483265689 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=41440, ID=708f)
27841 367.483266019 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=41440, ID=708f)
27842 367.483268289 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=42920, ID=708f)
27843 367.483268702 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=42920, ID=708f)
27844 367.483272738 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=44400, ID=708f)
27845 367.483273225 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=44400, ID=708f)
27846 367.483275238 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=45880, ID=708f)
27847 367.483275601 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=45880, ID=708f)
27848 367.483279251 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=47360, ID=708f)
27849 367.483279572 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=47360, ID=708f)
27850 367.483281658 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=48840, ID=708f)
27851 367.483282001 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=48840, ID=708f)
27852 367.483285801 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=50320, ID=708f)
27853 367.483286133 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=50320, ID=708f)
27854 367.483288336 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=51800, ID=708f)
27855 367.483288670 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=51800, ID=708f)
27856 367.483292337 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=53280, ID=708f)
27857 367.483292680 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=53280, ID=708f)
27858 367.483294676 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=54760, ID=708f)
27859 367.483295041 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=54760, ID=708f)
27860 367.483298769 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=56240, ID=708f)
27861 367.483299064 10.0.0.4    10.0.0.1     IPv4      1516 Fragmented IP protocol (proto=ICMP 1, off=56240, ID=708f)

▸ Frame 7232: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 192.168.72.130, Dst: 151.101.65.69
▸ Transmission Control Protocol, Src Port: 46704, Dst Port: 443, Seq: 1102, Ack: 4821, Len: 46
▸ Secure Sockets Layer

0000  00 04 00 01 00 06 00 0c  29 c9 bd 42 00 00 08 00   ········ )··B····
0010  45 00 00 56 2b 21 40 00  40 06 2d ac c0 a8 48 82   E··V+!@· @·-···H·
0020  97 65 41 45 b6 70 01 bb  90 49 77 dd 30 25 3e e1   ·eAE·p·· ·Iw·0%>·
0030  50 18 99 fc e2 1d 00 00  17 03 03 00 29 00 00 00   P······· ····)···
0040  00 00 00 00 09 fd df ed  10 4a 92 9d d2 c6 77 b1   ········ ·J····w·
0050  9e cb 2c 89 5b 38 78 f8  18 4d 8b 67 80 e3 88 18   ··,·[8x· ·M·g····
0060  9b ab 23 5e 2b b6                                  ··#^+·
```

Figure 29: Wireshark Capture for packet size greater than standard

✓ Mininet also gives us option to test the link strength between nodes via iperf.



```
sammie@ubuntu: ~
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet>
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['2.03 Gbits/sec', '2.04 Gbits/sec']
mininet> 
```

Figure 30: Mininet iperf

7.4 FLOODLIGHT

Floodlight is a popular SDN controller from Big Switch Networks. Based on Beacon, it is a Java based OpenFlow supporting controller.

The architecture includes modules such as topology management, MAC and IP tracking components, GUI for web access, OpenFlow counters and storage abstraction developed into SQL and NoSQL backend.

It uses REST API for event notification system and Java Event Listeners to allow applications to know the state of the controller.

Floodlight has a module called Floodlight Provider that handles input and output stream from switches, translating OpenFlow messages to events.

The Topology manager uses LLDP (Link Layer Discovery Protocol) to discover end points, both OpenFlow based or non-OpenFlow based.

Unlike other Onix based controllers, including POX, it has a component called BigDB that is a NoSQL based database used for storing information including configuration and element state.

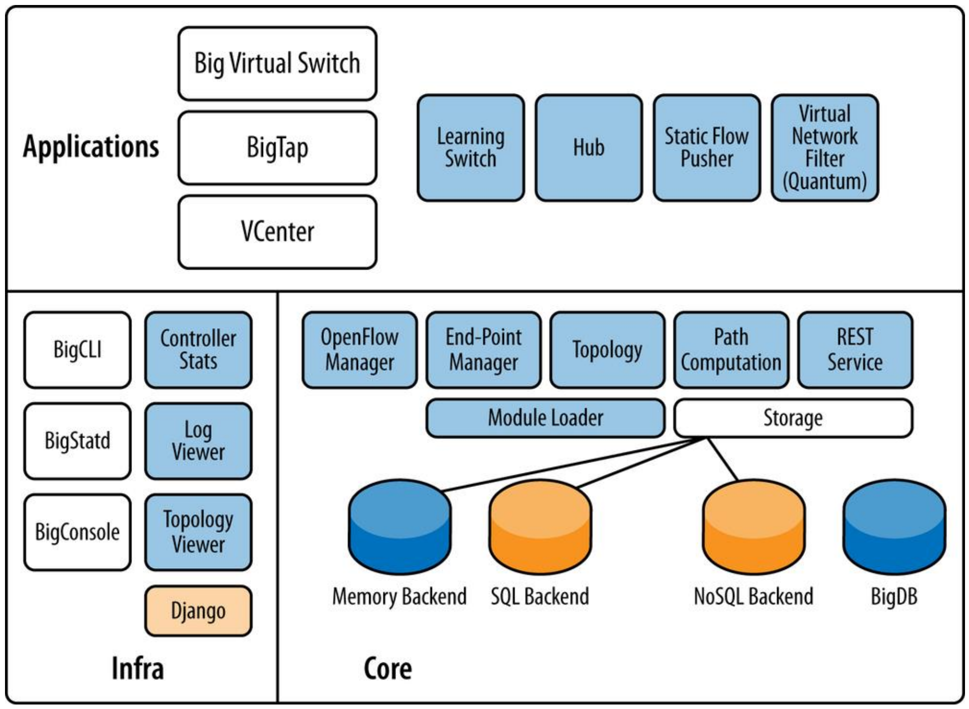


Figure 31: Floodlight architecture

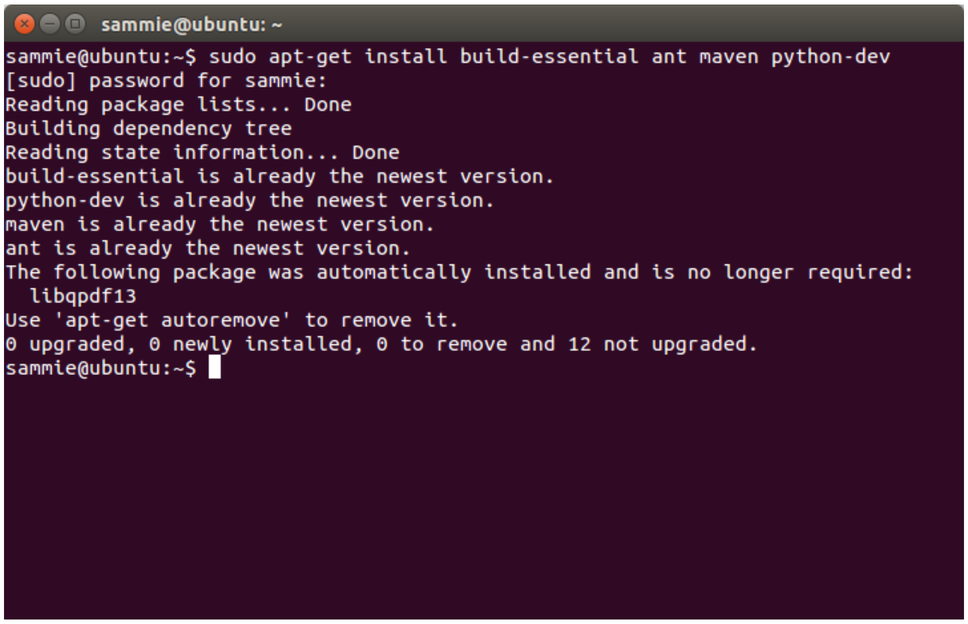
7.4.1 <u>Installing Floodlight</u>

✓ Since Floodlight is a Java based controller, we need Java Development kit which includes:

- ➢ JDK 8 for floodlight master
- ➢ JDK 7 for floodlight v1.2 and below

✓ For floodlight master, we download the dependencies:

```
$ sudo apt-get install build-essential ant maven python-dev
```



```
sammie@ubuntu:~$ sudo apt-get install build-essential ant maven python-dev
[sudo] password for sammie:
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version.
python-dev is already the newest version.
maven is already the newest version.
ant is already the newest version.
The following package was automatically installed and is no longer required:
  libqpdf13
Use 'apt-get autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 12 not upgraded.
sammie@ubuntu:~$ 
```

Figure 32: Installing Floodlight

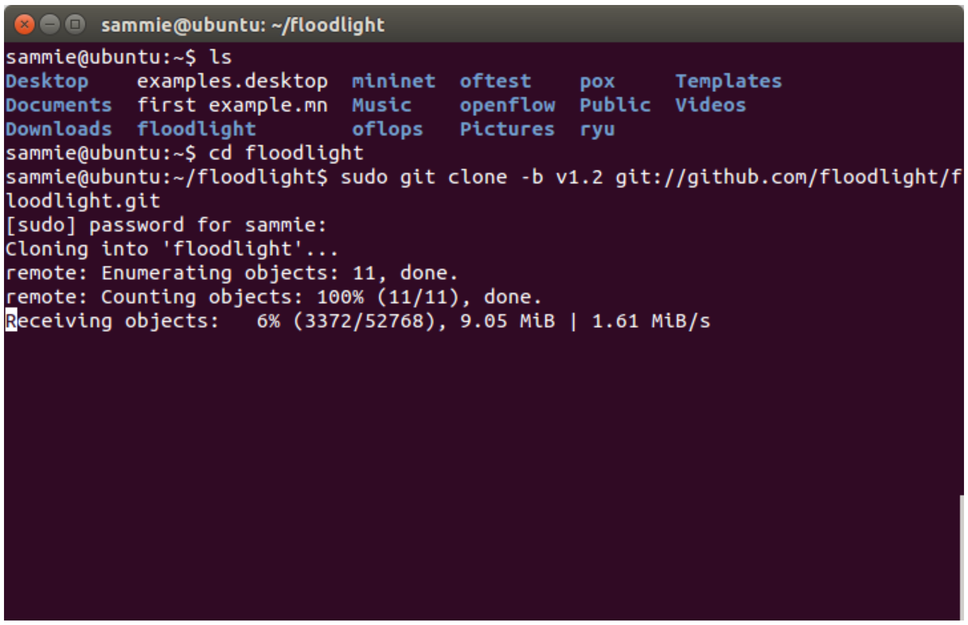
✓ For Floodlight v1.2 and below, we download dependencies:

$ sudo apt-get install build-essential openjdk-70jdk ant maven python-dev eclipse

Since we have the pre-requisites now, we download and build Floodlight from Github.

$ sudo git clone -b v1.2 git://github.com/floodlight/floodlight.git

(where -b v1.2 is the JDK version we're using, to ensure compatibility)

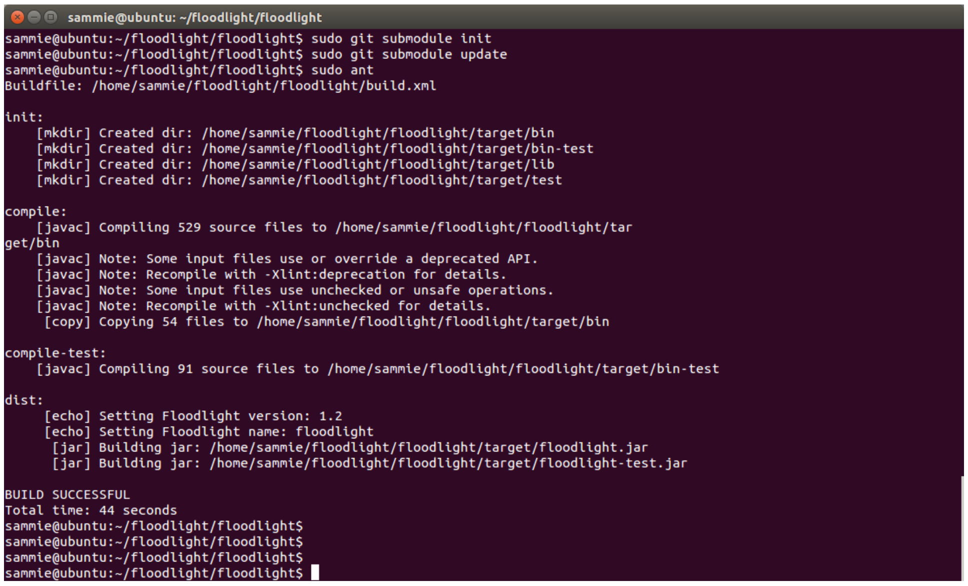


```
sammie@ubuntu: ~/floodlight
sammie@ubuntu:~$ ls
Desktop     examples.desktop  mininet  oftest    pox     Templates
Documents   first example.mn  Music    openflow  Public  Videos
Downloads   floodlight        oflops   Pictures  ryu
sammie@ubuntu:~$ cd floodlight
sammie@ubuntu:~/floodlight$ sudo git clone -b v1.2 git://github.com/floodlight/f
loodlight.git
[sudo] password for sammie:
Cloning into 'floodlight'...
remote: Enumerating objects: 11, done.
remote: Counting objects: 100% (11/11), done.
Receiving objects:   6% (3372/52768), 9.05 MiB | 1.61 MiB/s
```

Figure 33: Installing Floodlight from Git

✓ Installation is complete. Now we must build the controller:

```
$ cd floodlight
$ sudo git submodule init
$ sudo git submodule update
$ ant
```



```
sammie@ubuntu: ~/floodlight/floodlight
sammie@ubuntu:~/floodlight/floodlight$ sudo git submodule init
sammie@ubuntu:~/floodlight/floodlight$ sudo git submodule update
sammie@ubuntu:~/floodlight/floodlight$ sudo ant
Buildfile: /home/sammie/floodlight/floodlight/build.xml

init:
    [mkdir] Created dir: /home/sammie/floodlight/floodlight/target/bin
    [mkdir] Created dir: /home/sammie/floodlight/floodlight/target/bin-test
    [mkdir] Created dir: /home/sammie/floodlight/floodlight/target/lib
    [mkdir] Created dir: /home/sammie/floodlight/floodlight/target/test

compile:
    [javac] Compiling 529 source files to /home/sammie/floodlight/floodlight/tar
get/bin
    [javac] Note: Some input files use or override a deprecated API.
    [javac] Note: Recompile with -Xlint:deprecation for details.
    [javac] Note: Some input files use unchecked or unsafe operations.
    [javac] Note: Recompile with -Xlint:unchecked for details.
     [copy] Copying 54 files to /home/sammie/floodlight/floodlight/target/bin

compile-test:
    [javac] Compiling 91 source files to /home/sammie/floodlight/floodlight/target/bin-test

dist:
     [echo] Setting Floodlight version: 1.2
     [echo] Setting Floodlight name: floodlight
      [jar] Building jar: /home/sammie/floodlight/floodlight/target/floodlight.jar
      [jar] Building jar: /home/sammie/floodlight/floodlight/target/floodlight-test.jar

BUILD SUCCESSFUL
Total time: 44 seconds
sammie@ubuntu:~/floodlight/floodlight$
sammie@ubuntu:~/floodlight/floodlight$
sammie@ubuntu:~/floodlight/floodlight$
sammie@ubuntu:~/floodlight/floodlight$
```

Figure 34: Building Floodlight from jar file

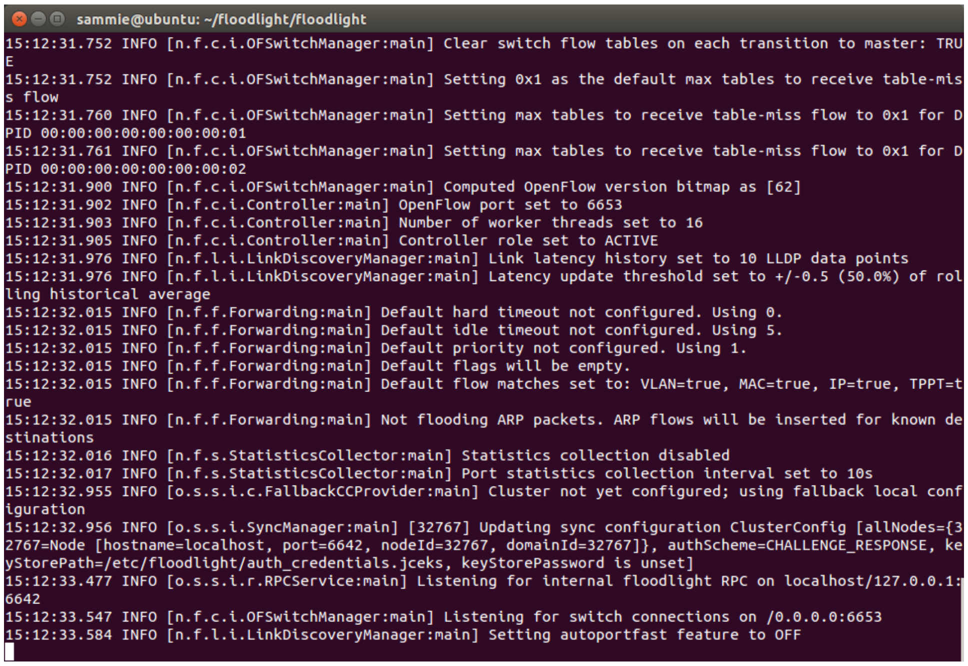
✓ After successfully building the controller, we need to make a floodlight directory with root permission:

```
$ sudo mkdir /var/lib/floodlight
$ sudo chmod 777 /var/lib/floodlight
```

✓ Now, we can run floodlight by:

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

(The floodlight.jar file is produced by ant during the build)

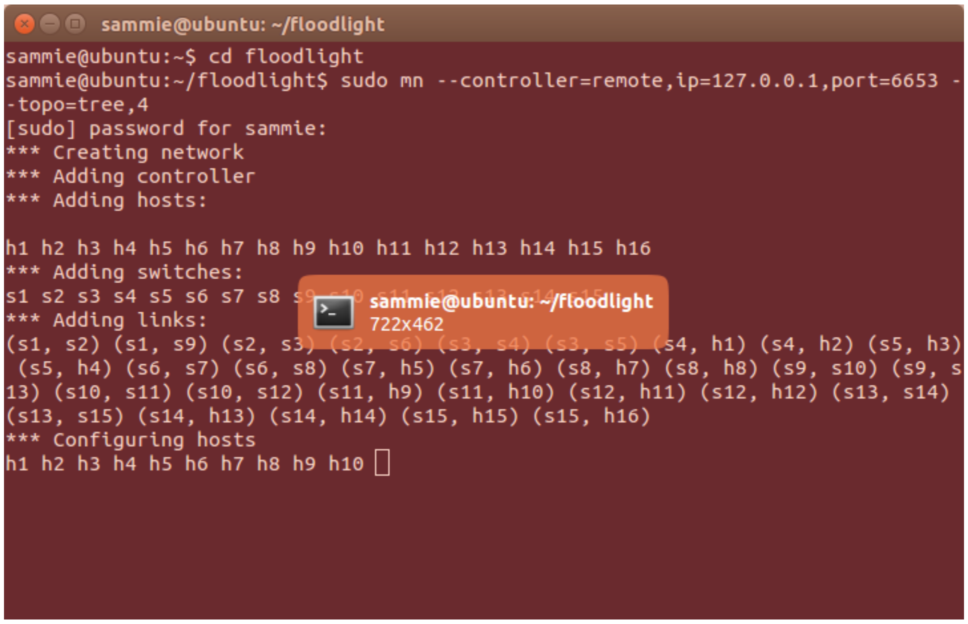


```
sammie@ubuntu: ~/floodlight/floodlight
15:12:31.752 INFO [n.f.c.i.OFSwitchManager:main] Clear switch flow tables on each transition to master: TRU
E
15:12:31.752 INFO [n.f.c.i.OFSwitchManager:main] Setting 0x1 as the default max tables to receive table-mis
s flow
15:12:31.760 INFO [n.f.c.i.OFSwitchManager:main] Setting max tables to receive table-miss flow to 0x1 for D
PID 00:00:00:00:00:00:00:01
15:12:31.761 INFO [n.f.c.i.OFSwitchManager:main] Setting max tables to receive table-miss flow to 0x1 for D
PID 00:00:00:00:00:00:00:02
15:12:31.900 INFO [n.f.c.i.OFSwitchManager:main] Computed OpenFlow version bitmap as [62]
15:12:31.902 INFO [n.f.c.i.Controller:main] OpenFlow port set to 6653
15:12:31.903 INFO [n.f.c.i.Controller:main] Number of worker threads set to 16
15:12:31.905 INFO [n.f.c.i.Controller:main] Controller role set to ACTIVE
15:12:31.976 INFO [n.f.l.i.LinkDiscoveryManager:main] Link latency history set to 10 LLDP data points
15:12:31.976 INFO [n.f.l.i.LinkDiscoveryManager:main] Latency update threshold set to +/-0.5 (50.0%) of rol
ling historical average
15:12:32.015 INFO [n.f.f.Forwarding:main] Default hard timeout not configured. Using 0.
15:12:32.015 INFO [n.f.f.Forwarding:main] Default idle timeout not configured. Using 5.
15:12:32.015 INFO [n.f.f.Forwarding:main] Default priority not configured. Using 1.
15:12:32.015 INFO [n.f.f.Forwarding:main] Default flags will be empty.
15:12:32.015 INFO [n.f.f.Forwarding:main] Default flow matches set to: VLAN=true, MAC=true, IP=true, TPPT=t
rue
15:12:32.015 INFO [n.f.f.Forwarding:main] Not flooding ARP packets. ARP flows will be inserted for known de
stinations
15:12:32.016 INFO [n.f.s.StatisticsCollector:main] Statistics collection disabled
15:12:32.017 INFO [n.f.s.StatisticsCollector:main] Port statistics collection interval set to 10s
15:12:32.955 INFO [o.s.s.i.c.FallbackCCProvider:main] Cluster not yet configured; using fallback local conf
iguration
15:12:32.956 INFO [o.s.s.i.SyncManager:main] [32767] Updating sync configuration ClusterConfig [allNodes={3
2767=Node [hostname=localhost, port=6642, nodeId=32767, domainId=32767]}, authScheme=CHALLENGE_RESPONSE, ke
yStorePath=/etc/floodlight/auth_credentials.jceks, keyStorePassword is unset]
15:12:33.477 INFO [o.s.s.i.r.RPCService:main] Listening for internal floodlight RPC on localhost/127.0.0.1:
6642
15:12:33.547 INFO [n.f.c.i.OFSwitchManager:main] Listening for switch connections on /0.0.0.0:6653
15:12:33.584 INFO [n.f.l.i.LinkDiscoveryManager:main] Setting autoportfast feature to OFF
```

Figure 35: Starting Floodlight

✓ We can connect it via mininet by going to floodlight directory and issuing a topology command:

```
$ sudo mn –topo=tree,4 –controller=remote, ip=127.0.0.1,port=6653
```

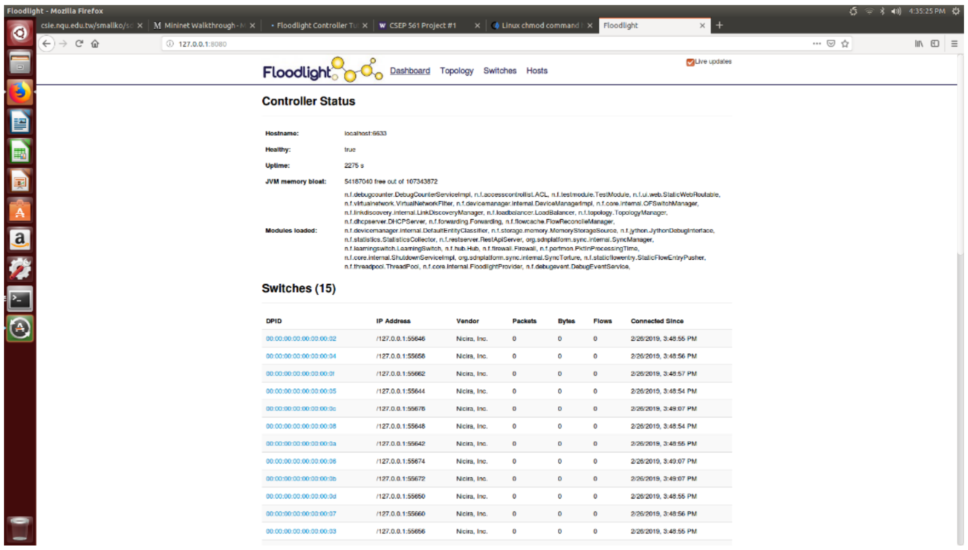


```
sammie@ubuntu:~$ cd floodlight
sammie@ubuntu:~/floodlight$ sudo mn --controller=remote,ip=127.0.0.1,port=6653 -
-topo=tree,4
[sudo] password for sammie:
*** Creating network
*** Adding controller
*** Adding hosts:

h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15
*** Adding links:
(s1, s2) (s1, s9) (s2, s3) (s2, s6) (s3, s4) (s3, s5) (s4, h1) (s4, h2) (s5, h3)
 (s5, h4) (s6, s7) (s6, s8) (s7, h5) (s7, h6) (s8, h7) (s8, h8) (s9, s10) (s9, s
13) (s10, s11) (s10, s12) (s11, h9) (s11, h10) (s12, h11) (s12, h12) (s13, s14)
(s13, s15) (s14, h13) (s14, h14) (s15, h15) (s15, h16)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10
```

Figure 36: Connecting Mininet with Floodlight

✓ We can use the Floodlight GUI (with topologies, hosts, switches and their information as MAC addresses etc) at

http://127.0.0.1:8080/ui/index.html



Controller Status

Hostname:	localhost:6633
Healthy:	true
Uptime:	2275 s
JVM memory bloat:	54187040 free out of 107343872
Modules loaded:	n.f.debugcounter.DebugCounterServiceImpl, n.f.accesscontrollist.ACL, n.f.testmodule.TestModule, n.f.ui.web.StaticWebRoutable, n.f.virtualnetwork.VirtualNetworkFilter, n.f.devicemanager.internal.DeviceManagerImpl, n.f.core.internal.OFSwitchManager, n.f.linkdiscovery.internal.LinkDiscoveryManager, n.f.loadbalancer.LoadBalancer, n.f.topology.TopologyManager, n.f.dhcpserver.DHCPServer, n.f.forwarding.Forwarding, n.f.flowcache.FlowReconcileManager, n.f.devicemanager.internal.DefaultEntityClassifier, n.f.storage.memory.MemoryStorageSource, n.f.jython.JythonDebugInterface, n.f.statistics.StatisticsCollector, n.f.restserver.RestApiServer, org.sdnplatform.sync.internal.SyncManager, n.f.learningswitch.LearningSwitch, n.f.hub.Hub, n.f.firewall.Firewall, n.f.perfmon.PktInProcessingTime, n.f.core.internal.ShutdownServiceImpl, org.sdnplatform.sync.internal.SyncTorture, n.f.staticflowentry.StaticFlowEntryPusher, n.f.threadpool.ThreadPool, n.f.core.internal.FloodlightProvider, n.f.debugevent.DebugEventService,

Switches (15)

DPID	IP Address	Vendor	Packets	Bytes	Flows	Connected Since
00:00:00:00:00:00:00:02	/127.0.0.1:55646	Nicira, Inc.	0	0	0	2/26/2019, 3:48:55 PM
00:00:00:00:00:00:00:04	/127.0.0.1:55658	Nicira, Inc.	0	0	0	2/26/2019, 3:48:56 PM
00:00:00:00:00:00:00:0f	/127.0.0.1:55662	Nicira, Inc.	0	0	0	2/26/2019, 3:48:57 PM
00:00:00:00:00:00:00:05	/127.0.0.1:55644	Nicira, Inc.	0	0	0	2/26/2019, 3:48:54 PM
00:00:00:00:00:00:00:0c	/127.0.0.1:55676	Nicira, Inc.	0	0	0	2/26/2019, 3:49:07 PM
00:00:00:00:00:00:00:08	/127.0.0.1:55648	Nicira, Inc.	0	0	0	2/26/2019, 3:48:54 PM
00:00:00:00:00:00:00:0a	/127.0.0.1:55642	Nicira, Inc.	0	0	0	2/26/2019, 3:48:55 PM
00:00:00:00:00:00:00:06	/127.0.0.1:55674	Nicira, Inc.	0	0	0	2/26/2019, 3:49:07 PM
00:00:00:00:00:00:00:0b	/127.0.0.1:55672	Nicira, Inc.	0	0	0	2/26/2019, 3:49:07 PM
00:00:00:00:00:00:00:0d	/127.0.0.1:55650	Nicira, Inc.	0	0	0	2/26/2019, 3:48:55 PM
00:00:00:00:00:00:00:07	/127.0.0.1:55660	Nicira, Inc.	0	0	0	2/26/2019, 3:48:56 PM
00:00:00:00:00:00:00:03	/127.0.0.1:55656	Nicira, Inc.	0	0	0	2/26/2019, 3:48:55 PM

Figure 37: Floodlight GUI showing all devices

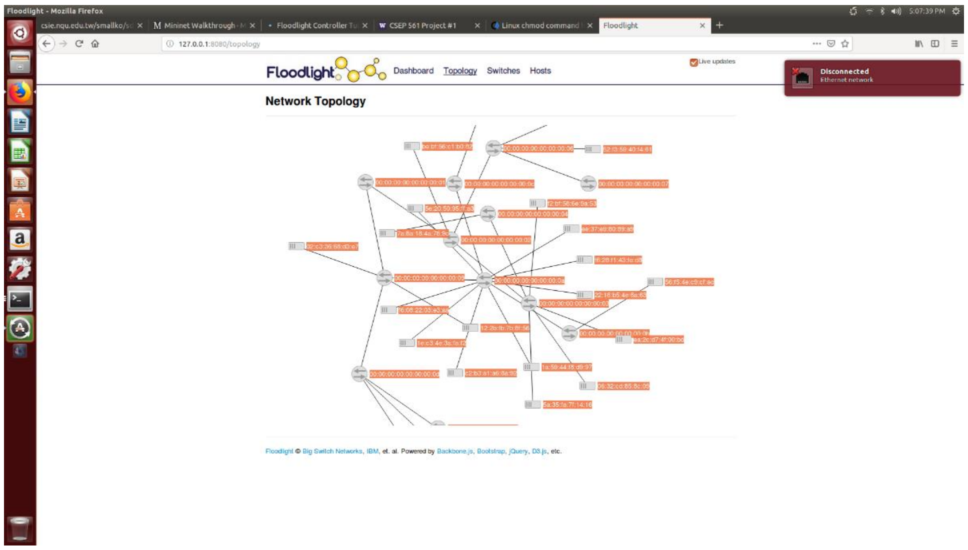
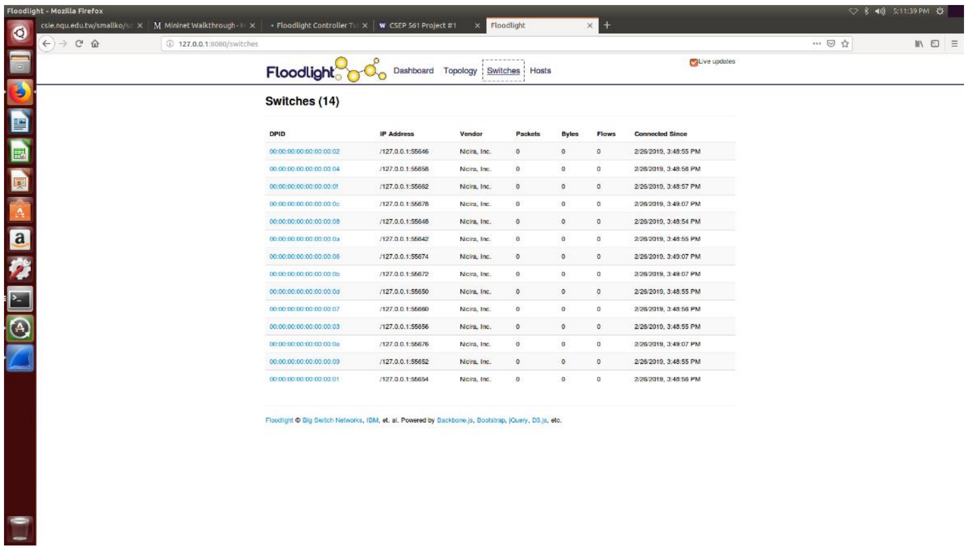

Floodlight
Dashboard
Topology
Switches
Hosts
Live updates
Disconnected
Ethernet network
Network Topology
Floodlight © Big Switch Networks, IBM, et. al. Powered by Backbone.js, Bootstrap, jQuery, D3.js, etc.

Figure 38: Floodlight GUI Topology

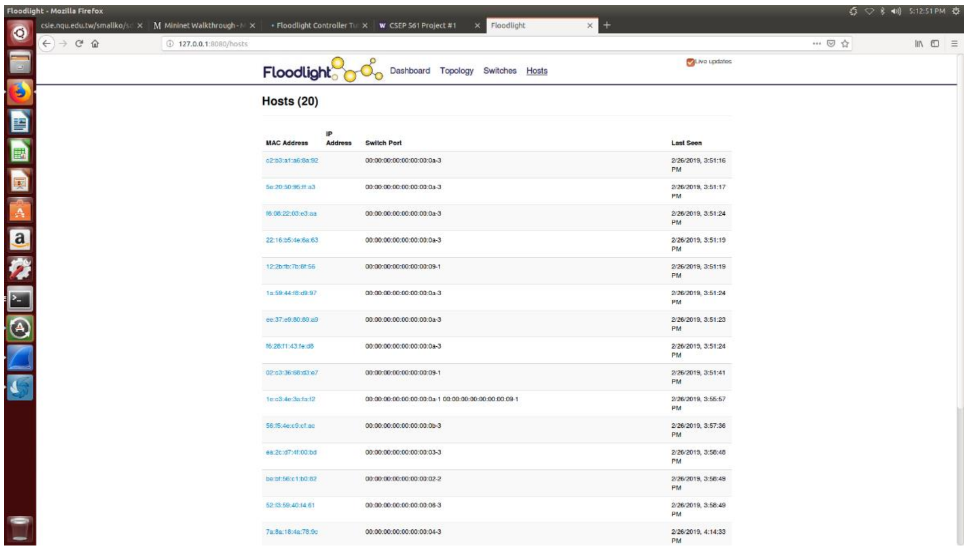


Switches (14)

DPID	IP Address	Vendor	Packets	Bytes	Flows	Connected Since
00:00:00:00:00:00:00:02	/127.0.0.1:55646	Nicira, Inc.	0	0	0	2/26/2019, 3:48:55 PM
00:00:00:00:00:00:00:04	/127.0.0.1:55658	Nicira, Inc.	0	0	0	2/26/2019, 3:48:56 PM
00:00:00:00:00:00:00:0f	/127.0.0.1:55662	Nicira, Inc.	0	0	0	2/26/2019, 3:48:57 PM
00:00:00:00:00:00:00:0c	/127.0.0.1:55678	Nicira, Inc.	0	0	0	2/26/2019, 3:49:07 PM
00:00:00:00:00:00:00:08	/127.0.0.1:55648	Nicira, Inc.	0	0	0	2/26/2019, 3:48:54 PM
00:00:00:00:00:00:00:0a	/127.0.0.1:55642	Nicira, Inc.	0	0	0	2/26/2019, 3:48:55 PM
00:00:00:00:00:00:00:06	/127.0.0.1:55674	Nicira, Inc.	0	0	0	2/26/2019, 3:49:07 PM
00:00:00:00:00:00:00:0b	/127.0.0.1:55672	Nicira, Inc.	0	0	0	2/26/2019, 3:49:07 PM
00:00:00:00:00:00:00:0d	/127.0.0.1:55650	Nicira, Inc.	0	0	0	2/26/2019, 3:48:55 PM
00:00:00:00:00:00:00:07	/127.0.0.1:55660	Nicira, Inc.	0	0	0	2/26/2019, 3:48:56 PM
00:00:00:00:00:00:00:03	/127.0.0.1:55656	Nicira, Inc.	0	0	0	2/26/2019, 3:48:55 PM
00:00:00:00:00:00:00:0e	/127.0.0.1:55676	Nicira, Inc.	0	0	0	2/26/2019, 3:49:07 PM
00:00:00:00:00:00:00:09	/127.0.0.1:55652	Nicira, Inc.	0	0	0	2/26/2019, 3:48:55 PM
00:00:00:00:00:00:00:01	/127.0.0.1:55654	Nicira, Inc.	0	0	0	2/26/2019, 3:48:56 PM

Floodlight © Big Switch Networks, IBM, et. al. Powered by Backbone.js, Bootstrap, jQuery, D3.js, etc.

Figure 39: Floodlight GUI switches

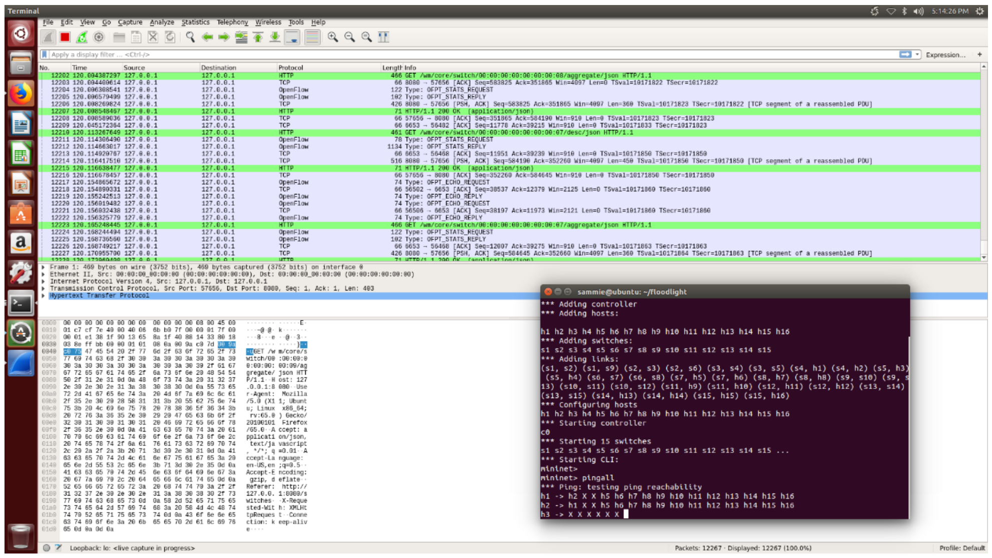


Hosts (20)

MAC Address	IP Address	Switch Port	Last Seen
c2:b3:a1:a6:8a:92		00:00:00:00:00:00:00:0a-3	2/26/2019, 3:51:16 PM
5e:20:50:96:ff:a3		00:00:00:00:00:00:00:0a-3	2/26/2019, 3:51:17 PM
f6:08:22:03:e3:aa		00:00:00:00:00:00:00:0a-3	2/26/2019, 3:51:24 PM
22:16:b5:4e:6a:63		00:00:00:00:00:00:00:0a-3	2/26/2019, 3:51:19 PM
12:2b:fb:7b:8f:56		00:00:00:00:00:00:00:09-1	2/26/2019, 3:51:19 PM
1a:59:44:f8:d9:97		00:00:00:00:00:00:00:0a-3	2/26/2019, 3:51:24 PM
ee:37:e9:80:89:a9		00:00:00:00:00:00:00:0a-3	2/26/2019, 3:51:23 PM
f6:28:f1:43:fe:d8		00:00:00:00:00:00:00:0a-3	2/26/2019, 3:51:24 PM
02:c3:36:68:d3:e7		00:00:00:00:00:00:00:09-1	2/26/2019, 3:51:41 PM
1e:c3:4e:3a:fa:f2		00:00:00:00:00:00:00:0a-1 00:00:00:00:00:00:00:09-1	2/26/2019, 3:55:57 PM
56:f5:4e:c9:cf:ac		00:00:00:00:00:00:00:0b-3	2/26/2019, 3:57:36 PM
ea:2c:d7:4f:00:bd		00:00:00:00:00:00:00:03-3	2/26/2019, 3:58:48 PM
be:bf:56:c1:b0:82		00:00:00:00:00:00:00:02-2	2/26/2019, 3:58:49 PM
52:f3:59:40:f4:61		00:00:00:00:00:00:00:06-3	2/26/2019, 3:58:49 PM
7a:8a:18:4a:78:9c		00:00:00:00:00:00:00:04-3	2/26/2019, 4:14:33 PM

Figure 40: Floodlight GUI Hosts

✓ To check if the topology is working or not:



```
No.   Time          Source     Destination Protocol Length Info
12202 120.004387297 127.0.0.1  127.0.0.1   HTTP      466 GET /wm/core/switch/00:00:00:00:00:00:00:08/aggregate/json HTTP/1.1
12203 120.004400614 127.0.0.1  127.0.0.1   TCP        66 8080 → 57656 [ACK] Seq=583825 Ack=351865 Win=4097 Len=0 TSval=10171822 TSecr=10171822
12204 120.006308541 127.0.0.1  127.0.0.1   OpenFlow  122 Type: OFPT_STATS_REQUEST
12205 120.006579499 127.0.0.1  127.0.0.1   OpenFlow  102 Type: OFPT_STATS_REPLY
12206 120.008269824 127.0.0.1  127.0.0.1   TCP       426 8080 → 57656 [PSH, ACK] Seq=583825 Ack=351865 Win=4097 Len=360 TSval=10171823 TSecr=10171822 [TCP segment of a reassembled PDU]
12207 120.008548467 127.0.0.1  127.0.0.1   HTTP       71 HTTP/1.1 200 OK  (application/json)
12208 120.008589036 127.0.0.1  127.0.0.1   TCP        66 57656 → 8080 [ACK] Seq=351865 Ack=584190 Win=910 Len=0 TSval=10171823 TSecr=10171823
12209 120.045172364 127.0.0.1  127.0.0.1   TCP        66 6653 → 56482 [ACK] Seq=11778 Ack=39215 Win=910 Len=0 TSval=10171833 TSecr=10171823
12210 120.113267649 127.0.0.1  127.0.0.1   HTTP      461 GET /wm/core/switch/00:00:00:00:00:00:00:07/desc/json HTTP/1.1
12211 120.114306490 127.0.0.1  127.0.0.1   OpenFlow   78 Type: OFPT_STATS_REQUEST
12212 120.114663017 127.0.0.1  127.0.0.1   OpenFlow 1134 Type: OFPT_STATS_REPLY
12213 120.114920767 127.0.0.1  127.0.0.1   TCP        66 6653 → 56468 [ACK] Seq=11951 Ack=39239 Win=910 Len=0 TSval=10171850 TSecr=10171850
12214 120.116417510 127.0.0.1  127.0.0.1   TCP       516 8080 → 57656 [PSH, ACK] Seq=584190 Ack=352260 Win=4097 Len=450 TSval=10171850 TSecr=10171850 [TCP segment of a reassembled PDU]
12215 120.116638477 127.0.0.1  127.0.0.1   HTTP       71 HTTP/1.1 200 OK  (application/json)
12216 120.116678457 127.0.0.1  127.0.0.1   TCP        66 57656 → 8080 [ACK] Seq=352260 Ack=584645 Win=910 Len=0 TSval=10171850 TSecr=10171850
12217 120.154865672 127.0.0.1  127.0.0.1   OpenFlow   74 Type: OFPT_ECHO_REQUEST
12218 120.154890331 127.0.0.1  127.0.0.1   TCP        66 56502 → 6653 [ACK] Seq=38537 Ack=12379 Win=2125 Len=0 TSval=10171860 TSecr=10171860
12219 120.155242513 127.0.0.1  127.0.0.1   OpenFlow   74 Type: OFPT_ECHO_REPLY
12220 120.156019482 127.0.0.1  127.0.0.1   OpenFlow   74 Type: OFPT_ECHO_REQUEST
12221 120.156032438 127.0.0.1  127.0.0.1   TCP        66 56506 → 6653 [ACK] Seq=38197 Ack=11973 Win=2121 Len=0 TSval=10171860 TSecr=10171860
12222 120.156325779 127.0.0.1  127.0.0.1   OpenFlow   74 Type: OFPT_ECHO_REPLY
12223 120.165248445 127.0.0.1  127.0.0.1   HTTP      466 GET /wm/core/switch/00:00:00:00:00:00:00:07/aggregate/json HTTP/1.1
12224 120.168244494 127.0.0.1  127.0.0.1   OpenFlow  122 Type: OFPT_STATS_REQUEST
12225 120.168736560 127.0.0.1  127.0.0.1   OpenFlow  102 Type: OFPT_STATS_REPLY
12226 120.168749217 127.0.0.1  127.0.0.1   TCP        66 6653 → 56468 [ACK] Seq=12007 Ack=39275 Win=910 Len=0 TSval=10171863 TSecr=10171863
12227 120.170955700 127.0.0.1  127.0.0.1   TCP       426 8080 → 57656 [PSH, ACK] Seq=584645 Ack=352660 Win=4097 Len=360 TSval=10171864 TSecr=10171863 [TCP segment of a reassembled PDU]

▸ Frame 1: 469 bytes on wire (3752 bits), 469 bytes captured (3752 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 57656, Dst Port: 8080, Seq: 1, Ack: 1, Len: 403
▸ Hypertext Transfer Protocol

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
0010  01 c7 cf 7e 40 00 40 06  6b b0 7f 00 00 01 7f 00   ···~@·@· k·······
0020  00 01 e1 38 1f 90 13 65  8a 1f 40 88 14 33 80 18   ···8···e ··@··3··
0030  03 8e ff bb 00 00 01 01  08 0a 00 9a c0 7d 00 9a   ········ ·····}··
0040  c0 75 47 45 54 20 2f 77  6d 2f 63 6f 72 65 2f 73   ·uGET /w m/core/s
0050  77 69 74 63 68 2f 30 30  3a 30 30 3a 30 30 3a 30   witch/00 :00:00:0
0060  30 3a 30 30 3a 30 30 3a  30 30 3a 30 39 2f 61 67   0:00:00: 00:09/ag
0070  67 72 65 67 61 74 65 2f  6a 73 6f 6e 20 48 54 54   gregate/ json HTT
0080  50 2f 31 2e 31 0d 0a 48  6f 73 74 3a 20 31 32 37   P/1.1··H ost: 127
0090  2e 30 2e 30 2e 31 3a 38  30 38 30 0d 0a 55 73 65   .0.0.1:8 080··Use
00a0  72 2d 41 67 65 6e 74 3a  20 4d 6f 7a 69 6c 6c 61   r-Agent:  Mozilla
00b0  2f 35 2e 30 20 28 58 31  31 3b 20 55 62 75 6e 74   /5.0 (X1 1; Ubunt
00c0  75 3b 20 4c 69 6e 75 78  20 78 38 36 5f 36 34 3b   u; Linux  x86_64;
00d0  20 72 76 3a 36 35 2e 30  29 20 47 65 63 6b 6f 2f    rv:65.0 ) Gecko/
00e0  32 30 31 30 30 31 30 31  20 46 69 72 65 66 6f 78   20100101  Firefox
00f0  2f 36 35 2e 30 0d 0a 41  63 63 65 70 74 3a 20 61   /65.0··A ccept: a
0100  70 70 6c 69 63 61 74 69  6f 6e 2f 6a 73 6f 6e 2c   pplicati on/json,
0110  20 74 65 78 74 2f 6a 61  76 61 73 63 72 69 70 74    text/ja vascript
0120  2c 20 2a 2f 2a 3b 20 71  3d 30 2e 30 31 0d 0a 41   , */*; q =0.01··A
0130  63 63 65 70 74 2d 4c 61  6e 67 75 61 67 65 3a 20   ccept-La nguage: 
0140  65 6e 2d 55 53 2c 65 6e  3b 71 3d 30 2e 35 0d 0a   en-US,en ;q=0.5··
0150  41 63 63 65 70 74 2d 45  6e 63 6f 64 69 6e 67 3a   Accept-E ncoding:
0160  20 67 7a 69 70 2c 20 64  65 66 6c 61 74 65 0d 0a    gzip, d eflate··
0170  52 65 66 65 72 65 72 3a  20 68 74 74 70 3a 2f 2f   Referer:  http://
0180  31 32 37 2e 30 2e 30 2e  31 3a 38 30 38 30 2f 73   127.0.0. 1:8080/s
0190  77 69 74 63 68 65 73 0d  0a 58 2d 52 65 71 75 65   witches· ·X-Reque
01a0  73 74 65 64 2d 57 69 74  68 3a 20 58 4d 4c 48 74   sted-Wit h: XMLHt
01b0  74 70 52 65 71 75 65 73  74 0d 0a 43 6f 6e 6e 65   tpReques t··Conne
01c0  63 74 69 6f 6e 3a 20 6b  65 65 70 2d 61 6c 69 76   ction: k eep-aliv
01d0  65 0d 0a 0d 0a                                     e····

Loopback: lo: <live capture in progress>        Packets: 12267 · Displayed: 12267 (100.0%)        Profile: Default
```

```
sammie@ubuntu: ~/floodlight
*** Adding controller
*** Adding hosts:

h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15
*** Adding links:
(s1, s2) (s1, s9) (s2, s3) (s2, s6) (s3, s4) (s3, s5) (s4, h1) (s4, h2) (s5, h3)
 (s5, h4) (s6, s7) (s6, s8) (s7, h5) (s7, h6) (s8, h7) (s8, h8) (s9, s10) (s9, s
13) (s10, s11) (s10, s12) (s11, h9) (s11, h10) (s12, h11) (s12, h12) (s13, s14)
(s13, s15) (s14, h13) (s14, h14) (s15, h15) (s15, h16)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
*** Starting controller
c0
*** Starting 15 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 ...
*** Starting CLI:
mininet>
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X X h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
h2 -> h1 X X h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
h3 -> X X X X X X
```

Figure 41: Wireshark Capture for connectivity test

8. DIFFERENCES BETWEEN FLOODLIGHT AND POX

For the two controllers implemented, the chosen criteria, for differentiating their performance, concern Southbound and Northbound communication, OpenFlow and OpenStack support, programming language, GUI, documentation and others.

Attribute	POX	Floodlight
Year	2011	2013
GUI	Python+QT4	Web based Java
Programming Language	C++, Python	Java, Python
Platform Support	Linux, Mac, Windows	Linux, Mac, Windows
OpenStack Support	No	No
Southbound API	OpenFlow 1.0	OpenFlow 1.0, 1.3
Northbound API	REST API	JSON/Rest API
Centralised/Distributed	Centralised	Centralised
Multithreading Support	No	Yes
Documentation	Poor, not updated	Medium
Owned by	Nicira	Big Switch Networks

Table 1: Differences between Floodlight and POX

9. SECURITY

Security is one haul stop to protect and maintain the integrity of design and information contained in any system, infrastructure, service or organisation. With the increasing complexity of the technology, so has the sophistication of the attacks increased.

9.1 SDN VULNERABILITIES

SDN relies greatly on the administrator to ensure that the entire network has been programmed to function correctly and effectively while maintaining the security. Weak security measures can compromise networks and that can be exploited to extract sensitive information, bring the entire network structure down or even have targeted attacks.

With the design of the SDN based networks, in operational mode, any unmatched packet is sent to the controller and this is one of the reasons why barriers for sophisticated attacks in SDN is low, even with TLS authentication between switch and the controller. This could bring in malicious switches and hosts packet spoofing to corrupt the controller state. This issue is one of the biggest vulnerabilities in the Data Plane.

Also, SDN might be affected by traditional network attacks and even more since, in traditional networks, switches can make decisions on their own but since in SDN, we have separated the control and data plane and OpenFlow mandates sending unmatched packets from switches to the controller, opening possibilities of malicious hosts tampering with SDN vulnerabilities.

Programmable soft switches such as Open vSwitches are a soft target for attackers as well. The end hosts can start control plane flooding which can saturate controller in terms of their network bandwidth and bring the entire network down.

The hosts can tamper with network topologies by forging packets that could go from switches to controller since they didn’t match and/or implement denial of service and/or extract flow rule information and/or traffic hijacking or re-routing.

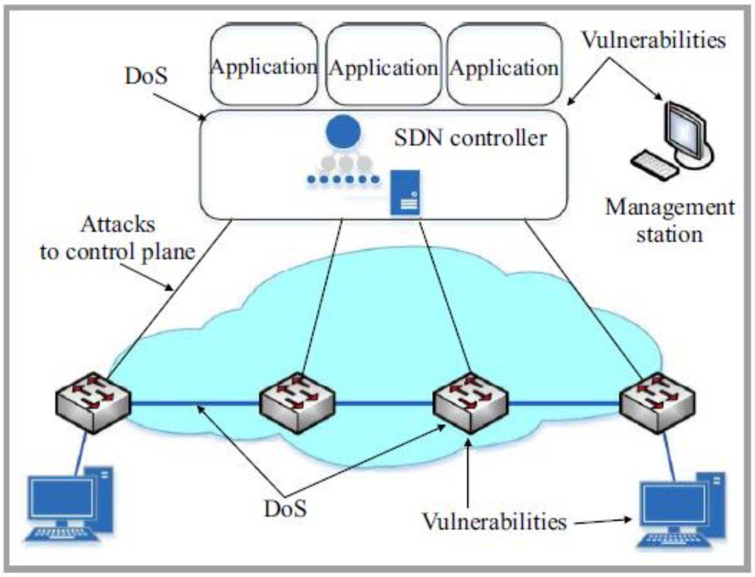


Figure 42: Security Vectors in SDN

9.2 VIRTUALIZATION & SECURITY CHALLENGES

In SDN, virtualisation, acting as an enabler, provides one with underlying network resources and every configuration is stored in a virtual image, in form of a file. Thus, virtualisation in SDN brings vulnerabilities, with its own set of pros.

Some of the vulnerabilities can be:

- ✓ Hypervisor attacks
- ✓ Guest operating system attacks or misconfiguration
- ✓ Inter-VM attacks

Hypervisor attacks and security: Since Hypervisor is responsible for creating and maintaining Virtual Images, there's a risk that it may allow the ability to modify/view operational and functional state of the images, including the possibility of HYPERVISOR HIJACKING.

The attacker is in full control of the hypervisor and can access all VMs and/or other hypervisors in the same infrastructure. With that possibility, not far lies the idea of misconfiguration of SDN controllers to play around with traffic.

Compatibility, configuration and trust relations amongst different hypervisor vendors could be another issue. Also, errors/bugs and misconfiguration on the part of the administration can allow an attacker to compromise the network easily and for even more serious attacks.

9.2.1 POSSIBLE SOLUTIONS

- **Virtual machine guest hardening**: Some of the ways to harden VMs and ensure their isolation could be:
    - Setting limits while resource allocation (reserves) for each VM can protect other VMs from performance degradation in case one of the VMs on the host is facing any attack, DDoS attacks in particular. Doing so, the limited shared resources don't interrupt other virtual machines.
    - Applying standard infrastructure security measures in VM infrastructure such as malware filters, IDS/IPS, firewalls and keeping them updated on a regular basis can help minimise general security risks.
    - Native Management services such as terminal services or ssh can be used to access VMs and manage the operational state. This would reduce the possibility of attacks via VM consoles that can help attackers to bring down the virtual machines.
- **Hypervisor Security:** Some of the ways to ensure security of the hypervisor are:
    - Thin hypervisors are OS independent hypervisors with minimal overheads and can limit the ways malicious code can. It mostly checks for digital signatures to ensure malware doesn't reach the system's internal.
    - Regularly updating and patching the system, including firewalls and active directory integration could enhance Hypervisor security.
    - Strong log-in credentials can ensure management tools security. Also, system roles configuration can help isolate system settings from regular users.
- Regular encryption of VM data.
- While transferring data or destroying VMs, one should ensure that no data is left behind on the disk which could later be recovered. This could be ensured via few techniques such as storage encryption and or Zeroing memory.

- Isolation in terms of traffic (VLANs), address space, performance and control can be implemented more strictly to ensure corruption of one VM won't affect the other.

9.3 NETWORK TOPOLOGY

Various kinds of protocol implemented packets are sent by switches to controllers to have the network topology and this can be used by compromised hosts to spoof IGMP messages used for multicast groups and then temper with controller's view of topology along with installing self implemented flow rules to launch number of attacks on the network.

9.3.1 TRADITIONAL ATTACKS AND HOW THEY MANIFEST IN SDN

SDN is based on the foundations from the traditional network and hence while using entities of the latter, same attacks can be triggered in SDN as well. And for preventive measures, it might or might not work or could be extended to the former since SDN switches work differently than the traditional switches.

For example, traditional network switches have verification via authentication against spoofing using cryptographic mechanisms which might be heavy. In SDN, even with TLS security levels, any packet that isn't part of any rule would be directed to the controller automatically and hence can result in a fake topology attack.

9.3.2 SDN SECURITY (SDSEC)

Traditional security mechanisms aren't the best way to deal with virtualised environments and hence SDSec approaches the design, deployment and management of security in a new way by separating processing and forwarding plane, like separating control and data plane in operational state.

This separation is useful as it gives a distributed security solution which is dynamic and virtualises security functions and provides a way to manage them as a logical, single system.

SDSec mostly replaces security hardware appliances like IDS/IPS, firewalls with software functions. Most of them have a control center at the middle of the network for policy enforcement and ensure security controls are distributed across Virtual Machine Appliances.

9.3.3 CONTROL PLANE AND VULNERABILITIES

Control Plane could face the following VULNERABILITIES:

- ✓ Centralised controller could be a ‘single point of failure’
- ✓ Communication interfaces
- ✓ Policy enforcement: As the size of the routing information grows, so will the responsibility of the advertising of the paths for reachability to destination and not only in case of between the local instances of the data plane but also administration.
- ✓ Dynamic flow rule modification
- ✓ Controller- switch communication flood
- ✓ System level security challenges
- ✓ Trust between controller and third-party applications
- ✓ Malicious SDN controller modules: SDN controller is the brain of the network which helps running services and application. Malicious modules can add dubious functionalities which can prevent an entire network to function properly and put data in danger.

MEASURES:

- ✓ Installation of security applications or authentication systems on Northbound Interface can be used to prevent unauthorised access to controller. Another way could be role-based authorization and access control.
- ✓ Access lists can be used to filter the traffic reaching the controller.
- ✓ Malicious SDN controller modules can be prevented/ taken care of using SDN controllers that coordinate tasks to select a trusted configuration and keep a check on their modules for the standards.
- ✓ Heterogeneous network topologies can be adopted to survive disruptions and attacks.

- ✓ Artificial Intelligence, neural networks and data mining techniques can be used to solve routing and optimisation problems in the dynamic environment.
- ✓ Access control can be defined by using Controller as the policy enforcer and basing the policies on a role hierarchy where administrators assign roles to SDN applications.
- ✓ Operations can be identified and then controlled via permissions to be able to implement the minimum privilege principle that guides operational state with an authority that authenticates and the caller and checks if they have access to a critical operation or not.

INDUSTRY USED SOLUTIONS:

- ✓ **Security Enhanced-Floodlight:** BigSwitch Floodlight Controller Extension
  Providing a role-based authorization system, this extension is one of its first implementation of an SDN based security policy in OF protocol stack.
- ✓ **Security Actuator**: OF Security Directive actuation service
  It helps enable network security tools that can start advanced security resolve logic and thus rewrite network flow paths for attacked and infected hosts.

9.3.4 DATA PLANE AND VULNERABILITIES

Data plane can suffer from various security threats, namely:

- ✓ Malicious switches and hosts: Malicious hosts as well as switches can send bad requests which can exhaust resources of the switches/controller and result in DoS flooding in the network.
- ✓ Flow rule discovery
- ✓ Flooding attacks
- ✓ Forged traffic: Communication between Controller and End devices is vulnerable and could be taken advantage of by spoofing the device flow table and/or forging new traffic rules or injecting/changing conversations, called as Man-in-the-middle-attack.
- ✓ Credential management

MEASURES:

- ✓ SDN SBI (Southbound Interface) and Protocols could use OpenFlow/Open vSwitch Database Management Protocol or BGP-LS or SNMP as these have their own algorithms to secure the network end devices.
- ✓ Regular internal or external audits can be put in place to check configurations and flow rules regularly.
- ✓ Regular checks by the administration for intra-switch misconfiguration within single flow-table.

INDUSTRY USED SOLUTIONS:

- ✓ **OpenFlow switches:** It divides the network into small logical networks that can allow users to use applications without affecting each other i.e. isolation. One of the most useful switches to test and implement new experiments since it supports encapsulation and encryption amongst many other options.
- ✓ **FlowChecker:** Centralised server application

  It receives queries from OF applications and that can include verification/analysis/debugging configurations. It can help verify consistency of switches and validate the correctness of the flow tables with the services and protocols in place.

9.3.5 APPLICATION PLANE AND VULNERABILITIES

Application Plane can suffer from vulnerabilities such as:

- ✓ Un-authorised applications or users
- ✓ Potential trust issues because of third party
- ✓ Fraudulent role insertion
- ✓ Lack of authentication methods
- ✓ Lack of secure provisioning

MEASURES:

✓ A granular permission system with OpenFlow specific permissions can be set up which can check for the potential caller and what all they can access.

✓ AI/Machine Learning can help build ways that can dynamically analyse controller program and can help channelize delays in input and receiving outputs.

✓ Threat detection and security monitoring systems like IDS/IPS, firewalls can be installed with regular patching and updating to ensure basic security.

INDUSTRY USED SOLUTIONS:

✓ **Procera**: Computational language

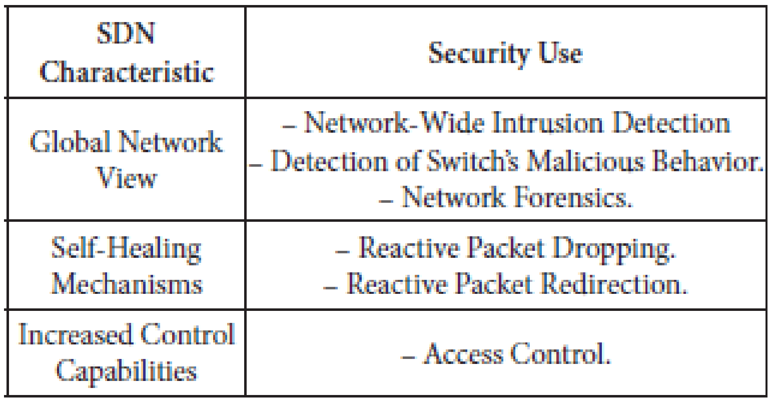
Procera can be used for defining high-level network policies which can be used to program how to react with a dynamic change in the network.

✓ **Flover**: Verification Tool

It can convert flow table into understandable format and detect anomalies for network security.

✓ **OFTesting:** Python based OF application

It can be used for debugging and automated testing of OF programs. Keeping server updated could also help avoid application manipulation



SDN Characteristic	Security Use
Global Network View	– Network-Wide Intrusion Detection – Detection of Switch's Malicious Behavior. – Network Forensics.
Self-Healing Mechanisms	– Reactive Packet Dropping. – Reactive Packet Redirection.
Increased Control Capabilities	– Access Control.

Figure 43: SDN & Security

10. DENIAL OF SERVICE(DoS)

DoS attack is a means of shutting down a machine/network, making it inaccessible to legitimate users. DoS attacks generally have few popular methods, namely: *Buffer Overflow Attacks, ICMP flood* and *SYN flood.*

Buffer Overflow: It's the most common DoS attack. Via this method, traffic more than the capacity of the server's buffer is sent.

ICMP flood: It leverages misconfigured network devices by sending spoofed messages. These are used to ping targeted hosts. This attack is also called smurf attack or ping of death.

SYN flood: In this the attacker sends a request to connect to the server but never completes the TCP handshake, which leads to open port and resources bind to that port. The main part is that multiple requests from multiple hosts is sent that can easily crash a system.

10.1 SYN FLOODING

TCP-SYN Flooding is one of the most popular amongst all DoS attacks which exploit the TCP vulnerability on the Web Server side.

SYN flood is a type of denial-of-service attack which exploits the TCP three-way handshake and once successful can consume resources on the server and render it unresponsive to other clients.

During SYN flooding, huge quantities of TCP packets with only SYN flag set are sent to the server. SYN flags are usually the first part of the three-way handshake which are responded by the server using SYN-ACK packet.

Since web server is dependent on the TCP for it is the underlying transport protocol, it keeps connections open or half-open until the final ACK arrives from the client side.

So, attackers usually open many incomplete connections, depleting SYN-Queue which will ultimately deny or delays legitimate connection requests. Thus, any network service associated with that TCP socket goes down.

This large quantity can clog bandwidth, further leading to resource depletion or worse crashing of the server, leading to no services to any clients.

Some of the known measures for protection against SYN flooding are:

- ✓ SYN cookies, SYN cache and SYN proxy
- ✓ Updated Firewall through Access Control Lists
- ✓ IDS/IPS with signature-based mechanisms

INDUSTRY USED SOLUTIONS:

- ✓ **AVANT-GUARD**: It is an extension of the data plane with two modules, namely: *connection migration module* and *actuating trigger module.* Connection migration adds intelligence to data plane and helps in differentiating sources that would complete TCP handshake from those who won't. The ones who complete the handshake are further exposed to the control plane. Access tables collaborate with the module and maintain TCP session information to provide session details later to the control plane.

  Connection migration module has a four-stage operational stage:

  a) Classification: CM engages the client in a stateless TCP handshake using SYN cookies and on completion, the client moves to the report stage.

  b) Report: CM determines if the client is anyway associated with any entry in the flow table. If not, the entry is passed on to the control plane else it's passed on to migration stage.

  c) Migration: CM initiates a TCP connection with the client's destination host and if it responds, a successful connection is established.

  d) Relay: After a successful TCP connection is established, CM module enters relay stage and relays all TCP data packets between client and destination as normal TCP session.

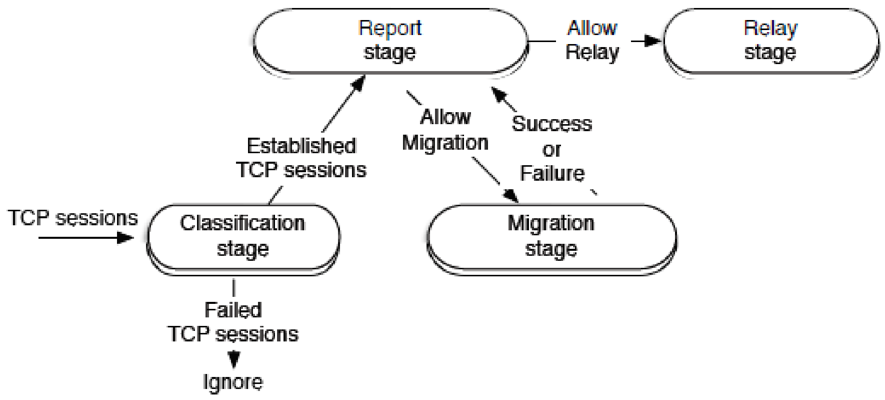


Figure 44: Connection Migration Module Stages

Actuating trigger module collects network status information and packet payload. It offers conditional flow rule activation that is the ability to activate flow rules when any event occurs.

10.2 DDoS

Distributed Denial of Service attacks are an attempt to make a machine/network resource unavailable to intended users. These could be initiated by two or more sources or bots. A bot is a device used to penetrate a computed by software from a malware code.

DDoS attacks can be divided into two types based on their targeted protocols:

- ✓ Network/transport-level: Mostly launched using TCP/UDP/ICMP/DNS protocol packets and they focus on dropping legitimate requests as they exhaust network resources.
- ✓ Application-level: These aim to exhaust server resources i.e. sockets, CPU, memory or I/O bandwidth.

In SDN, possible DDoS attacks can be in three categories:

- ✓ Application layer: DDoS attack can either be on the launched application or an attack on the Northbound API. And since isolation of application via resources is a weak link, crashing of one application can affect the others.
- ✓ Control layer: Controller in SDN is a single point of failure risk, so it attracts many attack possibilities. Ways to attack Control layer could include via controller, Northbound API, southbound API, westbound API or eastbound API.
- ✓ Infrastructure layer: One can attack Infrastructure layer either via switches or via Southbound API.

Some of the methods that could help SDN detect and comprehend DDoS attacks could be:

- ✓ SDN has a centralised logical controller which can view the network and that can dynamically quarantine compromised hosts and authenticate legitimate hosts based on the traffic patterns/flow entries/authentication and other implemented security measures.
- ✓ Intelligence from existing IDS/IPS systems can be harnessed with SDN and flexible system can be built up to detect DDoS attacks.
- ✓ Software based traffic analysis using neural networks and AI/ML can be performed which can help predict DDoS attacks and with help of IDS/IPS signatures can be set up to alert/ prevent those attacks.
- ✓ DDoS can also be prevented using packet dropping and rate limiting techniques.
- ✓ SDN controllers can be set up at ingress traffic that can detect anomaly traffic, filter malicious packets or validate source IP and then allow it further into the network.
- ✓ A Flow collector module can be setup which could periodically request flow entries from all flow tables and extract features like average of packets per flow, percentage of pair-flows etc. for DDoS flooding attack detection.

INDUSTRY USED SOLUTIONS:

- ✓ **Fresco**: An OF security application development framework, it is used for different detection and migration modules. It is useful in protecting network against detected threat by constraining flow of data.
- ✓ **Fortnox:** It enables administration to define and implement strict network policies which can override dynamic derived flow rules. It also supports OF application authentication with the help of digital signatures.
- ✓ **Onix:** Platform that helps implement control plane as a distributed system. Onix provides an API which allows control plane to make their trade-offs based on consistency, durability and scalability amongst many.

CURRENTLY IN RESEARCH

- ✓ **Cross-Layer Traffic Analysis:** Cross-layer traffic analysis helps looking at information at multiple protocol layer to detect/respond to DDoS. With an emphasis on L2-L4, SDN can focus and extend traffic intelligence to L4-L7.
- ✓ **Multiple Locations Defensive:** This method uses deployment of multiple defence nodes at locations such as source, destination or networks. This deployment can help detection at the sites it is placed. Also, the nodes initiate and distribute information to other nodes.

11. CONCLUSIONS

From the Capstone project, I can conclude on the following learnings:

- The basis of SDN as a concept has existed for more than 15 years but due to the increase in network traffic and programmability of networks, we are becoming more aware about it.
- Controller, being the central building block of SDN is necessary yet 'single point-of-failure'.
- Open Source community has had a lot of contribution to the concept building and implementation of SDN.
- SDN controllers vary in many magnitudes. This has both its pros and cons. With handling multiple aspects, we also can face the issue when it comes to things like programming languages, multi-threading and so on.
- Existing research for controller as a subject is yet to fully develop.
- SDN has handled many of traditional architectural issues but also has opened many attack vectors that need to be focused on and improved.

12. REFERENCES

- https://www.citrix.com/products/citrix-adc/resources/sdn-101.html
- https://learning.oreilly.com/library/view/sdn-softwaredefined/9781449342425/ch02.html
- https://www.nojitter.com/4-challenges-lying-wait-sdn
- https://www.sdxcentral.com/networking/sdn/definitions/what-is-openflow/
- https://github.com/mininet/openflow-tutorial/wiki/Learn-Development-Tools
- https://en.wikipedia.org/wiki/Open_vSwitch
- https://github.com/openvswitch/ovs/blob/master/Documentation/intro/why-ovs.rst
- https://github.com/mininet/openflow-tutorial/wiki/Installing-Required-Software
- https://github.com/mininet/openflow-tutorial/wiki
- https://www.opennetworking.org/
- https://github.com/mininet/mininet
- http://mininet.org/download/
- http://www.brianlinkletter.com/set-up-mininet/
- https://ieeexplore-ieee-org.login.ezproxy.library.ualberta.ca/document/7750830
- https://github.com/mininet/mininet/wiki/Introduction-to-Mininet
- https://openflow.stanford.edu/display/ONL/POX+Wiki.html
- http://www.brianlinkletter.com/using-pox-components-to-create-a-software-defined-networking-application/
- http://www.academia.edu/9021253/Simulation_in_an_SDN_network_scenario_using_the_POX_Controller
- https://github.com/mininet/openflow-tutorial/wiki/Learn-Development-Tools
- https://pradeepaphd.wordpress.com/2018/03/06/floodlight-controller-tutorial/
- https://www.semanticscholar.org/paper/SPHINX%3A-Detecting-Security-Attacks-in-Networks-Dhawan-Poddar/4f358dbee87adc305b3e983e5ff6dff1074d3cf6
- https://wikisites.cityu.edu.hk/sites/netcomp/articles/Pages/HardeningStepstoSecureVirtualisationEnvironment-VirtualMachine.aspx

- https://www.ericsson.com/en/ericsson-technology-review/archive/2015/identifying-and-addressing-the-vulnerabilities-and-security-issues-of-sdn
- https://pdfs.semanticscholar.org/4ef3/92fe135271bbfbecf1a1e9735aae80bc85d3.pdf
- https://www.openairinterface.org/?page_id=466
- https://ieeexplore-ieee-org.login.ezproxy.library.ualberta.ca/document/8423699
- https://dl-acm-org.login.ezproxy.library.ualberta.ca/citation.cfm?id=2516684
- https://ieeexplore-ieee-org.login.ezproxy.library.ualberta.ca/document/7289347
- https://www.paloaltonetworks.com/cyberpedia/what-is-a-denial-of-service-attack-dos
- https://searchnetworking.techtarget.com/tip/SDN-security-strategies-for-network-attack-prevention
- https://ieeexplore-ieee-org.login.ezproxy.library.ualberta.ca/document/8389262
- https://www.researchgate.net/publication/304457462_SDN_controllers_A_comparative_study

IMAGES SOURCE

- https://www.researchgate.net/figure/SDN-vs-traditional-network-architecture_fig2_311496181
- 2017_Book_GuideToSecurityInSDNAndNFV.pdf
- https://www.openairinterface.org/?page_id=466
- https://www.udemy.com/https://www.sdxcentral.com/open-source/definitions/what-is-open-vswitch/?c_action=related_articles
- https://www.semanticscholar.org/paper/Mininet-as-Software-Defined-Networking-Testing-KaurSingh/b1c7f8ac477a5553303802bb7785dd3b53372057
- https://learning.oreilly.com/library/view/sdn-software-defined/9781449342425/ch04.html
- https://www.researchgate.net/publication/311496181
- https://www.researchgate.net/publication/311496181