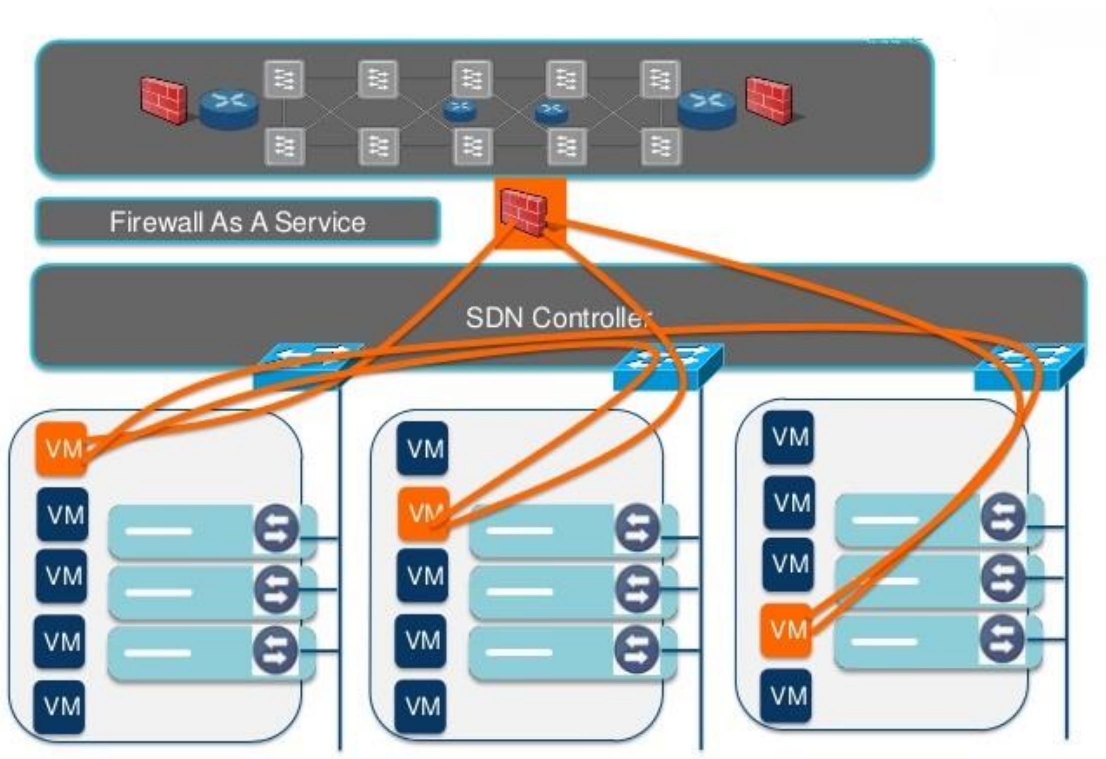
UNIVERSITY OF ALBERTA



SDN- Data Center Automation with Firewall as a Service

MINT 709
Capstone Project Report
Feb, 2016

Submitted By:
Saminderjit Singh Chahal

Supervisor: Muhammad Durrani
Cisco Systems Limited

Submitted To:
Dr. M.H. (Mike) MacGregor
MINT Program Director
&
Mr. Shahnawaz Mir
MINT Program Coordinator

ABSTRACT

The software-defined data center (SDDC) has already redefined the traditional ways of implementing the storage, networking, computing and security services by eliminating the dependency on the dedicated hardware. Enterprise data centres are evolving to provide the higher performance and agility, driven by rapid adoption of mobility/BYOD, big data, cloud, and SaaS. But, the data center security needs to guard against the more sophisticated attacks , while keeping up with rapidly increasing bandwidth requirements.

Software defined networking (SDN) and network virtualization had already been abstracted the networking and security service attributes, while creating the network fabric overlays for on-demand policy orchestration and ensuring the consistent visibility and control. So, the intent of my project is to deploy an orchestrated firewall plugin in multi-tenant data center network in automated manner using Openflow Controller (POX) and Python API. The firewall plugin developed in this project supports the network segmentation and security service function deployment.

ACKNOWLEDGMENTS

I sincerely thank Mr. Muhammad Durrani and Mr. Shahnawaz Mir for their guidance and encouragement in carrying out this project work on this emerging technology and providing me the equipment in the MINT lab required to implement this project.

I also thank the MINT program director Dr. M.H. (Mike) MacGregor for providing me the opportunity to embark on this project.

UNIVERSITY OF ALBERTA

Table of Contents

Abstract 2
Acknowledgments 3
Table of Figures 5
Why we need SDN? 6
History 7
Definition 8
SDN Architecture 9
OpenFlow 10
Discussion 11
Mininet: 11
Why I used Mininet? 12
POX SDN Controller 12
Environment Setup 13
Install Ubuntu or other Linux Operating system 13
Install Git 13
Install Mininet 14
Install POX 15
Install Editor 15
Implementation 16
Use Case: Deploy Tenant Firewall Using SDN Controller (POX) using Python API 19
SDN - DC as a firewall Service 20
Deploy Hierarchical multi-Tenant Data Center network in the automated manner using Mininet Python API infrastructure simulation. 21
Deploy Python based SDN Controller (POX) in layer 2 switch mode and verify end to end connectivity of the hosts. 23
Deploy the firewall plugin in the layer 2 switch controller. 26
TestCases 29
Summary and Conclusion 31
Future Work to do: 31
Bibliography & References: 32

UNIVERSITY OF ALBERTA

TABLE OF FIGURES

Figure 1 History of SDN: Before OpenFlow 7
Figure 2 SDN Architecture 9
Figure 3 OpenFlow Protocol 10
Figure 4 Mininet Installation Confirmation 14
Figure 5 POX Installation 15
Figure 6 Running basic topology in Mininet 16
Figure 7 Checking POX controller 17
Figure 8 Verifying connectivity of hosts for Mininet basic topology 18
Figure 9 Checking the POX console 18
Figure 10 Our Project Scenario 19
Figure 11 Running Mininet for our required topology 21
Figure 12 Checking Hosts Configurations 22
Figure 13 Checking POX connectivity for our required topology 23
Figure 14 Verifying end to end connectivity for required topology 24
Figure 15 Checking flows on POX console 25
Figure 16 CSV file format 26
Figure 17 Firewall Plugin code screenshot 27
Figure 18 Installing Firewall flow rules on switch 28
Figure 19 Checking the Switch Flow Table entries 28
Figure 20 Verifying Firewall Rule Implementation between H1 and H2/H4 29
Figure 21 Verifying end to end connectivity between H1 and H3 29
Figure 22 Verifying Firewall Rule Implementation between H2 and H1/H3 30
Figure 23 Verifying end to end connectivity between H2 and H4 30
Figure 24 Verifying end to end connectivity between H3 and H4 31

WHY WE NEED SDN?

The explosion of mobile devices, server virtualization, and advent of cloud services are among the trends driving the networking industry to re-examine the traditional network architectures. Many conventional networks are hierarchical, built with tiers of Ethernet switches arranged in a tree structure. This design made sense when client-server computing was dominant, but such a static architecture is ill-suited to the dynamic computing and storage needs of today's enterprise data centers, campuses, and carrier environments.

So, today network administrators are in hunt for something that fulfills the scalability issues and provides a reliable high performance services to the clients listed as

1. **Virtualization**: Use network resource without worrying about where it is follows:physically located, how much it is, how it is organised etc.
2. **Orchestration**: Should be able to control and manage thousands of devices with one command
3. **Programmable**: Should change behaviour according to the network's needs.
4. **Dynamic Scaling**: Should be able to change size accordingly.
5. **Automation**: Minimize the manual involvement for doing tasks like troubleshooting, policy enforcement, add new workloads, sites etc.
6. **Performance**: Provides the network device utilization optimization for traffic engineering, failure handling, capacity optimization and high utilization.
7. **Multi-tenancy:** To give complete control to tenants for managing addresses, topology, security etc.
8. **Service Integration:** To provide the load balancer, firewall, intrusion Detection System (IDS) services on demand.

The requirements of above services in modern networks gave birth to a new era of technology named **"Software Defined Networking (SDN)"**.SDN has the potential to revolutionize the legacy data centers by providing a flexible way to control the network according to the needs and provide high performance services.

HISTORY

The origins of software defined networking (SDN) began shortly after when the Java was released by the Sun Microsystems in 1995.The first notable SDN project was AT&T's GeoPlex platform that manages the networks and on-line services.

In 2000, WebSprocket a Silicon Valley startup launched by Mark Medovich, designed a platform in Java that had the ability to instantiate network stack(s), interfaces , protocols as multiple threads and an object oriented structured runtime model that could be modified by a networked compiler/class-loader in real time.

In early 2001, Ericsson and WebSprocket made a license contract and created the first soft switch. OARnet ran the SDN tests successfully over Internet2 in collaboration with Ohio State University. In 2003, Bob Burke and Zac Carman continued the SDN work by developing the Content Delivery Control Network patent application.

Finally, in 2007 Nicira startup focused on software-defined networking and network virtualization was launched by Martino Casado, Nick McKeown and Scott Shenker. Nicira created their own proprietary versions of OpenFlow, Open vSwitch and OpenStack networking projects.

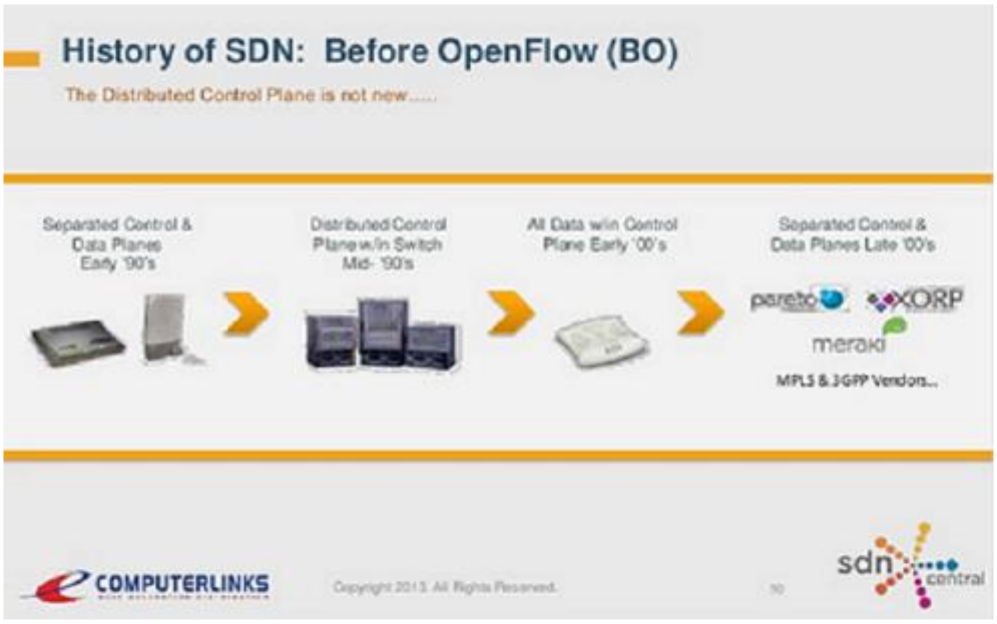

History of SDN: Before OpenFlow (BO)
The Distributed Control Plane is not new.....
Separated Control & Data Planes Early '90's
Distributed Control Plane w/in Switch Mid- '90's
All Data w/in Control Plane Early '00's
Separated Control & Data Planes Late '00's
pareto
XORP
meraki
MPLS & 3GPP Vendors...
COMPUTERLINKS
Copyright 2013 All Rights Reserved.
sdn central

Figure 1 History of SDN: Before OpenFlow

DEFINITION

SDN has many definitions given by the experts:

“**Software Defined Networking (SDN)** is an approach to design, build and manage the networks while separating the network’s control plane and the forwarding planes to better optimize each

Or

“**Software-defined networking** (**SDN**) is an approach to computer-networking that allows network administrators to manage the network services through abstraction of higher-level functionality. This is done by decoupling the system that makes decisions about where traffic is sent (the control plane) from the underlying systems that forward traffic to the selected destination (the data plane).

--Wikipedia

SDN ARCHITECTURE

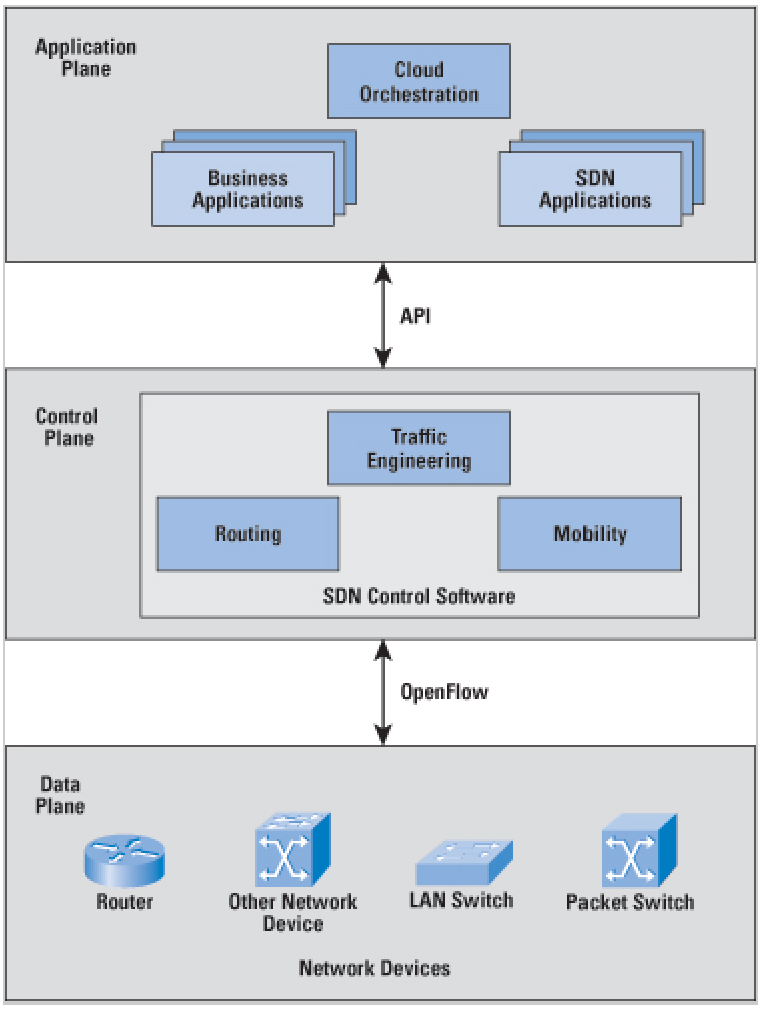


Figure 2 SDN Architecture

The figure describes the logical structure of SDN. SDN architecture has basically three layers:

1. Application Plane: The applications like cloud orchestration, business applications and other SDN applications are deployed in the application plane. The Application plane communicates with the control plane using Application Programming Interfaces (APIs) like REST API or Java API and these APIs are also known as Northbound APIs.
2. Control Plane: A central controller performs all the complex functions like routing, naming, policy declaration and security checks. It basically consists of one or more SDN servers. SDN controller allows the data flows that occur

through SDN Data Plane. If the controller allows a flow, it then computes the route for the flow to take, and then adds a flow entry for that flow in every switch along the path. The control plane communicates with the data plane through various protocols like OpenFlow which talks to the networking devices through open APIs known as Southbound APIs.

3. Data Plane: The main purpose of data plane is to forward the traffic based on the rules installed on it by the control plane. Data plane network devices like switches, routers, firewalls or load balancers simply manage flow tables whose entries can be populated only by the controller.

SDN controllers and the switches can be deployed for Ethernet switches (Layer 2), Internet Routers (Layer 3), transport switching (Layer 4) or application layer functionality.

OPENFLOW

OpenFlow is basically a network protocol which allows the communication between the control and the data plane through open APIs. It is designed to manage and direct the traffic among routers and switches from various vendors. Thus, it differentiates the programming of routers and switched from the underlying hardware.

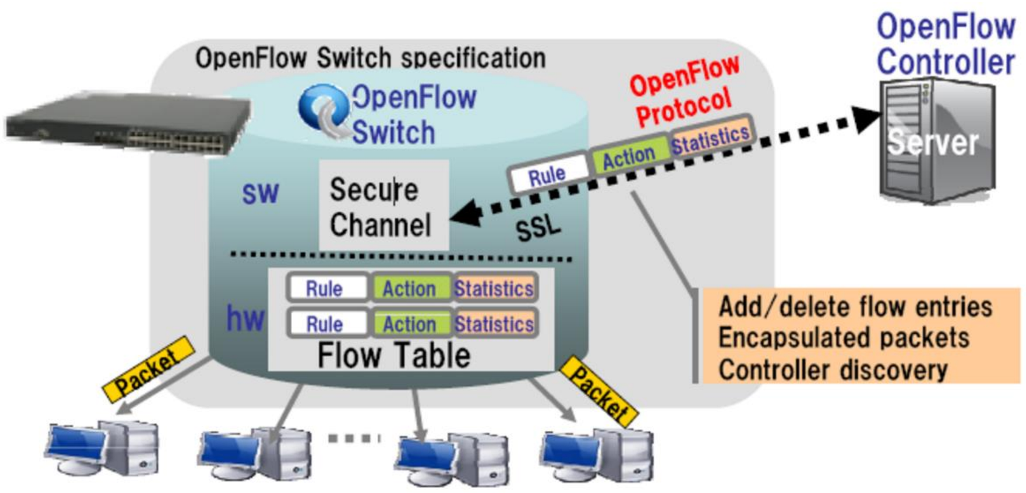


Figure 3 OpenFlow Protocol

Controller communicates with the switch using OpenFlow over the SSH giving instructions to the switch like installing the new rule by describing the action (like send-out-port, modify field or drop action) to take for the particular flow according to the network topology requirements. When a switch see a packet that has no matching flow entries to switch flow table, it simply sends that packet to the controller. The controller then makes the decision on how to handle this packet and installs a flow rule on switch flow table to handle the similar packets in future.

DISCUSSION

MININET:

Mininet is a network emulator used to create the networks of virtual hosts, switches, controllers and links. It runs standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and software-defined networking. The github codebase for Mininet is available at https://github.com/mininet/mininet.git.

Mininet key features are:

- Provides a simple and inexpensive network environment to develop OpenFlow applications
- Enables complex topology testing, without the need to wire up a physical network.
- Easy to debug or running network-wide tests
- Provides an extensible Python API for network creation and experimentation on local machine.
- Supports arbitrary custom topologies.

Mininet networks run real code which includes Unix/Linux network applications as well as the real Linux kernel and network stack. Because of this the code we develop and test on Mininet, for an OpenFlow controller, Open vSwitch or host can be moved to a real system with small amendments, to deploy over the real-world testing and performance evaluation. So, the design I created using Mininet can be moved directly to hardware switches for line-rate packet forwarding.

WHY I USED MININET?

Mininet is totally based on Python, except for a short C utility. It has the following best features among other emulators, hardware testbeds and the simulators:

Compared to other virtualization approaches:

- Boots faster
- Highly Scalable – It supports hundreds of hosts and switches.
- Provides more bandwidth.
- Easy to use and install.

Compared to other hardware testbeds:

- Inexpensive and open source.
- Quickly reconfigurable.

Compared to other simulators:

- Connects easily to real networks
- High performance
- Runs real code including application code, OS kernel code, and control plane code (for both OpenFlow controller code and Open vSwitch code).

POX SDN CONTROLLER

POX is a python based framework used to develop the SDN controllers to communicate with the SDN switches using either the OpenFlow or OVSDB protocol. It is highly used for teaching and doing research on software defined networking and network applications programming.

POX is open source product and can be directly used as a SDN controller because it has stock components bundled with it. The github codebase for POX is available at https://github.com/noxrepo/pox.git.

POX key features are:

- “Pythonic” OpenFlow interface.
- Inbuilt topology discovery.
- Easy to use and open source.
- Compatible with Linux, Mac OS, and Windows environment.
- Performs well compared to other Python based SDN controllers like NOX or Ryu.

ENVIRONMENT SETUP

METHOD 1

INSTALL UBUNTU OR OTHER LINUX OPERATING SYSTEM

We can install Ubuntu on any compute node that can scale to install mininet on Ubuntu with FW plugin and POX controller binaries or install it in Virtual Machine (VirtualBox, VMware , KVM etc).

There are many ways to install Ubuntu, I installed it in Virtual Box from the iso image of Ubuntu available at http://www.ubuntu.com/download/desktop.

INSTALL GIT

Git is a free and open source distributed version control system designed to handle large projects with speed and efficiency. Once the Ubuntu is installed, we can simply install the git with the following commands :

- sudo apt-get install git

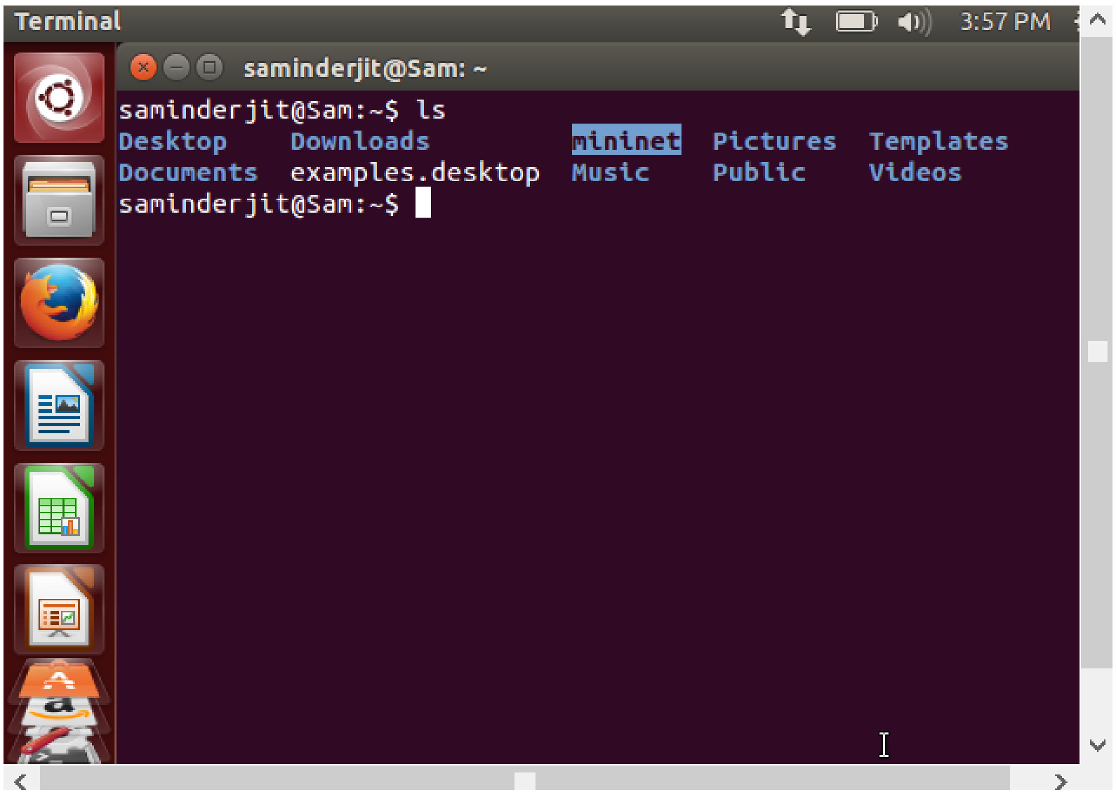
INSTALL MININET

Mininet is a network emulator which can create a network of hosts, links and switches on a single machine. Installing the mininet from github is very easy use the following command from your home directory to install it:

- git clone https://github.com/mininet/mininet.git

The above command will create a new repository named Mininet at your current location and all the mininet binaries are installed in this directory.

To check if mininet is installed successfully type ls in your terminal and check if Mininet is there as follows:



```
saminderjit@Sam:~$ ls
Desktop    Downloads         mininet  Pictures  Templates
Documents  examples.desktop  Music    Public    Videos
saminderjit@Sam:~$ 
```

Figure 4 Mininet Installation Confirmation

If you can see mininet directory as shown in screenshot , it means Mininet is installed successfully on your machine.

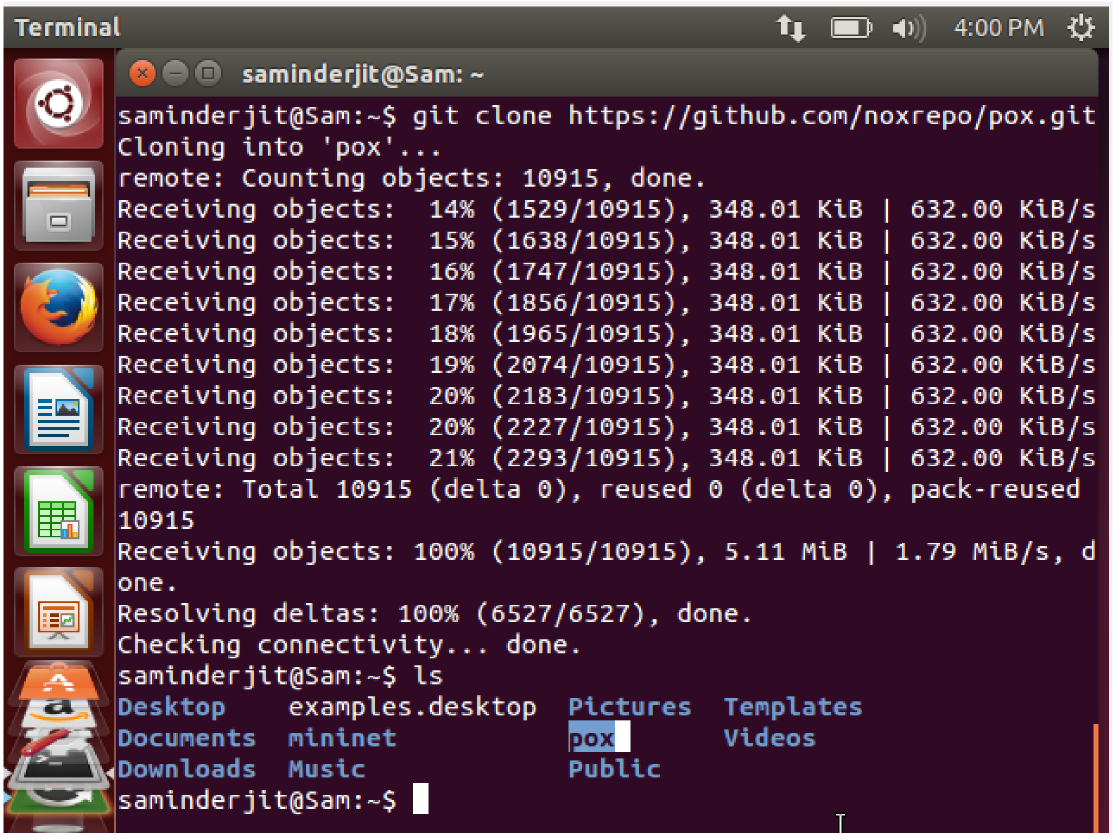
Now go inside the mininet and type "*sudo apt-get install mininet*" to install the mininet default topologies which we use in our project.

INSTALL POX

Similarly, we can easily install the pox from the pox github repository as follows:

- git clone https://github.com/noxrepo/pox.git

This will create a new repository named Pox with all the pox binaries pre-installed in this repository. Type ls to check if you can see the pox directory as shown below:



```
saminderjit@Sam:~$ git clone https://github.com/noxrepo/pox.git
Cloning into 'pox'...
remote: Counting objects: 10915, done.
Receiving objects:  14% (1529/10915), 348.01 KiB | 632.00 KiB/s
Receiving objects:  15% (1638/10915), 348.01 KiB | 632.00 KiB/s
Receiving objects:  16% (1747/10915), 348.01 KiB | 632.00 KiB/s
Receiving objects:  17% (1856/10915), 348.01 KiB | 632.00 KiB/s
Receiving objects:  18% (1965/10915), 348.01 KiB | 632.00 KiB/s
Receiving objects:  19% (2074/10915), 348.01 KiB | 632.00 KiB/s
Receiving objects:  20% (2183/10915), 348.01 KiB | 632.00 KiB/s
Receiving objects:  20% (2227/10915), 348.01 KiB | 632.00 KiB/s
Receiving objects:  21% (2293/10915), 348.01 KiB | 632.00 KiB/s
remote: Total 10915 (delta 0), reused 0 (delta 0), pack-reused
10915
Receiving objects: 100% (10915/10915), 5.11 MiB | 1.79 MiB/s, d
one.
Resolving deltas: 100% (6527/6527), done.
Checking connectivity... done.
saminderjit@Sam:~$ ls
Desktop     examples.desktop  Pictures  Templates
Documents   mininet           pox       Videos
Downloads   Music             Public
saminderjit@Sam:~$
```

Figure 5 POX Installation

INSTALL EDITOR

You can install any editor you want, I installed vim editor as it’s very easy to use.

- sudo apt-get install vim

METHOD 2

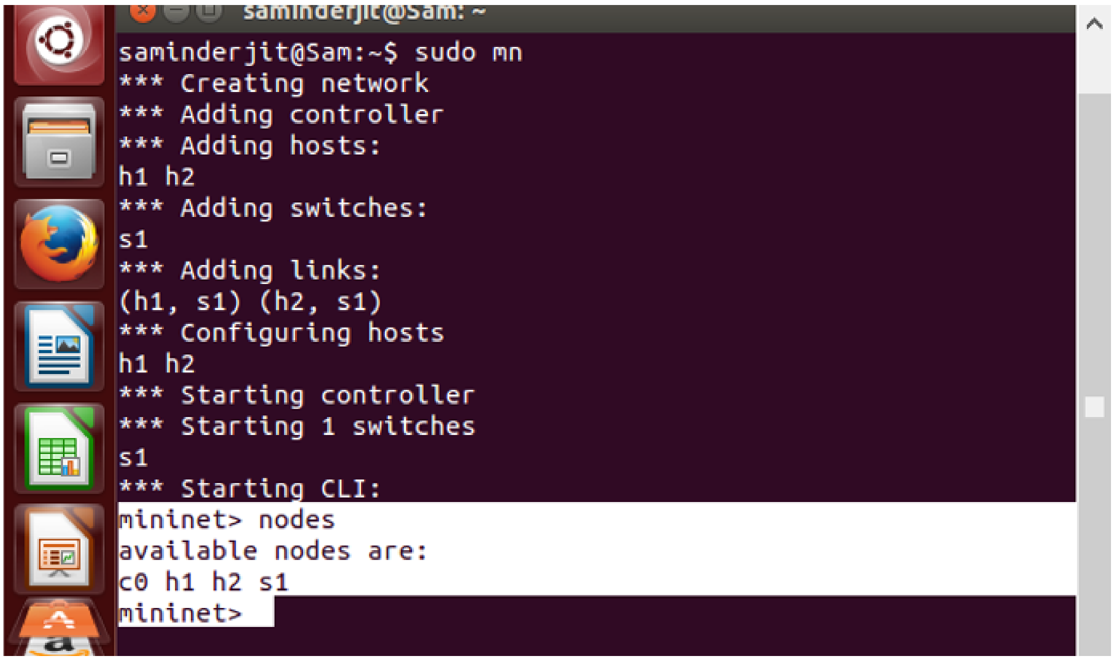
We can also install the mininet from the mininet pre installed virtual machine image available at https://github.com/mininet/mininet/wiki/Mininet-VM-Images

- Simply download the vm image from the above link and import in to your own VM. This VM has mininet, git and vim pre-installed with it. Then install POX same as mentioned above by running git clone https://github.com/noxrepo/pox.git

IMPLEMENTATION

Till now we have successfully installed Ubuntu, Mininet, Pox and Vim Editor required for our project. Now it's time to implement them.

Step 1: Type "*sudo mn*" from your home repository to check mininet is working. It will create a default topology with single switch and two hosts as shown below:

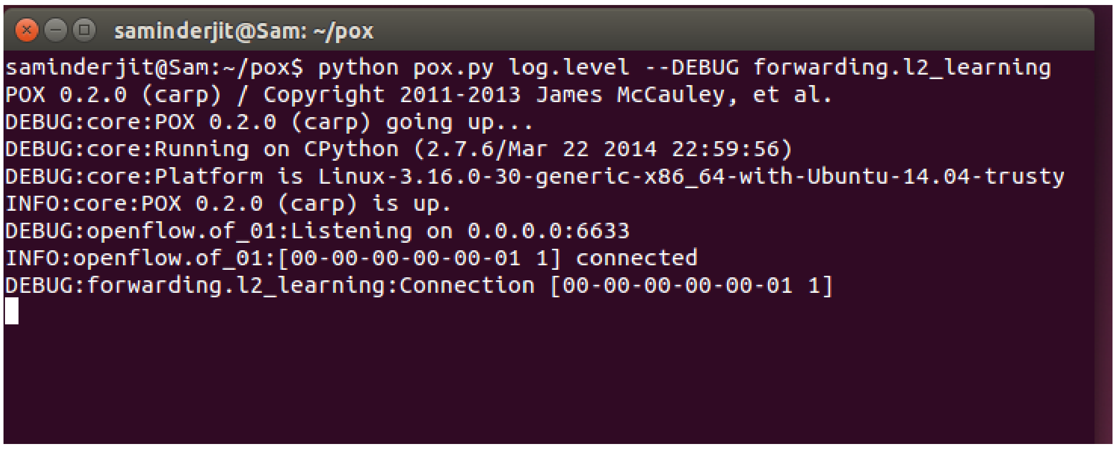


```
saminderjit@Sam:~$ sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet> nodes
available nodes are:
c0 h1 h2 s1
mininet>
```

Figure 6 Running basic topology in Mininet

If “*sudo mn*” is working and creating a default topo as shown in screenshot it means mininet is working fine. We can customize the topology according to our requirement which we will see later.

Step 2: Exit the mininet by typing “*exit*” and again run it with “*sudo mn –controller=remote”***.** Open another terminal and go inside the pox repository. Type “*python pox.py log.level –DEBUG forwarding.l2_leaning”*. Default port for the openflow traffic is 6633 on which mininet is already running. So, it should show the datapath name and id of switch running on mininet connected.

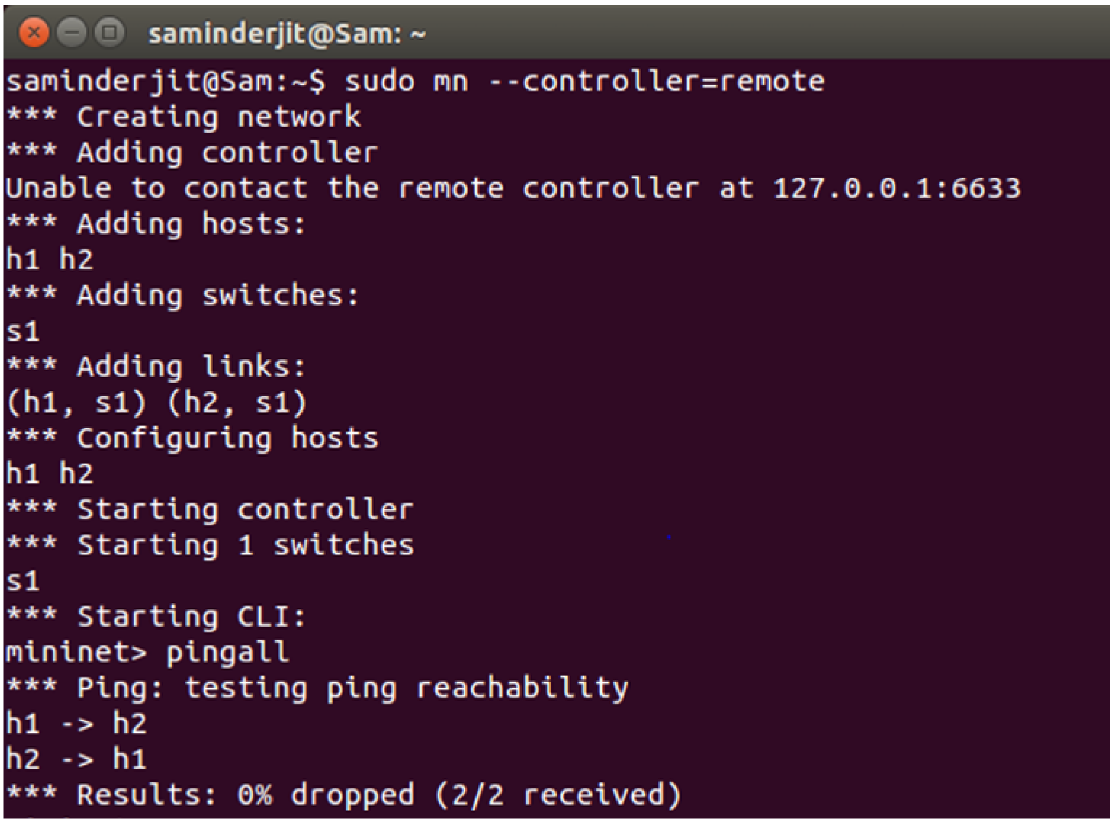


```
saminderjit@Sam: ~/pox
saminderjit@Sam:~/pox$ python pox.py log.level --DEBUG forwarding.l2_learning
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:56)
DEBUG:core:Platform is Linux-3.16.0-30-generic-x86_64-with-Ubuntu-14.04-trusty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-01 1]
```

Figure 7 Checking POX controller

Type “*pingall”* to ping every host with each other on mininet , then it should show 0% dropped packet on mininet console as follows:

Mininet Console:

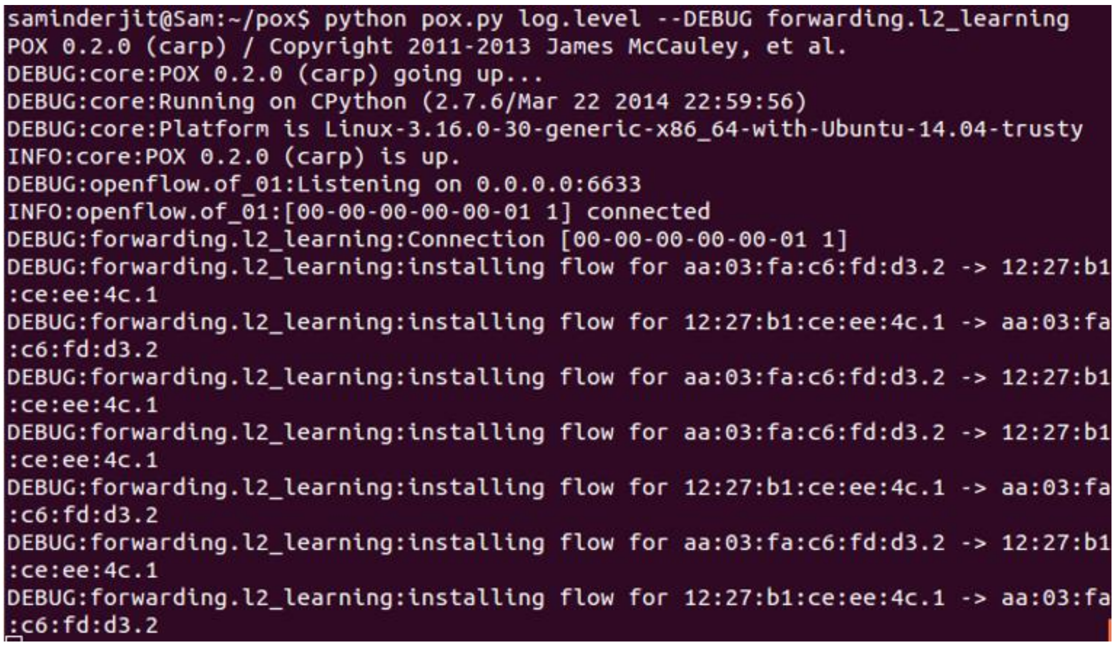


```
saminderjit@Sam: ~
saminderjit@Sam:~$ sudo mn --controller=remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
```

Figure 8 Verifying connectivity of hosts for Mininet basic topology

On pox console it will show the mac addresses of the hosts pinging with each other as follows:

POX Console:



```
saminderjit@Sam:~/pox$ python pox.py log.level --DEBUG forwarding.l2_learning
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:56)
DEBUG:core:Platform is Linux-3.16.0-30-generic-x86_64-with-Ubuntu-14.04-trusty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-01 1]
DEBUG:forwarding.l2_learning:installing flow for aa:03:fa:c6:fd:d3.2 -> 12:27:b1
:ce:ee:4c.1
DEBUG:forwarding.l2_learning:installing flow for 12:27:b1:ce:ee:4c.1 -> aa:03:fa
:c6:fd:d3.2
DEBUG:forwarding.l2_learning:installing flow for aa:03:fa:c6:fd:d3.2 -> 12:27:b1
:ce:ee:4c.1
DEBUG:forwarding.l2_learning:installing flow for aa:03:fa:c6:fd:d3.2 -> 12:27:b1
:ce:ee:4c.1
DEBUG:forwarding.l2_learning:installing flow for 12:27:b1:ce:ee:4c.1 -> aa:03:fa
:c6:fd:d3.2
DEBUG:forwarding.l2_learning:installing flow for aa:03:fa:c6:fd:d3.2 -> 12:27:b1
:ce:ee:4c.1
DEBUG:forwarding.l2_learning:installing flow for 12:27:b1:ce:ee:4c.1 -> aa:03:fa
:c6:fd:d3.2
```

Figure 9 Checking the POX console

So, this is how a basic topology on mininet can be run and how a remote controller can control the traffic based on the type of application we are running on the controller.

USE CASE: DEPLOY TENANT FIREWALL USING SDN CONTROLLER (POX) USING PYTHON API

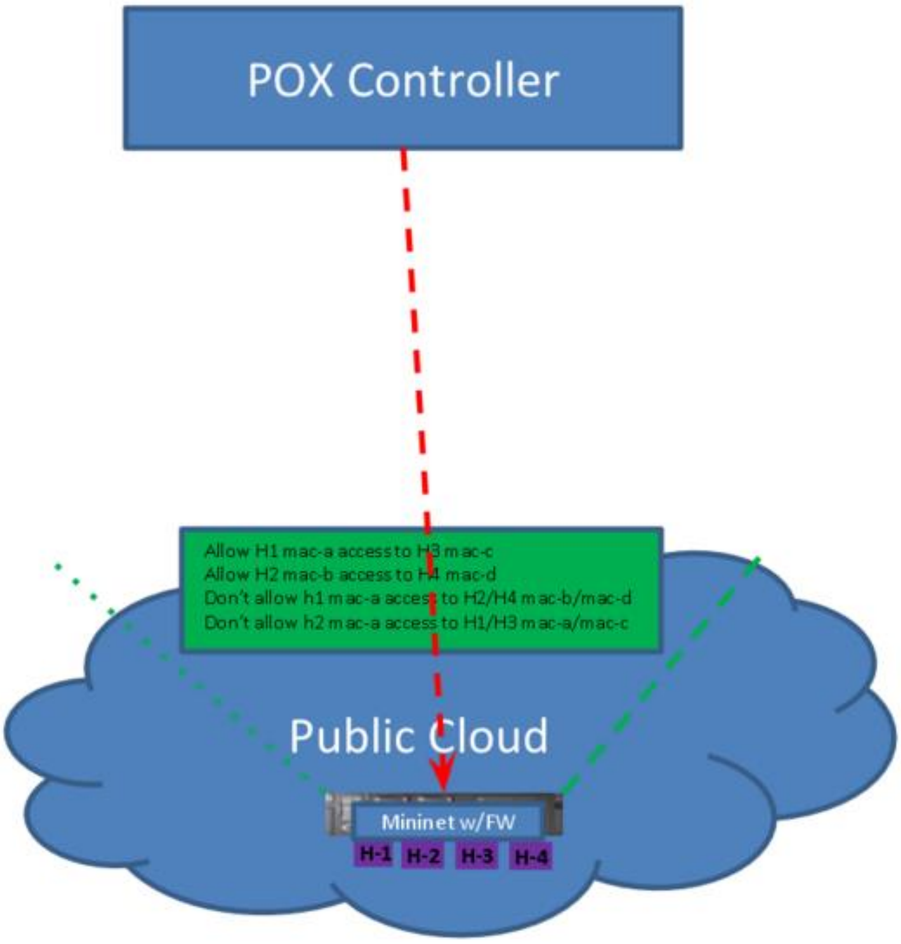

POX Controller
Allow H1 mac-a access to H3 mac-c
Allow H2 mac-b access to H4 mac-d
Don't allow h1 mac-a access to H2/H4 mac-b/mac-d
Don't allow h2 mac-a access to H1/H3 mac-a/mac-c
Public Cloud
Mininet w/FW
H-1
H-2
H-3
H-4

Figure 10 Our Project Scenario

SDN - DC AS A FIREWALL SERVICE

SDN created the whole new world of network design and enabled the innovative approaches to networking due to the separation of control plane and the data plane. Thus, SDN also caused us to reconsider how the security policies are enforced in the network.

Open vSwitch has traditionally supported only stateless matches on the policies. However, work is being done by the Open vSwitch community to have connection tracking to maintain the state tables of existing sessions. If any organisation requires the stateful firewall protections specifically, then they must use the SDN policies to steer the traffic with service-chaining toward a stateful packet inspection Network Function Virtualization (NFV) firewall.

I deployed the firewall as a service for hierarchical multi-tenant Data Center network by provisioning rules to allow specific source MAC addresses to communicate specific Tenants while resolving the conflict between the standard programmed flow and Firewall programmed flows.

My research work can be divided into following key points:

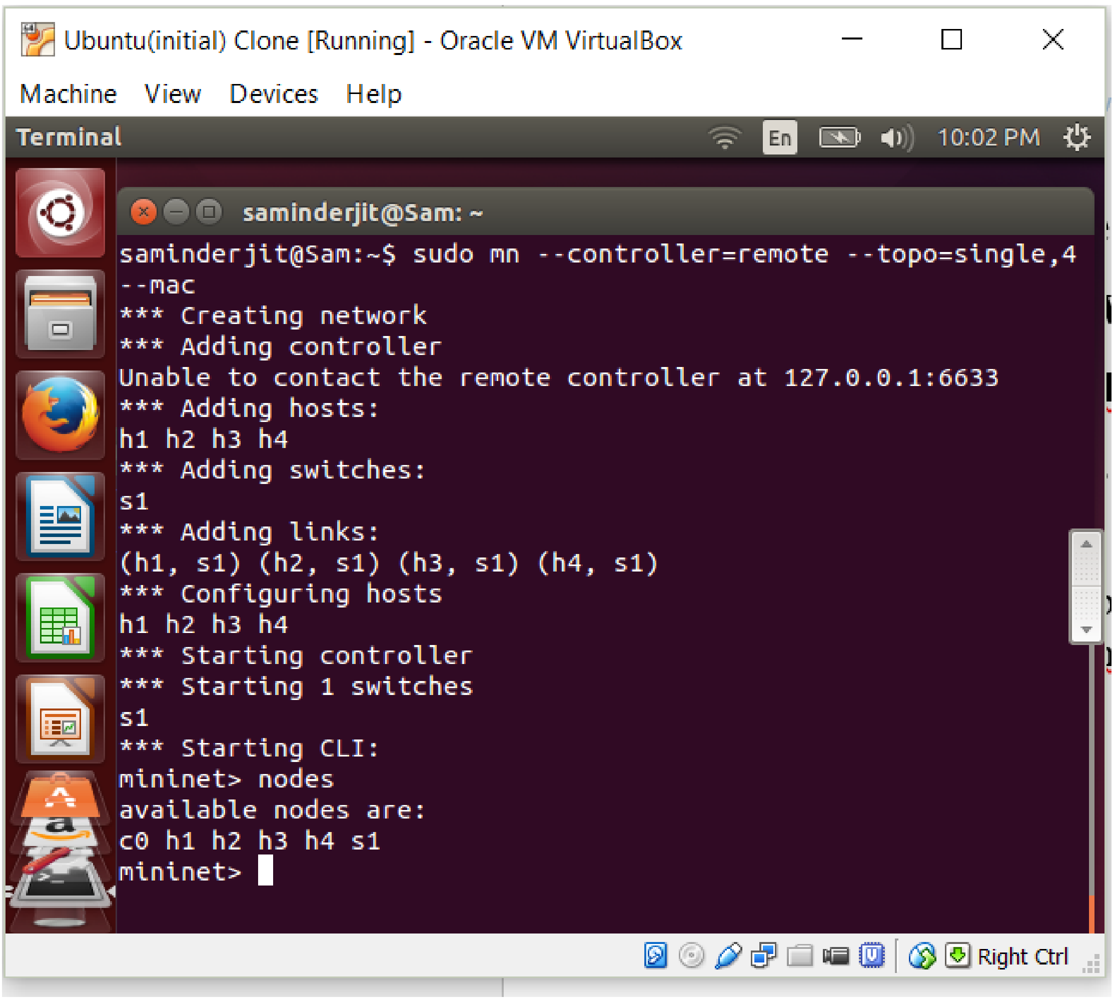
- Deploy Hierarchical multi-Tenant Data Center network in the automated manner using Mininet Python API infrastructure simulation.
- Then deploy Python based SDN Controller (POX) in layer 2 switch mode and verify end to end connectivity of the hosts.
- Finally deploy the firewall plugin in the layer 2 switch controller which installs the firewall flow rules by reading the csv file containing the rules taking care to resolve the conflicts between standard programmed flow and firewall programed flows.

DEPLOY HIERARCHICAL MULTI-TENANT DATA CENTER NETWORK IN THE AUTOMATED MANNER USING MININET PYTHON API INFRASTRUCTURE SIMULATION.

According to our project requirement, we need to deploy single switch with four hosts in mininet environment.

Type “*sudo mn –controller=remote –topo=single, 4 –mac”*

It will create a single switch with four hosts as shown below:

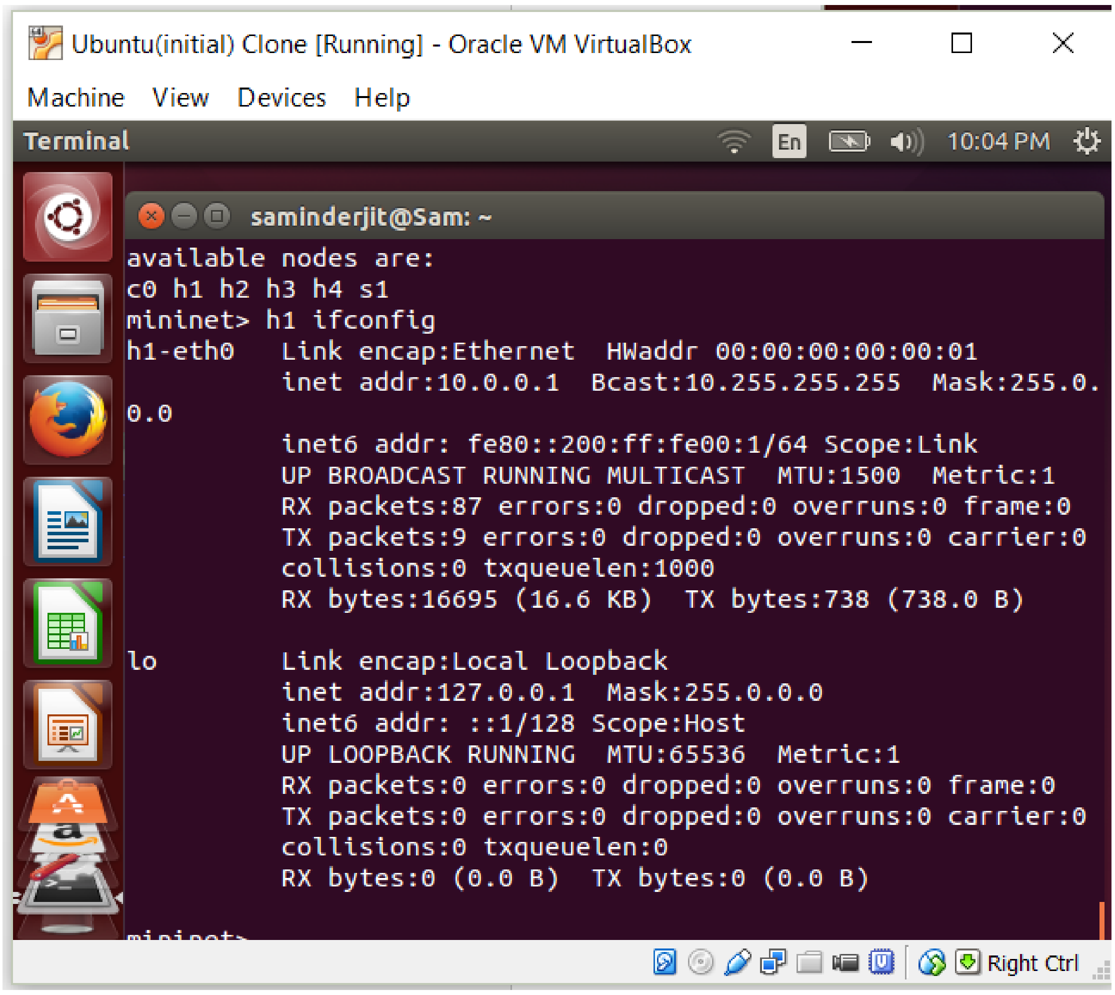

Ubuntu(initial) Clone [Running] - Oracle VM VirtualBox
Machine View Devices Help
Terminal
En
10:02 PM
saminderjit@Sam: ~
Right Ctrl

```
saminderjit@Sam:~$ sudo mn --controller=remote --topo=single,4
--mac
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet> nodes
available nodes are:
c0 h1 h2 h3 h4 s1
mininet>
```

Figure 11 Running Mininet for our required topology

We can also see the ip address and mac address of our hosts by typing

"hostname ifconfig"


Ubuntu(initial) Clone [Running] - Oracle VM VirtualBox
Machine View Devices Help
Terminal
10:04 PM
saminderjit@Sam: ~
Right Ctrl

```
available nodes are:
c0 h1 h2 h3 h4 s1
mininet> h1 ifconfig
h1-eth0   Link encap:Ethernet  HWaddr 00:00:00:00:00:01
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.
0.0
          inet6 addr: fe80::200:ff:fe00:1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:87 errors:0 dropped:0 overruns:0 frame:0
          TX packets:9 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:16695 (16.6 KB)  TX bytes:738 (738.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet>
```

Figure 12 Checking Hosts Configurations

In out topology our host have following configuration:

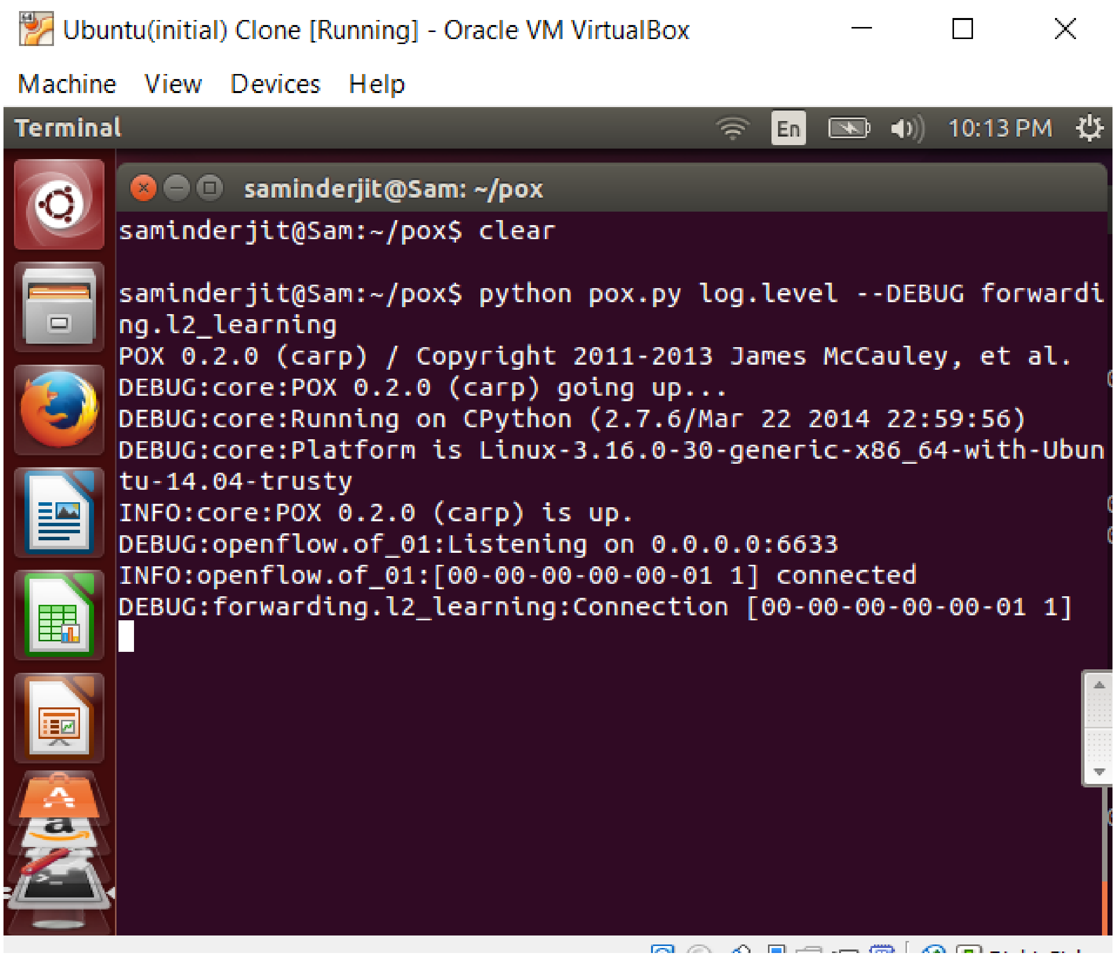
hostname	*ip*	*mac*
h1	*10.0.0.1*	*00:00:00:00:00:01*
h2	*10.0.0.2*	*00:00:00:00:00:02*
h3	*10.0.0.3*	*00:00:00:00:00:03*
h4	*10.0.0.4*	*00:00:00:00:00:04*

DEPLOY PYTHON BASED SDN CONTROLLER (POX) IN LAYER 2 SWITCH MODE AND VERIFY END TO END CONNECTIVITY OF THE HOSTS.

Now with mininet running in one terminal , open the other terminal and go inside the pox repository and type:

"python pox.py log.level –DEBUG forwarding.l2_learning"

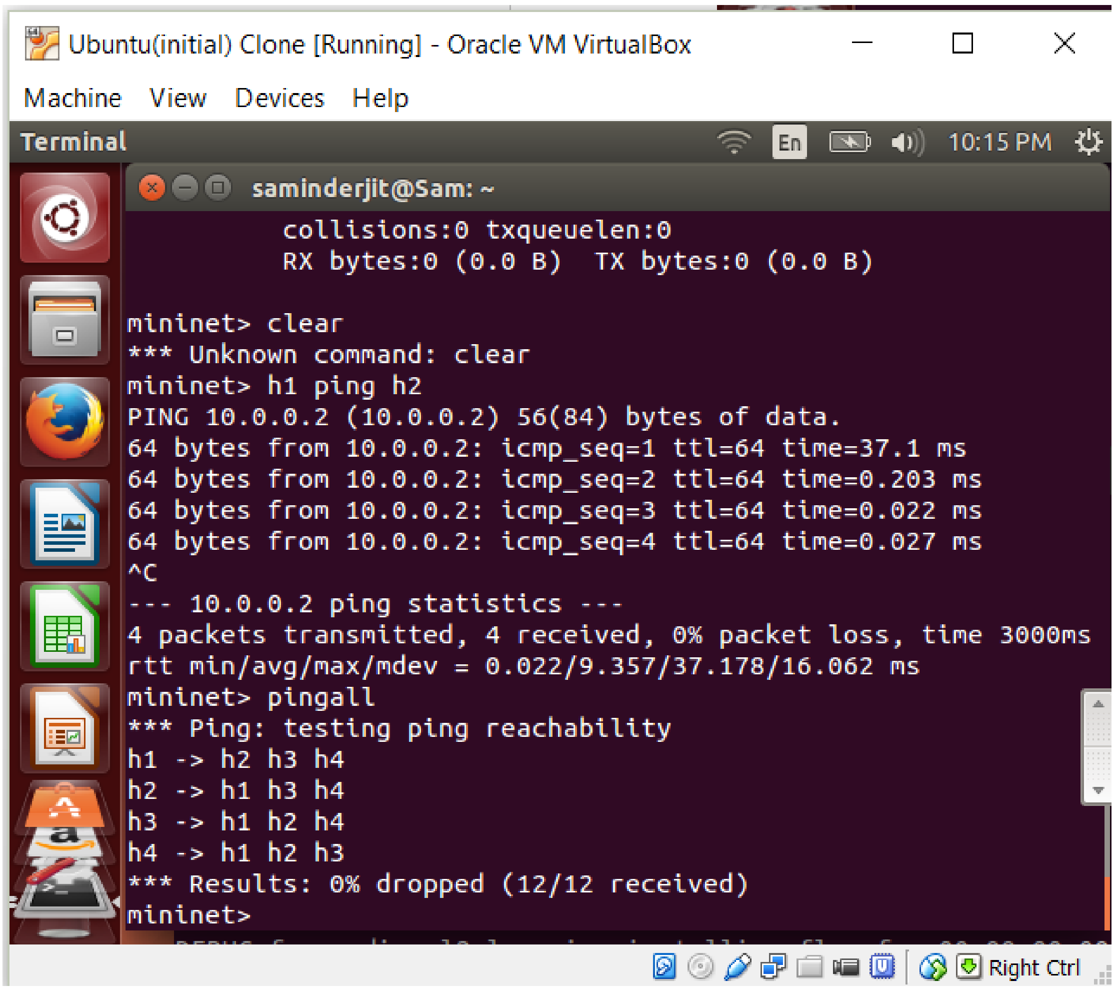
It will show the datapath name and dpid (of our running topology in mininet) connected as follows:


Ubuntu(initial) Clone [Running] - Oracle VM VirtualBox
Machine View Devices Help
Terminal
En
10:13 PM
saminderjit@Sam: ~/pox

```
saminderjit@Sam:~/pox$ clear

saminderjit@Sam:~/pox$ python pox.py log.level --DEBUG forwardi
ng.l2_learning
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:56)
DEBUG:core:Platform is Linux-3.16.0-30-generic-x86_64-with-Ubun
tu-14.04-trusty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-01 1]
```

Figure 13 Checking POX connectivity for our required topology

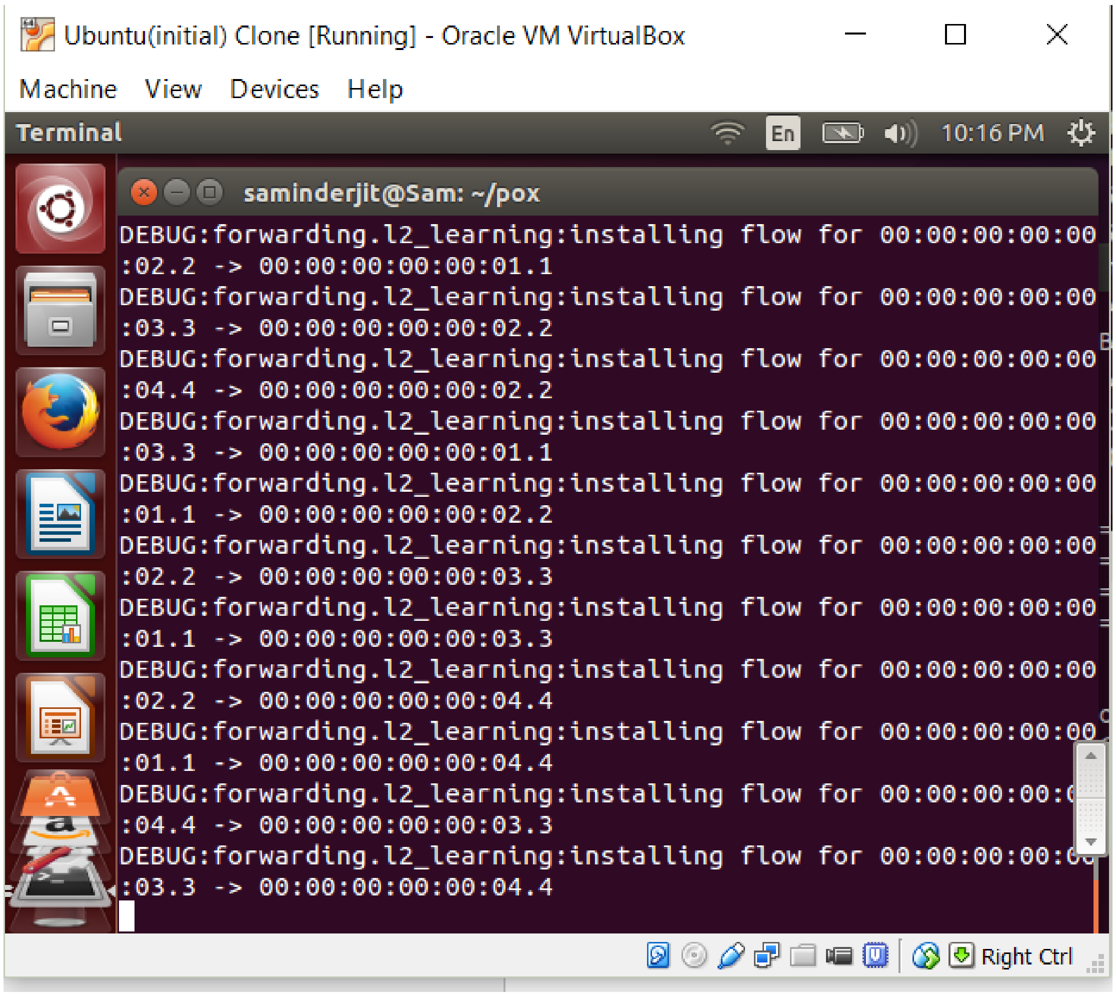
Now we can verify end to end connectivity of our hosts by typing "*pingall or <hostname1> ping <hostname2>*" as shown in screenshots below and it should work:


Ubuntu(initial) Clone [Running] - Oracle VM VirtualBox
Machine View Devices Help
Terminal
10:15 PM
saminderjit@Sam: ~
Right Ctrl

```
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet> clear
*** Unknown command: clear
mininet> h1 ping 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=37.1 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.203 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.022 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.027 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.022/9.357/37.178/16.062 ms
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet>
```

Figure 14 Verifying end to end connectivity for required topology

Our pox controller shows the mac addresses of the flows it installs on the switch as below:


Ubuntu(initial) Clone [Running] - Oracle VM VirtualBox
Machine View Devices Help
Terminal
En
10:16 PM
saminderjit@Sam: ~/pox
Right Ctrl

```
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00
:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00
:03.3 -> 00:00:00:00:00:02.2
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00
:04.4 -> 00:00:00:00:00:02.2
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00
:03.3 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00
:01.1 -> 00:00:00:00:00:02.2
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00
:02.2 -> 00:00:00:00:00:03.3
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00
:01.1 -> 00:00:00:00:00:03.3
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00
:02.2 -> 00:00:00:00:00:04.4
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00
:01.1 -> 00:00:00:00:00:04.4
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:0
:04.4 -> 00:00:00:00:00:03.3
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:0
:03.3 -> 00:00:00:00:00:04.4
```

Figure 15 Checking flows on POX console

DEPLOY THE FIREWALL PLUGIN IN THE LAYER 2 SWITCH CONTROLLER.

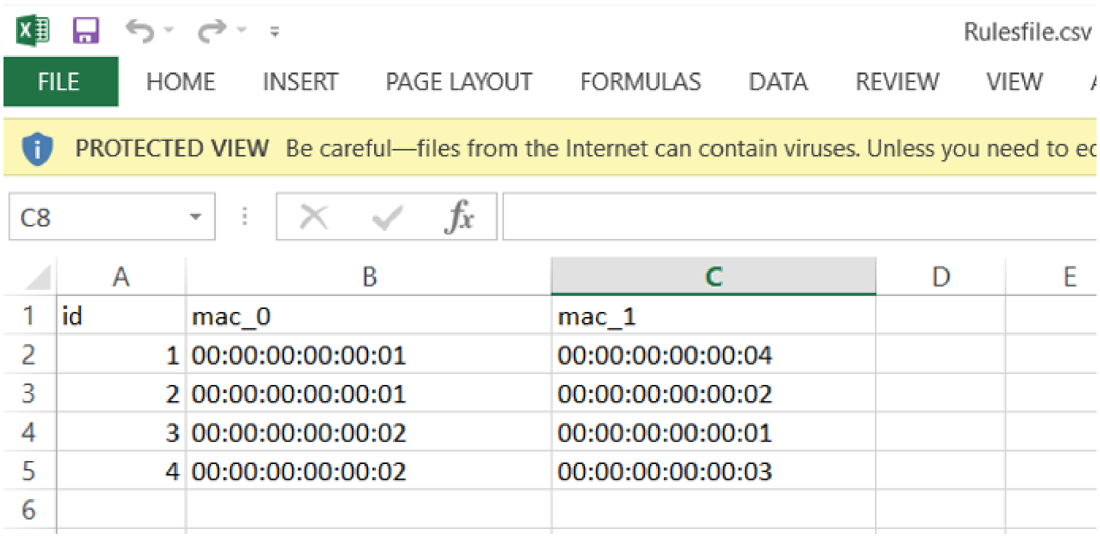
So far our basic layer 2 switching operations are working fine, but our main task is to orchestrate the firewall plugin in our main controller to install the firewall rules on our switch to allow/disallow the flow accordingly.

I wrote the firewall rules in a csv file with mac addresses which should not communicate with each other and my controller will install all these rules at the starting when our controller connects to the switch.

According to our project:

- H1 should not communicate with H2/H4
- H2 should not communicate with H1/H3
- H1 should communicate with H3
- H2 should communicate with H4
- H3 should communicate with H4

So, I wrote the mac addresses of H1 and H4; H1 and H4; H2 and H1; H2 and H3 in a csv file from where our firewall will read the rules and install these flows to disallow communication between these hosts.

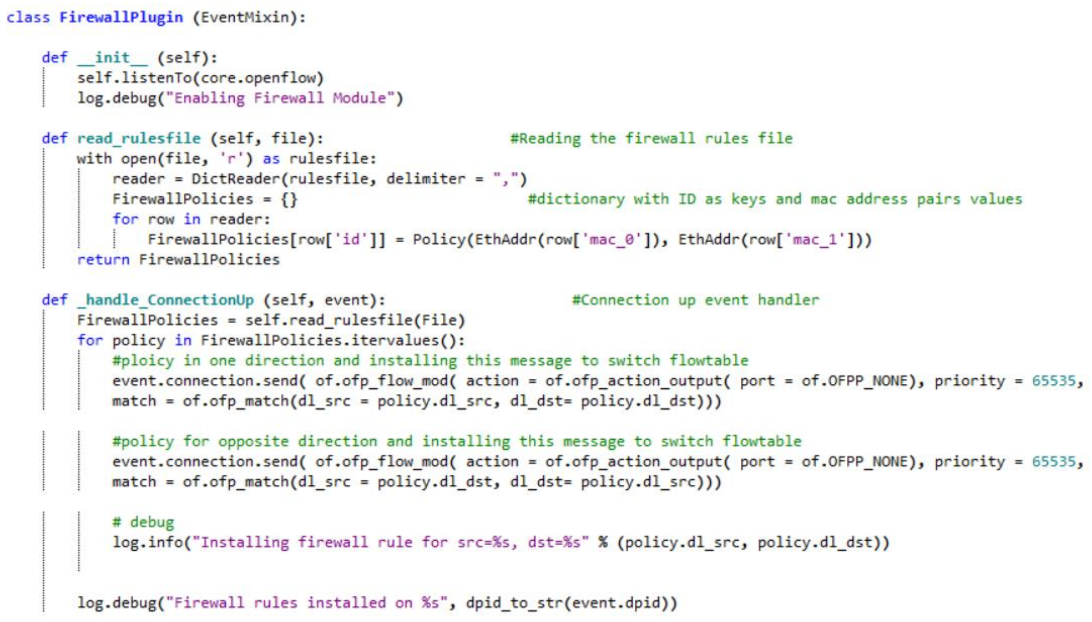


PROTECTED VIEW Be careful—files from the Internet can contain viruses. Unless you need to ed

	A	B	C	D	E
1	id	mac_0	mac_1		
2	1	00:00:00:00:00:01	00:00:00:00:00:04		
3	2	00:00:00:00:00:01	00:00:00:00:00:02		
4	3	00:00:00:00:00:02	00:00:00:00:00:01		
5	4	00:00:00:00:00:02	00:00:00:00:00:03		
6					

Figure 16 CSV file format

So, now I changed the layer 2 switch controller code by adding some lines to install the firewall plugin which reads the rules from csv file above and install the rules on switch to disallow communication between these tenants.



```
class FirewallPlugin (EventMixin):

    def __init__ (self):
        self.listenTo(core.openflow)
        log.debug("Enabling Firewall Module")

    def read_rulesfile (self, file):                    #Reading the firewall rules file
        with open(file, 'r') as rulesfile:
            reader = DictReader(rulesfile, delimiter = ",")
            FirewallPolicies = {}                          #dictionary with ID as keys and mac address pairs values
            for row in reader:
                FirewallPolicies[row['id']] = Policy(EthAddr(row['mac_0']), EthAddr(row['mac_1']))
        return FirewallPolicies

    def _handle_ConnectionUp (self, event):                  #Connection up event handler
        FirewallPolicies = self.read_rulesfile(File)
        for policy in FirewallPolicies.itervalues():
            #ploicy in one direction and installing this message to switch flowtable
            event.connection.send( of.ofp_flow_mod( action = of.ofp_action_output( port = of.OFPP_NONE), priority = 65535,
            match = of.ofp_match(dl_src = policy.dl_src, dl_dst= policy.dl_dst)))

            #policy for opposite direction and installing this message to switch flowtable
            event.connection.send( of.ofp_flow_mod( action = of.ofp_action_output( port = of.OFPP_NONE), priority = 65535,
            match = of.ofp_match(dl_src = policy.dl_dst, dl_dst= policy.dl_src)))

            # debug
            log.info("Installing firewall rule for src=%s, dst=%s" % (policy.dl_src, policy.dl_dst))


        log.debug("Firewall rules installed on %s", dpid_to_str(event.dpid))
```

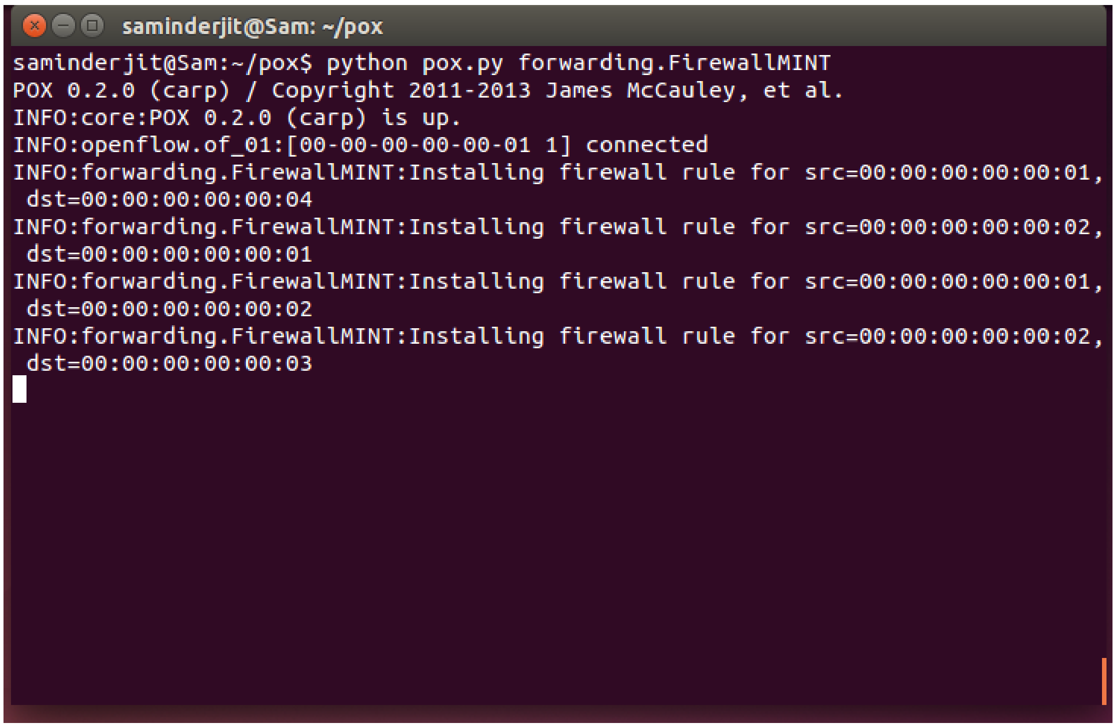
Figure 17 Firewall Plugin code screenshot

Now, copy and paste the FirewallMint.py file inside the pox controller repository at /pox/pox/forwarding location I submitted with this report.

Then, copy the rules file inside the pox repository from where we run our pox controller.

Finally our job is done and we can verify the results by running FirewallMint.py file as a controller with the topology we are using in mininet.

Our Firewall installs the default rules on the switch as follows when we run the controller file :

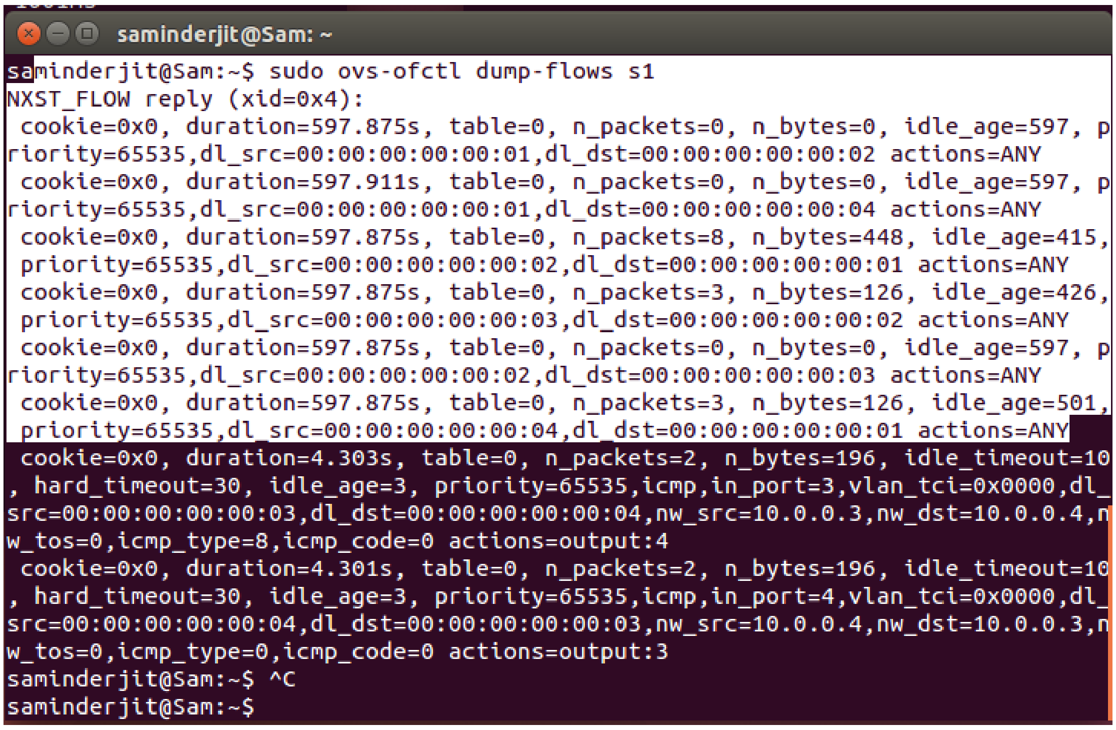

saminderjit@Sam: ~/pox

```
saminderjit@Sam:~/pox$ python pox.py forwarding.FirewallMINT
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
INFO:forwarding.FirewallMINT:Installing firewall rule for src=00:00:00:00:00:01,
 dst=00:00:00:00:00:04
INFO:forwarding.FirewallMINT:Installing firewall rule for src=00:00:00:00:00:02,
 dst=00:00:00:00:00:01
INFO:forwarding.FirewallMINT:Installing firewall rule for src=00:00:00:00:00:01,
 dst=00:00:00:00:00:02
INFO:forwarding.FirewallMINT:Installing firewall rule for src=00:00:00:00:00:02,
 dst=00:00:00:00:00:03
```

Figure 18 Installing Firewall flow rules on switch

Switch flows on Switch S1 as below:

(white part is firewall flow and other part is normal flow with output port action)

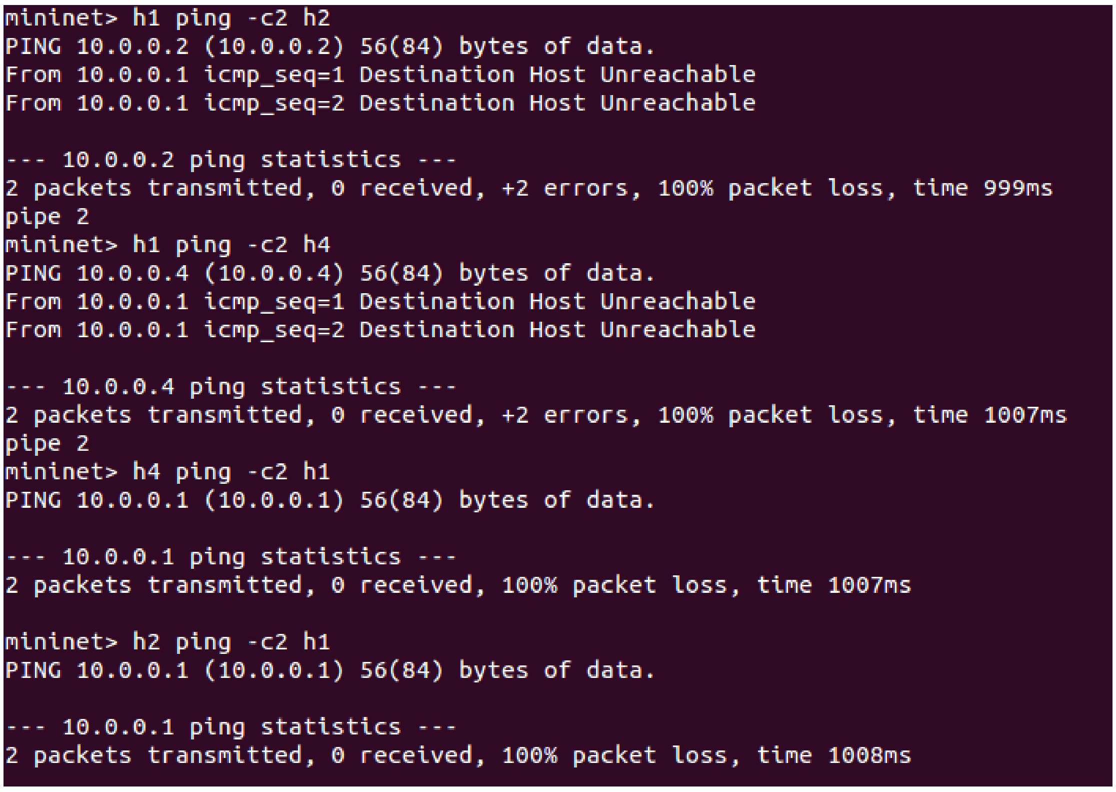

saminderjit@Sam: ~

```
saminderjit@Sam:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=597.875s, table=0, n_packets=0, n_bytes=0, idle_age=597, p
riority=65535,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02 actions=ANY
 cookie=0x0, duration=597.911s, table=0, n_packets=0, n_bytes=0, idle_age=597, p
riority=65535,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:04 actions=ANY
 cookie=0x0, duration=597.875s, table=0, n_packets=8, n_bytes=448, idle_age=415,
 priority=65535,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01 actions=ANY
 cookie=0x0, duration=597.875s, table=0, n_packets=3, n_bytes=126, idle_age=426,
 priority=65535,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:02 actions=ANY
 cookie=0x0, duration=597.875s, table=0, n_packets=0, n_bytes=0, idle_age=597, p
riority=65535,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:03 actions=ANY
 cookie=0x0, duration=597.875s, table=0, n_packets=3, n_bytes=126, idle_age=501,
 priority=65535,dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01 actions=ANY
 cookie=0x0, duration=4.303s, table=0, n_packets=2, n_bytes=196, idle_timeout=10
, hard_timeout=30, idle_age=3, priority=65535,icmp,in_port=3,vlan_tci=0x0000,dl_
src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:04,nw_src=10.0.0.3,nw_dst=10.0.0.4,n
w_tos=0,icmp_type=8,icmp_code=0 actions=output:4
 cookie=0x0, duration=4.301s, table=0, n_packets=2, n_bytes=196, idle_timeout=10
, hard_timeout=30, idle_age=3, priority=65535,icmp,in_port=4,vlan_tci=0x0000,dl_
src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:03,nw_src=10.0.0.4,nw_dst=10.0.0.3,n
w_tos=0,icmp_type=0,icmp_code=0 actions=output:3
saminderjit@Sam:~$ ^C
saminderjit@Sam:~$
```

Figure 19 Checking the Switch Flow Table entries

TESTCASES

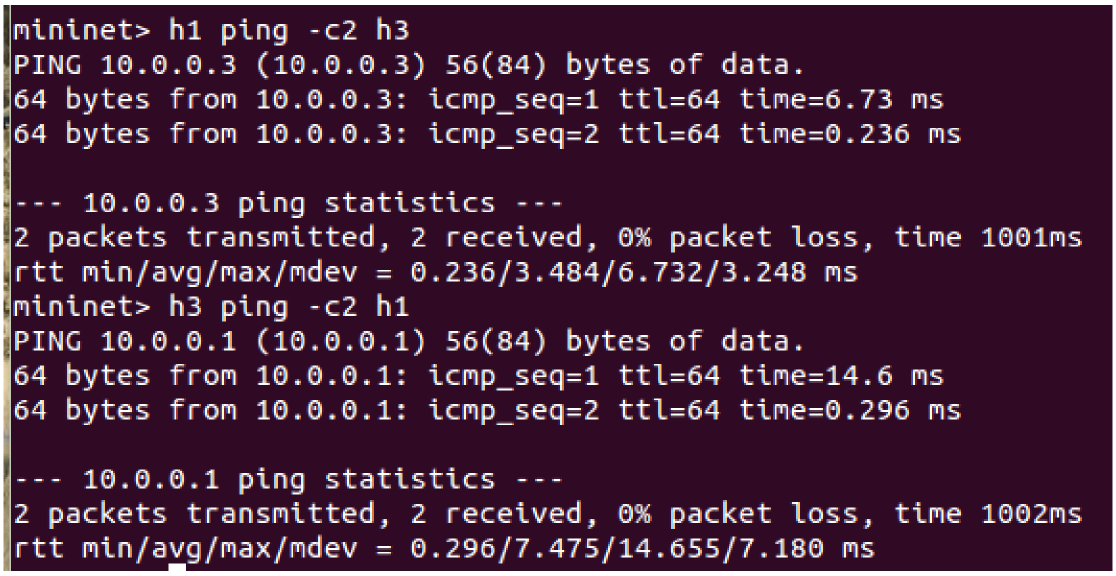
- H1 ping should not work with H2/H4



```
mininet> h1 ping -c2 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 999ms
pipe 2
mininet> h1 ping -c2 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable

--- 10.0.0.4 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1007ms
pipe 2
mininet> h4 ping -c2 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.

--- 10.0.0.1 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1007ms

mininet> h2 ping -c2 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.

--- 10.0.0.1 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1008ms
```

Figure 20 Verifying Firewall Rule Implementation between H1 and H2/H4

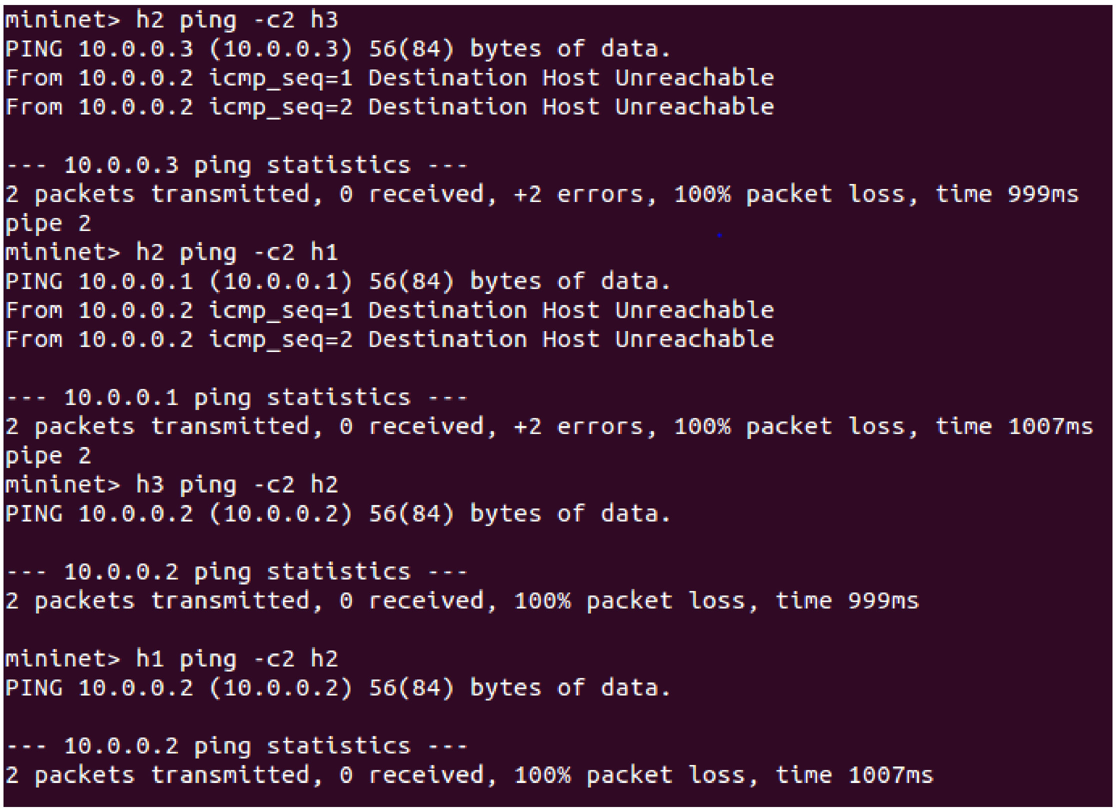
- H1 ping should work with H3



```
mininet> h1 ping -c2 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=6.73 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.236 ms

--- 10.0.0.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.236/3.484/6.732/3.248 ms
mininet> h3 ping -c2 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=14.6 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.296 ms

--- 10.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.296/7.475/14.655/7.180 ms
```

Figure 21 Verifying end to end connectivity between H1 and H3

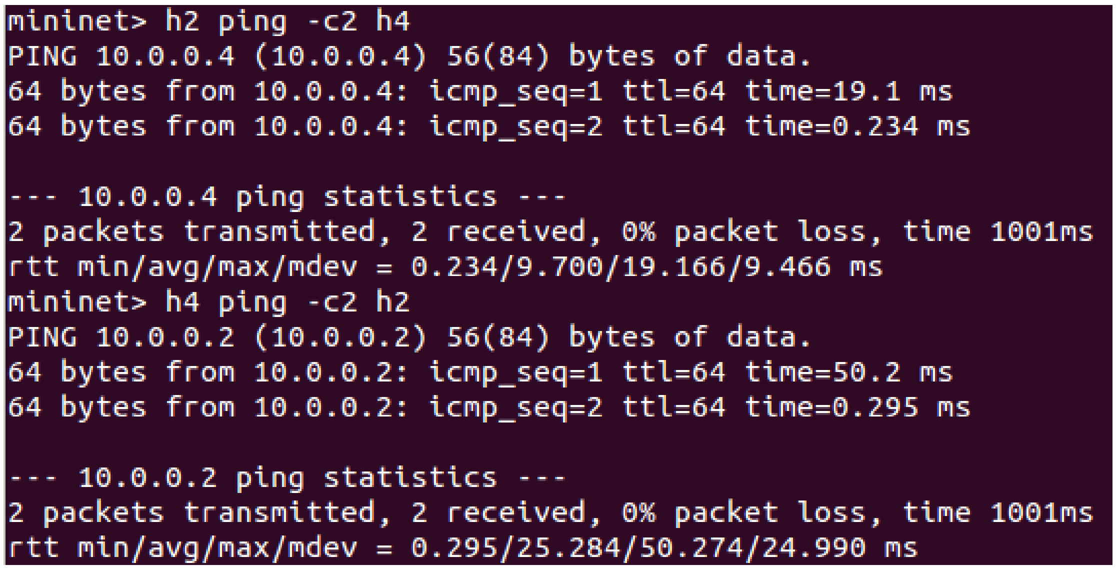
- H2 ping should not work with H1/H3



```
mininet> h2 ping -c2 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
From 10.0.0.2 icmp_seq=1 Destination Host Unreachable
From 10.0.0.2 icmp_seq=2 Destination Host Unreachable

--- 10.0.0.3 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 999ms
pipe 2
mininet> h2 ping -c2 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
From 10.0.0.2 icmp_seq=1 Destination Host Unreachable
From 10.0.0.2 icmp_seq=2 Destination Host Unreachable

--- 10.0.0.1 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1007ms
pipe 2
mininet> h3 ping -c2 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 999ms

mininet> h1 ping -c2 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1007ms
```

Figure 22 Verifying Firewall Rule Implementation between H2 and H1/H3

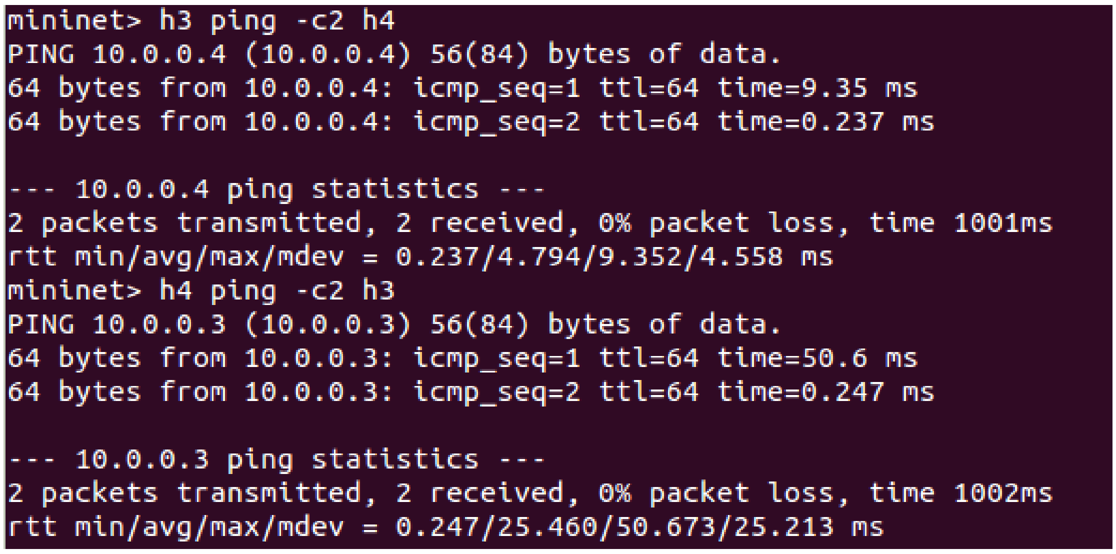
- H2 ping should work with H4



```
mininet> h2 ping -c2 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=19.1 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.234 ms

--- 10.0.0.4 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.234/9.700/19.166/9.466 ms
mininet> h4 ping -c2 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=50.2 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.295 ms

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.295/25.284/50.274/24.990 ms
```

Figure 23 Verifying end to end connectivity between H2 and H4

- H3 ping should work with H4



```
mininet> h3 ping -c2 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=9.35 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.237 ms

--- 10.0.0.4 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.237/4.794/9.352/4.558 ms
mininet> h4 ping -c2 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=50.6 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.247 ms

--- 10.0.0.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.247/25.460/50.673/25.213 ms
```

Figure 24 Verifying end to end connectivity between H3 and H4

SUMMARY AND CONCLUSION

We successfully deployed the firewall plugin to enable the firewall rules in multi-tenant data center automated environment. This can be used to deploy the firewall policies for thousands of tenants simultaneously in real data center environment with centralised control and visibility.

FUTURE WORK TO DO:

- Support for real world leading network virtualization and SDN platforms including VMware NSX, Cisco Application ACI and OpenStack SDN.
- Extensible management interface – API's for the cloud automation and orchestration.

BIBLIOGRAPHY & REFERENCES:

https://en.wikipedia.org/wiki/Software-defined_networking

https://www.sdxcentral.com/resources/sdn/what-the-definition-of-software-defined-networking-sdn/

http://www.slideshare.net/SDNCentral/sdnu-101-final

http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_16-1/161_sdn.html

http://ofsdn.blogspot.ca/

http://www.networkworld.com/article/2905257/sdn/is-an-sdn-switch-a-new-form-of-a-firewall.html

http://mininet.org/overview/

http://www.brianlinkletter.com/using-the-pox-sdn-controller/

http://searchsdn.techtarget.com/definition/POX

http://www.fortinet.com/solutions/data-center-firewalls.html

http://www.fortinet.com/solutions/sdn.html

https://git-scm.com/

http://www.ubuntu.com/download/desktop/install-ubuntu-desktop

https://github.com/mininet/mininet

https://github.com/mininet/mininet/wiki/Mininet-VM-Images

http://www.ubuntu.com/download/desktop

http://image.slidesharecdn.com/sdnpresentation-bbf-final-131111113336-phpapp02/95/sdn-openflow-17-638.jpg?cb=1384169835