School of Information Sciences

Term Project Report

Network Orchestration using OpenFlow

Registration Number	Name	Branch	E-mail
171041009	Preethi S R	Computing technologies and Virtualization	preethi.rudradevaramatt@gmail.com
171041005	Anish N Shetty	Computing technologies and Virtualization	anishshetty@live.com

Guide/Panel Team

Name
Mr. Mohan Kumar J
Mr. Sathyanarayan Shenoy
Mr. Samarendra Bhat

Submitted on

06-Nov-2017

Table of Contents

List of Figures

Acronyms Used

SDN	Software Defined Networking
CLI	Command Line Interface
GUI	Graphical User Interface
NOS	Network Operation System
VM	Virtual Machine
NIC	Network Interface Card
VNC	Virtual Network Computer
SSH	Secure Shell
API	Application Programming Interface
ACL	Access Control List
TCP	Transmission Control Protocol
TLS	Transport Layer Security
NAT	Network Address Translation
AWS	Amazon Web Service
VDI	Virtual Desktop Infrastructure
EC2	Elastic Compute Cloud

Abstract

Keywords: Software-defined networking (SDN); Mininet; Openflow protocol; POX Controller; Amazon EC2; VirtualBox; Firewall.

A new approach to computer networking is given by the concept of Software Defined Networking, which disintegrates the functionality of the underlying layers of the existing networking model for making network administration easier. SDN is solely based on the disintegration of the system which decides where the data is to be sent, from the system that actually forwards the data. SDN has showed its importance in various fields like cloud services, security enhancements, resource utilization, network management, bandwidth calendaring.

In case of traditional networks, the networking devices need to deployed and configured individually through Command Line Interface (CLI) or script. This requireS trained professionals to be employed in order to maintain the networking infrastructure. Moreover, many of the protocols used in traditional networking devices are proprietary to particular companies. Hence, professionals specific to the devices used have to be employed.

Software defined networking removing all these bottlenecks, if developed correctly, the networking infrastructure can entirely be managed by GUI based applications. This way it ensures reduced operational expenditures (Opex), since trained professionals need not be employed for simple configurations.

It also reduces the capital expenditures (Capex) by using what are called “Openflow Switches”. These switches do not have complex processing or decision making capabilities; they rely on a controller to perform these decision makings. Thereby removing unnecessary processing power from several switches and moving it into a single device i.e. the controller.

Project Objective & Motivation

The main objective of the project is to build a virtual network infrastructure consisting of many switches and hosts and manage them. Each of the switches are connected to a centralized controller. This way a communication platform could be set up among all the connected devices. These devices are then connected to the internet. And a GUI based application is developed and used to perform simple network management such as firewalling. Thereby showcasing the ease in which network-related operations can be performed using Software Defined Networking (SDN).

Chapter – 1

1.1 Introduction

There is a swift development of Software-Defined Networking (SDN) worldwide and is gaining attention of lot of networking industries who show firm support for the open-flow protocol. A lot of development has been done contributing to the southbound and northbound interfaces of the SDN architecture. Traditional networks face the disadvantage of not being able to configure the network according to predefined policies, and to reconfigure it to respond to faults, load, and changes. SDN promises to change this state of affairs, by breaking vertical integration, separating the networks control logic from the underlying routers and switches, promoting (logical) centralization of network control, and introducing the ability to program the network. The separation of concerns, introduced between the definition of network policies, their implementation in switching hardware, and the forwarding of traffic, is key to the desired exibility by breaking the network control problem into tractable pieces, SDN makes it easier to create and introduce new abstractions in networking, simplifying network management and facilitating network evolution.

In the principles of SDN, a network is separated into three distinct components. Communication layer, network operating system (NOS), and control program. The communication layer consists of the physical network devices like routers and switches; the NOS manages network resources; and the control program controls the network through the NOS.

SDN supports changing traffic patterns in the network and makes it more reliable. In contrast to client-server applications where the bulk of the communication occurs between one client and one server, today's applications access different databases and servers, creating a flurry of east-west machine-to-machine traffic before returning data to the end user device in the classic north-south traffic pattern. SDN also supports the rise in demand for cloud services. The need is the agility to access applications, infrastructure, and other IT resources on demand. SDN architecture may enable, facilitate or enhance network-related security applications due to the controller's central view of the network, and its capacity to reprogram the data plane at any time. Software defined networks provide a centralized view of the entire network, making it easier to centralize enterprise management and provisioning. Enterprise networks have to set up new applications and virtual machines on demand to accommodate new processing requests such as those for big data. Adopting SDN also gives new life to existing network devices. The ability to shape and control data traffic is one of the primary advantages of software defined networking.

1.2 Literature Review

The concept of SDN essentially arised in the mid-90s shortly after the release of Java in 1995. One of the initial projects of SDN was AT&T's GeoPlex. GeoPlex didn't focus on developing an operating system which would run on the networking devices. It focused more on developing what were called a "Soft Switch" that could reconfigure hardware switches and setup new

services. However, the actual operating systems running on the hardware devices acted as a barrier to this early project.

Thereafter many organizations and developers have contributed to the development of SDN. Some of the prominent ones were Sun Microsystems, WebSprocket, Ericsson, Garter groups, OARnet and others.

However rapid development of SDN could be hugely credited to the Open Networking Foundation which was founded in 2011. Since then, the foundation has held several conferences to address the capabilities of Openflow and SDN to the common masses.

Chapter – 2

Progress Plan

Week	Objectives
1	Literature Review
2	Installing Mininet, Putty and Xming
3	Getting familiar with mininet commands and configurations
4	Developed a program to implement firewall function
5	Hosted the controller on the cloud and connected to it
6	Collaborate the done work with a cloud based controller
7	Replicated the above on actual VMs and OpenVSwitch
8	Finalizing the project setup

Chapter – 3

Requirements Specification

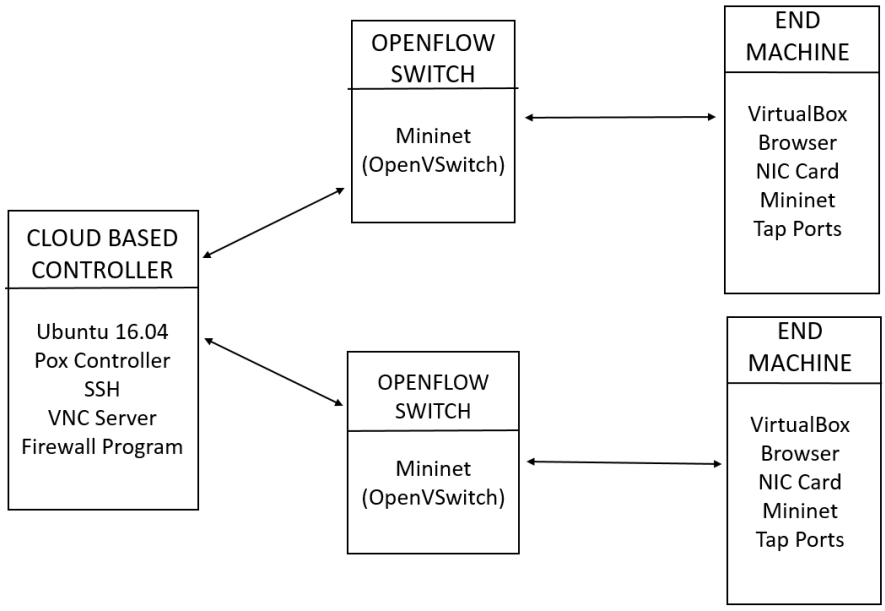

Figure 3.1: Requirements

3.1 Cloud based Controller:

This controller is required to control all the switches in the network. And these switches in turn control the hosts and virtual machines communicating with him. In our project we have set up a Ubuntu 64-bit virtual machines in the Amazon EC2 cloud platform. And we have chosen POX as our controller. In order to communicate with the switches, the controller has to connect to them via a certain port. All the configurations and setup in the controller can be done via SSH or VNC. The same can be used to access the controller while performing the firewalling function. And finally, the cloud would be running the python code which was developed as a part of this project. This code provides the required GUI, and can be used to perform the operations such as Enabling and disabling the firewall.

3.2 OpenFlow Switches:

Contrary to traditional switches, OpenFlow switches do not perform any sort of learning functionality. They depend on the controller to provide the required intelligence, and in order to do so, it requires to be connected to the controller via a particular port. In our project, we use a

virtual emulation of an OpenFlow Switch. This can be done by either installing OpenVSwitch or Mininet. Even though in traditional networks, the switches have a dedicated hardware meant for them, in this case, we are going to use a software emulation of it. We are going to do the emulation on the End machine. OpenVSwitch emulates an OpenFlow switch, and allows us to connect to other ports. It even allows us to set up tap ports and connect those ports to Virtual Machines.

Mininet is a network simulator. It simulates the network with the required no of hosts and switches. Mininet inturn uses OpenVSwitch as a switch while simulation the network. however, it automatically add the tap ports so that it could be connected to the hosts.

3.3 End Device:

The end device could be a physical machine which could be running on any Operating System such as Ubuntu, Windows, Mac etc. It should have a Network Interface Card (NIC) so that it could access the internet (and thereby the controller). Further, it should have the necessary softwares which utilize the internet, such as a browser or a ftp client. The device should be able to host a virtual machine over it. And for this purpose, it would require any virtualization software such as VirtualBox. In our implementation, these virtual boxes needs to be connected to the OpenFlow Switches, and in order to do so, the device should allow tap ports to be created.

Chapter 4

Design

The design for implementing our project depends on various softwares and technologies. An introduction about the major softwares and technologies are explained below.

4.1Software Defined Networking

Software-defined networking is an approach to computer networking that allows network administrators to manage 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). SDN is defined as a network architecture with four pillars.

- The control and data planes are decoupled. Control functionality is removed from network devices that will become simple (packet) forwarding elements.
- Forwarding decisions are flow based, instead of destination based. A flow is broadly defined by a set of packet field values acting as a match (filter) criterion and a set of actions (instructions). In the SDN/Open Flow context, a flow is a sequence of packets between a source and a destination. All packets of a flow receive identical service policies at the forwarding devices. The flow abstraction allows unifying the behaviour of different types of network devices, including routers, switches, firewalls, and middle boxes. Flow programming enables unprecedented flexibility , limited only to the capabilities of the implemented flow tables.
- Control logic is moved to an external entity, the so-called SDN controller or NOS. The NOS is a software platform that runs on commodity server technology and provides the essential resources and abstractions to facilitate the programming of forwarding devices based on a logically centralized, abstract network view. Its purpose is therefore similar to that of a traditional operating system.
- The network is programmable through software applications running on top of the NOS that interacts with the underlying data plane devices.

Centrally placing the intelligence of a network system (e.g., the intelligence is logically centralized by a so-called SDN controller) increases the flexibility of network utilization, it also keeps its complexity hidden from operators which is why SDN is easy to operate and maintain. SDN also introduces the abstraction of lower network infrastructure functionality, which is well suited for the efficient development of new applications, and thereby promotes SDN as helpful in bringing innovative services to the market in a more timely manner. SDN can realize all of this in a cost-effective manner (capital expenditure/operating expenditure) there also exist a variety of other proposals and technical points of focus regarding SDN, such as investigating how to support coexistence with existing devices.

4.1.1 SDN Architecture

A software-defined networking (SDN) architecture (or SDN architecture) defines how a networking and computing system can be built using a combination of open, software- based technologies and commodity networking hardware that separate the control plane and the data layer of the networking stack. The architecture is shown in figure 2.

In the SDN architecture, the splitting of the control and data forwarding functions is referred to as disaggregation, because these pieces can be sourced separately, rather than deployed as one integrated system. This architecture gives the applications more information about the state of the entire network from the controller, as opposed to traditional networks where the network is application aware.

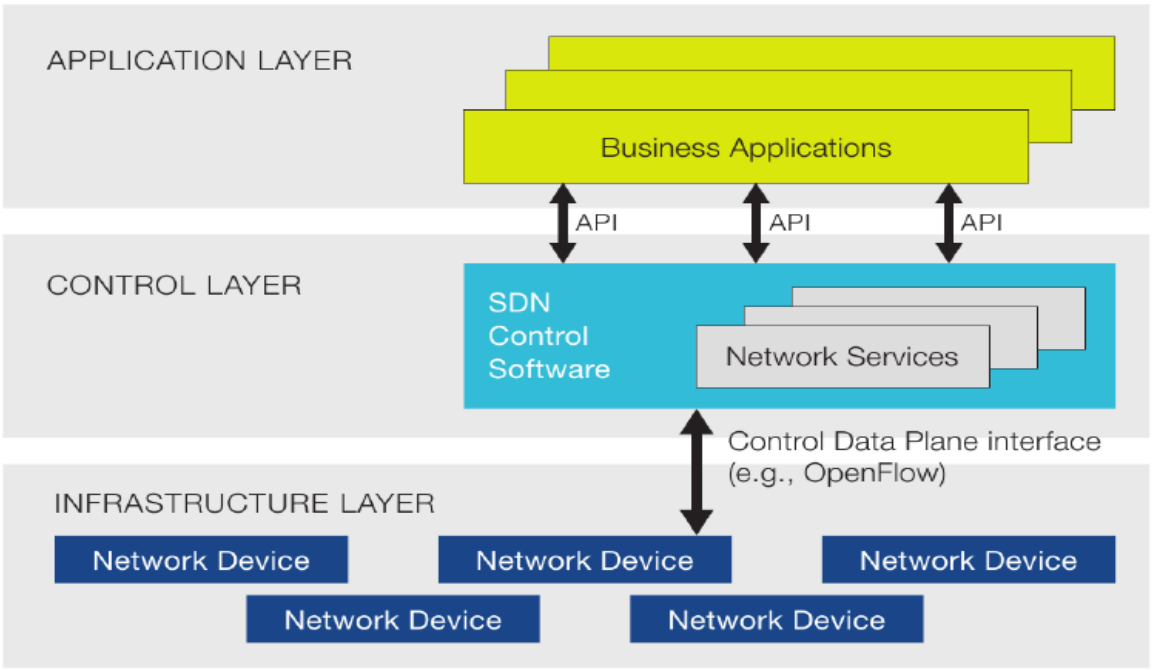

Figure 4.1: Architecture of SDN.

SDN architectures generally have three components or groups of functionality: _ SDN Applications: SDN Applications are programs that communicate behaviours And the needed resources with the SDN Controller via application programming interface (APIs). In addition, the applications can build an abstracted view of the network by collecting information from the controller for decision-making purposes. These applications could include networking management, analytics, or business applications used to run large data centres. For example, an analytics application might be built to recognize suspicious network activity for security purposes. _ SDN Controller: The SDN Controller is a logical entity that receives instructions or requirements from the SDN Application layer and relays them to the networking components. The controller also extracts information about the network from the hardware devices and

communicates back to the SDN Applications with an abstract view of the network, including statistics and events about what is happening. _ SDN Networking Devices: The SDN networking devices control the forwarding 11 and data processing capabilities for the network. This includes forwarding and processing of the data path.
The SDN architecture APIs are often referred to as northbound and southbound inter- faces, defining the communication between the applications, controllers, and networking systems. A Northbound interface is defined as the connection between the controller and applications, whereas the Southbound interface is the connection between the controller and the physically networking hardware. Because SDN is a virtualized architecture, these elements do not have to be physically located in the same place.

4.2 OpenFlow

OpenFlow is a communications protocol that gives access to the forwarding plane of a network switch or router over the network. OpenFlow enables network controllers to determine the path of network packets across a network of switches. The controllers are distinct from the switches. This separation of the control from the forwarding allows for more sophisticated traffic management than is feasible using access control lists (ACLs) and routing protocols. Also, OpenFlow allows switches from different vendors often each with their own proprietary interfaces and scripting languages to be managed remotely using a single, open protocol. The protocol's inventors consider OpenFlow an enabler of software defined networking (SDN). OpenFlow allows remote administration of a layer 3 switch's packet forwarding tables, by adding, modifying and removing packet matching rules and actions. This way, routing decisions can be made periodically or ad hoc by the controller and translated into rules and actions with a configurable lifespan, which are then deployed to a switch's ow table, leaving the actual forwarding of matched packets to the switch at wire speed for the duration of those rules. Packets which are unmatched by the switch can be forwarded to the controller. The controller can then decide to modify existing ow table rules on one or more switches or to deploy new rules, to prevent a structural ow of traffic between switch and controller. It could even decide to forward the traffic itself, provided that it has told the switch to forward entire packets instead of just their header.
The OpenFlow protocol is layered on top of the Transmission Control Protocol (TCP), and prescribes the use of Transport Layer Security (TLS). Controllers should listen on TCP port 6653 for switches that want to set up a connection.

4.3 Pox Controller:

Pox Controller is an open-source development platform licensed under Apache License 2.0.It is available for Linux, Windows and Mac OS and may other operating systems. However it requires that Python 2.7 be installed on the device. This SDN Controller is responsible for maintaining all of the network rules and providing the necessary instructions to the underlying infrastructure on how traffic should be handled. This enables businesses to better adapt to their changing needs and have better control over their networks.

It handles all the southbound communications with the networking devices. It communicates with the OpenFlow switches using OpenFlow1.0 protocol. And it provides the necessary API for the northbound communications. Pox controller is a component based controller. These components can be used to provide functionalities to the networking devices.

4.4 VirtualBox:

VirtualBox is a high performance virtualization product which is freely available as Open Source Software under the GNU General Public License version 2. VirtualBox is available for Linux, Windows, Mac and Solaris. And it has a support for a huge number of guest operation systems. it was initially developed by Innotek GmbH, and later was acquired by Sun Microsystems in 2008. And Sun MicroSystems was in turn acquired by Oracle. It offers Hardware-assisted virtualization by supporting both Intel VT-x and AMD-v hardware-virtualization. In absence of any hardware assistance, it also provides software based virtualization. However Software based virtualization supports only 32-bit guest OSs.

It allows various network configurations such as NAT (Network address translation), NAT network, Host-only adapter, Bridged Networks. And it emulates the hard disks as one of the three image formats VDI, VMDK or VHD.

4.5 Amazon EC2:

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. It allows complete control of the computing resources and reduces the time required to obtain and boot new devices in minutes, allowing scaling the devices up and downs as per our requirements.

Few of the benefits of amazon EC2:

- Elastic Web-Scale Computing: Allows decreasing and increasing the capacities of the devices within minutes.
- Completely Controlled: The user will have the complete control of his instances including root access and the ability to interact with them.
- Integrated: Amazon EC2 is integrated with many of the others AWS services such as Amazon Relational Database Service, Amazon Virtual Private Cloud, and Amazon Simple Storage Service.
- Secure: The devices hosted by users are provides with networking architecture which is built to meet a great level of security.

Chapter 5

Implementation

5.1 Setting up the OpenVSwitch

A basic requirement for setting up our project requires an OpenVSwitch to be used and it needs to be set up as the default route which the OS would select while sending the packets out of the device.

Initially the device would be configured so that it would send all the data packets through the default Ethernet card i.e. the NIC card.

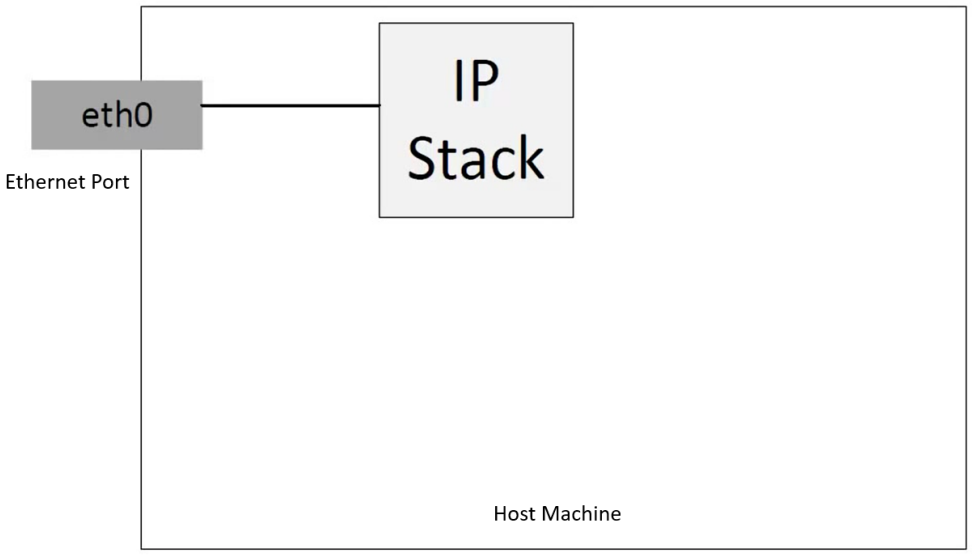

Fig 5.1: Internal network connection overview before setting up the OpenVSwitch

As we can see in the above diagram, the IP Stack of the OS is directed towards the default NIC card. The default route taken by the packet can be seen using the "route -n " command.

In order to implement our project, first OpenVSwitch has to be installed and then the default route selected by the IP Stack should be set through the OpenVSwitch.

OpenVSwitch could be installed through the command "sudo apt-get install openvswitch-switch"

```
sois@sois-HP-280-G1-MT:~$ sudo apt-get install openvswitch-switch
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Fig 5.2: Installing OpenVSwitch

After installing the OpenVSwitch, a switch has to be created. This can be done using the command " sudo ovs-vsctl add-br mybridge" where "mybridge" would be the name of the switch.

Once the OpenVSwitch has been added, it should be connected to the default Ethernet port. This has to be done so that, any packet sent into the switch could be passed out of the device through the Ethernet port. Doing so satisfies one of the requirements to obtain internet connectivity through the switch.

Then, the IP address corresponding to the default Ethernet port has to be removed. And an IP address has to be assigned to the switch. The IP address for the switch could be obtained through the same DHCP server which serves the Ethernet port.

These actions could be performed through the following two commands

“ sudo ifconfig eth0 0”

“ sudo dhclient mybridge”

After execution of these commands, an IP address will be assigned to the switch created. This IP address can be used for communication with other devices.

```
mybridge  Link encap:Ethernet  HWaddr 08:00:27:30:46:37
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe30:4637/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:114 errors:0 dropped:0 overruns:0 frame:0
          TX packets:113 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:16645 (16.6 KB)  TX bytes:104760 (104.7 KB)
```

Fig 5.3: IP configurations of the OpenVSwitch

The next step would be to create tap ports, and connect one of the end of the tap to the switch and the other end to the virtual machine.

Tap port can be created using the command "sudo ip tuntapp add mode tap vport1" where vport1 would specify the name of the tap. After creating the tap, it has to be switched up using the command " sudo ifconfig vport1 up"

As mentioned earlier, after creating the tap ports, its ends should be connected to the switch and the virtual machine. In order to connect it to the Switch, the following command can be used " sudo ovs-vsctl add-port mybridge vport1"

And then, we can check the list of ports connected to the switch using the command "sudo ovs-vsctl show"

```
aa@aaa-VirtualBox:~$ sudo ovs-vsctl add-port mybridge vport1
aa@aaa-VirtualBox:~$ sudo ovs-vsctl show
041a798-82c6-4c61-96d0-756ac8ad5c1c
    Bridge mybridge
        Port "enp0s3"
            Interface "enp0s3"
        Port mybridge
            Interface mybridge
                type: internal
        Port "vport1"
            Interface "vport1"
    ovs_version: "2.5.2"
```

Fig 5.4: OpenVSwitch ports and interfaces

Finally, the last step would be to connect the other end of the port to the virtual machine. VirtualBox, the virtualization product that we have used in our project allows the virtual machines to be connected to the tap ports through a bridged interface. By selecting that particular interface, we can connect the VMs to the tap port.

Once, all the above set up procedures has been completed, the VMs would then be connected to the external network. And the virtual infrastructure created could be visualized as below.

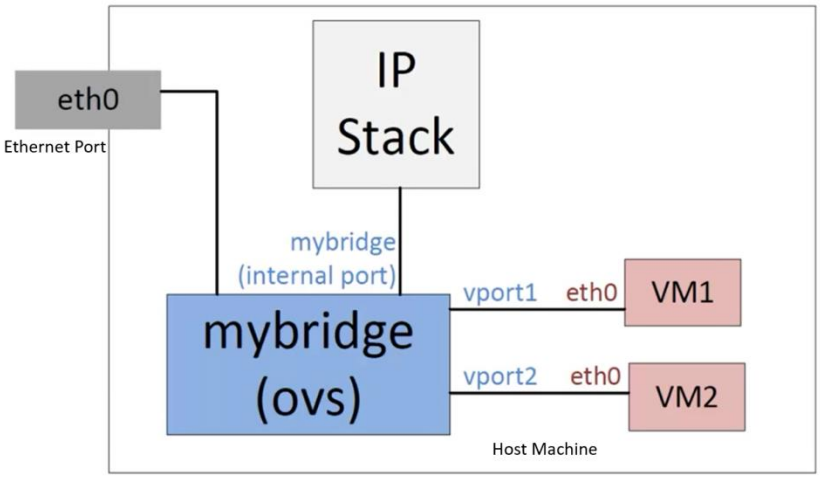

Fig 5.5: Internal network connection overview after setting up the OpenVSwitch

5.2 Setting up the cloud based controller:

As mentioned above, in order to implement a cloud based controller, we have used the Amazon EC2 service. To set up the controller, we have created a VM (running on Ubuntu 16.04 Server) and then hosted the controller.

The Virtual Machine can be created using the 'Launch instance" option in the Amazon EC2 service.

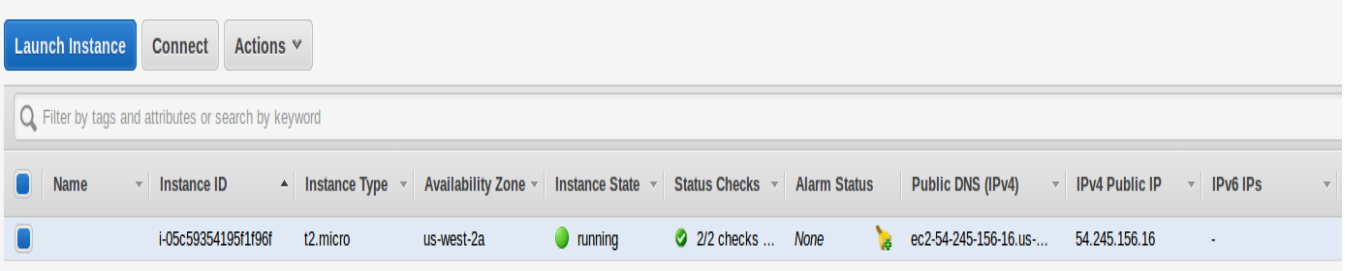

Fig 5.6: Creating a Virtual Machine on Amazon EC2

After creating the controller, the POX Controller has be installed by simply downloading the required components from POX Controller's official GitHub repository. And the POX Controller can be booted up by running the pox.py program which comes as a part of the download. Further, any additional features (such as our firewall app) has to be written separately, and has to be passed as a CLA (Command Line Argument) while booting up the POX Controller i.e. while running the pox.py program as shown below

```
john@john-Lenovo-Flex-2-15:~/Downloads/pox-carp/ext$ python ../pox.py forwarding.l2_learning
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
```

Fig 5.7: Booting up POX Controller

5.3 Implementation using Mininet:

Mininet being a network simulator, takes care of creating the hosts and switches. It even creates the default controller if no external controller is specified. It establishes connections between the hosts and the switches as well. But the mininet hosts are not connected to the internet by default. However, it could be achieved by following the steps as mentioned in the chapter 5.1.

A mininet network can be emulated by simply using the "mn" command and specifying the other arguments as shown below.

```
aaa@aaa-VirtualBox: ~
aaa@aaa-VirtualBox:~$ sudo mn --topo=linear,4 --controller=remote,ip=54.210.45.6
2:6633
[sudo] password for aaa:
*** Creating network
*** Adding controller
```

Fig 5.8: Creating a mininet topology

The above command specifies that 4 mininet hosts need to be created, and each one of them should be connected to separate switches. The second argument specifies that the default controller need not be used. Instead a remote controller with an IP address 54.210.45.62 has to be connected. Further, it has been specified that the controller is listening through the port 6633.

The controller that we have setup here is a remote controller hosted on the amazon cloud with a public IP address of 54.210.45.62. After executing the command, a virtual network infrastructure would be built. An example of such a setup is as shown below.

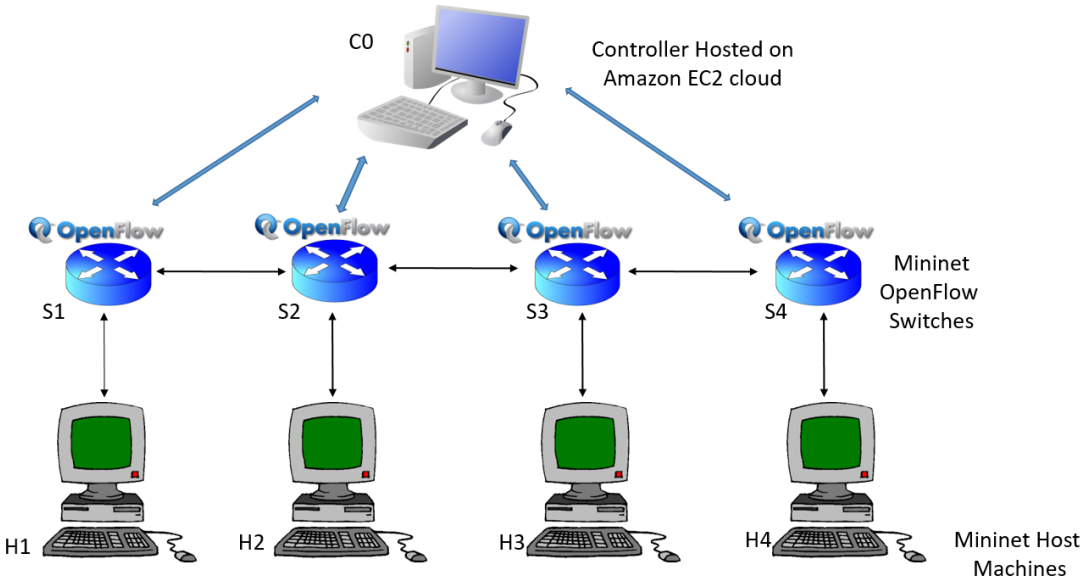

Fig 5.9: Figure setup with Mininet

Once the network has been set up, the hosts will be able to communicate with each other. This can be checked by pinging each other. Also, since we have connected our mininet hosts with the internet, the mininet hosts can access the internet. This can be checked by opening up a browser in any of the hosts and accessing any website.

Now, in order to perform the firewall function, we have built a GUI based application through which we can enable firewall on certain hosts or certain websites. This application would be running on the cloud based controller. This would allow firewalling to be done centrally, instead of having to enable firewall on each of the connected devices individually. The application that we have built can be seen below:

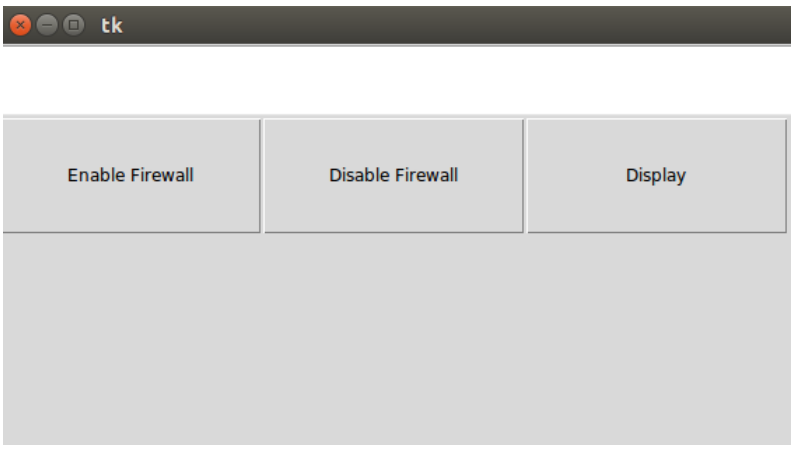

Fig 5.10: Firewall Application

This application can perform three types of activities:

Enable Firewall: This would enable the firewall on whichever IP address that has been provided by the user in Text Box. The input given by the user can not only be an IP address, it could also be the domain name of a website. Further, multiple values can be given, each of them being separated by a semicolon.

Disable Firewall: Similar to the Enable Firewall option, this would disable the firewall on whichever IP address that has been provided by the user in Text Box. The input given by the user can not only be an IP address, it could also be the domain name of a website. Further, multiple values can be given, each of them being separated by a semicolon.

5.4 Implementation using VirtualBox:

As mentioned above, mininet performs the task of setting up tap ports and connecting them to the virtual hosts, however in case of building a network infrastructure consisting of virtual machines created using VirtualBox, we are responsible for setting up the tap ports and connecting the VMs to them. The steps for setting up the tap ports have been mentioned in the chapter 5.1. Once, the tap port have been set up, we need to connect the virtual machines to it. In VirtualBox, we can directly connect the virtual machines to the tap ports using the "Bridged Adapter" mode of connection as shown below.

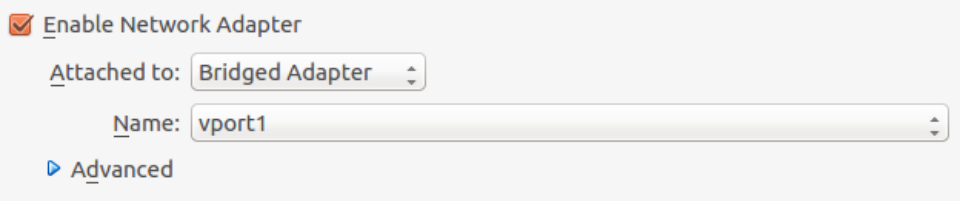

Fig 5.11: Connecting Virtual Machine to tap port

Once the devices have been connected to the tap ports, and thereby the OpenVSwitch, a virtual network infrastructure would be created. However such a network would be local to that particular device which hosts the virtual machines. In order to bring all the devices under observation into a single control plane, we have to setup a controller and connect the switches to it. The controller has be hosted on any on the devices (if the devices are in the same network) or it can be hosted on the cloud. Such a setup could be as seen below.

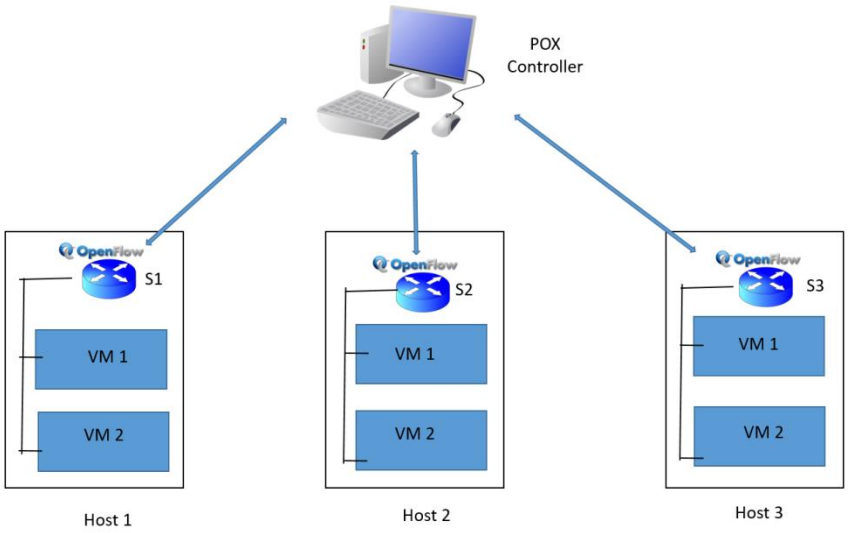

Fig 5.12: Network setup with OpenVSwitch and VirtualBox

Chapter 6

Test Documentation

6.1 Firewall Test

After connecting mininet to the internet we can check the connectivity by opening up a browser in any of the hosts, and checking if a desired page opens up.

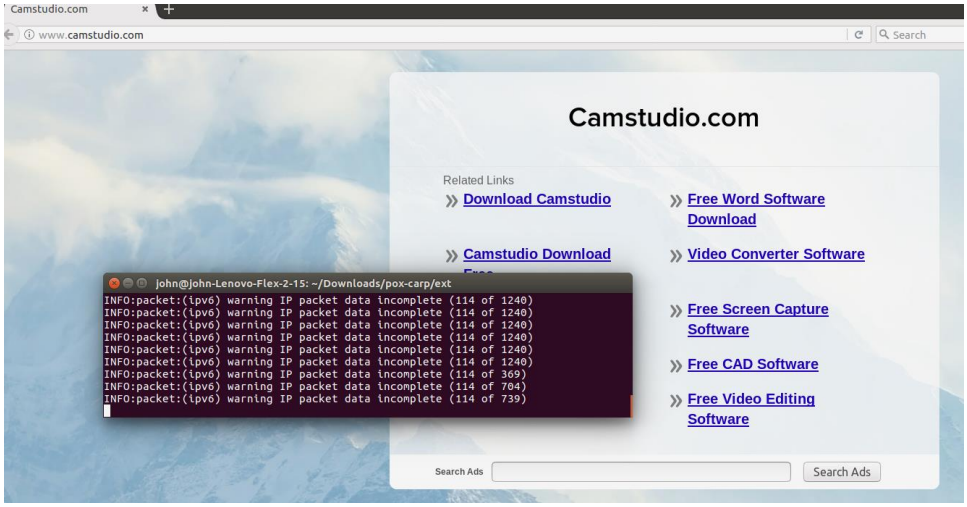

Fig 6.1(a): Accesing a websitte before enabling firewall

In the above example, we have opened the web page www.camstudio.com. The page loads up as expected. But once, the firewall is enabled using our firewall application, the website fails to load up. As seen below:

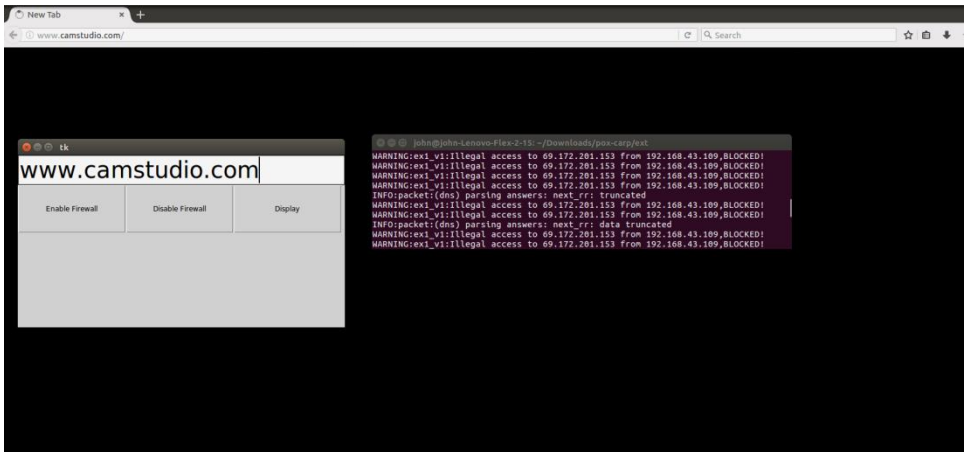

Fig 6.1 (b): Accessing a website after enabling firewall

The same results can be expected when the VirtualBox implementation of our project is considered.

6.2 Speed Test:

Even though using a controller along with the switches facilitates a better management of the network infrastructure, it has to be noted that it adds an additional node through which the packets must transverse in order to reach the destination.

Presently due to advancements in technology, in most of the cases network bandwidth doesn't act as a bottleneck in an organization or even in domestic usage. However, it is necessary to check the performance overhead caused by adding an additional node. The following diagram depicts the results of the speed tests performed in three situations.

1. Without any controller
2. With a controller, hosted locally
3. With the controller hosted on the Amazon EC2 cloud.

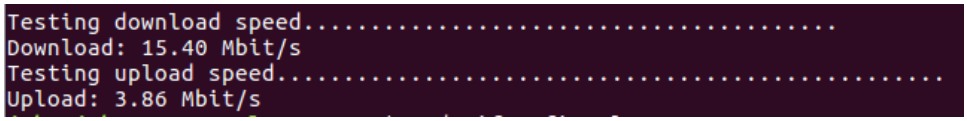

Fig 6.2(a): Speed Test without any controller

Fig 6.2(b): Speed Test with controller hosted locally

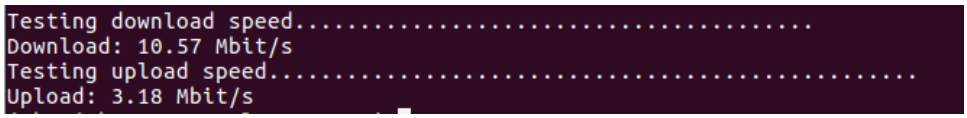

Fig 6.2(c): Speed Test with controller hosted on Amazon EC2

From the results, it can be seen that, while using a locally hosted controller, the performance overhead caused is negligible. However, when a cloud based controller is used, the performance overhead is significant. Hence, depending on the requirement, keeping in mind the tradeoff between ease of management and network speed, the appropriate setup has to be chosen.

Chapter 7

References

[1] Wikipedia, "Software-defined networking — wikipedia, the free encyclopedia," 2017, [Online; accessed 10-June-2017]. [Online]. Available: https://en.wikipedia.org/w/ndex.php?title=Software-definednetworking&oldid=724491793

[2] D. Kreutz, F. M. V. Ramos, P. E. Verssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," Proceedings of the IEEE , vol. 103, no. 1, pp. 14–76, Jan 2015.

[3] SDxCentral, "Understanding the SDN architecture," 2016, [Online; accessed 5-July-2017]. [Online]. Available: https://www.sdxcentral.com/sdn/definitions/inside-sdn-architecture/

[4] Wikipedia, "Openflow — wikipedia, the free encyclopedia," 2016, [Online;accessed 5-July-2017].[Online].Available: https://en.wikipedia.org/wiki/OpenFlow/

[5] N. H. B. H. Bob Lantz and V. Jeyakumar. (2015) Introduction to mininet. [Online]. Available: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

[6] Amazon Elastic Compute Cloud Documentation (2017) [Online;accessed 10-Oct-2017].[Online] Available: https://aws.amazon.com/documentation/ec2/

[7] POX Wiki,(2013) [Online;accessed 01-Oct-2017].[Online] Available: https://openflow.stanford.edu/display/ONL/POX+Wiki

Appendix

Source Code

1.firewall.py

```
#Program to implement firewall function '"
#
# The list of IPs and domains to be blocked or unblocked are put into
# or removed from a file called 'fw_rules.txt' respectively



from Tkinter import *
import logging
logging.basicConfig(level="DEBUG")
logger=logging.getLogger()

def block_ip():
 '" Program to perform enabling of firewall'"
 data=temp_data.get()
 all_ips=data.split(";")
 #print("1:",all_ips)
 file_ips=[]

 with open("fw_rules.txt","r") as fw:
  file_data=fw.read()
  #print ("6:",file_data)
  file_ips=file_data.split('\n')
  #print ("5:",file_ips)
```

```
 with open("fw_rules.txt","a+") as fw:
  for i in all_ips:
   # check if already present
   if i not in file_ips:
    #print("4:",file_ips)
    fw.write('\n'+i)



def unblock_ip():
 ''' Program to perform disabling of firewall'''
 data=temp_data.get()
 all_ips=data.split(";")
 file_data=[]
 with open("fw_rules.txt") as fw:
  file_data=fw.read()

 with open("fw_rules.txt","w+") as fw:
  for i in all_ips:
   file_data=file_data.replace('\n'+i,'')
  fw.write(file_data)
def display_ip():
 ''' Program to display the list of blocked IPs and Domains'''
 with open("fw_rules.txt","r") as fw:
  file_data=fw.read()
  print ("\n")
  file_ips=file_data.split('\n')
  print ("\n".join(file_ips[1:]))
```

```
# initializing and deploying the GUI components
root = Tk()
root.geometry('570x300')
topFrame = Frame()
temp_data=StringVar()
topFrame.pack()
block_button= Button(topFrame,text="Enable Firewall",command=block_ip,height=5,width=20)
unblock_button= Button(topFrame,text="Disable
Firewall",command=unblock_ip,height=5,width=20)
display_button= Button(topFrame,text="Display",command=display_ip,height=5,width=20)
text_field= Entry(topFrame,textvariable=temp_data,width=100,font=('Verdana',30)).pack()


block_button.pack(side="left")
unblock_button.pack(side="left")
display_button.pack(side="left")
root.mainloop()
```

2 packet_check.py

```
# Checks the list of websites present in “fw_rules.txt” and enables firewall on them
"""An L2 learning switch.
It is derived from one written live for an SDN crash course.
It is somwhat similar to NOX's pyswitch in that it installs
exact-match rules for each flow.
"""
from pox.core import core
```

```
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpid_to_str
from pox.lib.util import str_to_bool
log = core.getLogger()
class LearningSwitch (object):
  """  The learning switch "brain" associated with a single OpenFlow switch.  """

  def __init__ (self, connection, transparent):
    # Switch we'll be adding L2 learning switch capabilities to
    self.connection = connection
    self.transparent = transparent
    # Our table
    self.macToPort = {}
    # We want to hear PacketIn messages, so we listen
    # to the connection
    connection.addListeners(self)
    # We just use this to know when to log a helpful message
    self.hold_down_expired = _flood_delay == 0
    #log.debug("Initializing LearningSwitch, transparent=%s",
    #          str(self.transparent))
  def _handle_PacketIn (self, event):
    """
    Handle packet in messages from the switch to implement above algorithm.
    """
    packet = event.parsed
    def flood (message = None):
      """ Floods the packet """
      msg = of.ofp_packet_out()
```

```
    if time.time() - self.connection.connect_time >= _flood_delay:
      # Only flood if we've been connected for a little while...
      if self.hold_down_expired is False:
        # Oh yes it is!
        self.hold_down_expired = True
        log.info("%s: Flood hold-down expired -- flooding",
            dpid_to_str(event.dpid))
      if message is not None: log.debug(message)
      #log.debug("%i: flood %s -> %s", event.dpid,packet.src,packet.dst)
      # OFPP_FLOOD is optional; on some switches you may need to change
      # this to OFPP_ALL.
      msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
    else:
      pass
      #log.info("Holding down flood for %s", dpid_to_str(event.dpid))
    msg.data = event.ofp
    msg.in_port = event.port
    self.connection.send(msg)
  def drop (duration = None):
    """
    Drops this packet and optionally installs a flow to continue
    dropping similar ones for a while
    """
    if duration is not None:
      if not isinstance(duration, tuple):
        duration = (duration,duration)
      msg = of.ofp_flow_mod()
      msg.match = of.ofp_match.from_packet(packet)
```

```
        msg.idle_timeout = duration[0]
        msg.hard_timeout = duration[1]
        msg.buffer_id = event.ofp.buffer_id
        self.connection.send(msg)
      elif event.ofp.buffer_id is not None:
        msg = of.ofp_packet_out()
        msg.buffer_id = event.ofp.buffer_id
        msg.in_port = event.port
        self.connection.send(msg)
    self.macToPort[packet.src] = event.port # 1
#####################################################################
#  FIREWALL


    ip=packet.find('ipv4')
    if ip is None:
        pass
    else:
        log.debug("___Packet moving from %s - %s"%(ip.srcip,ip.dstip))
        with open("fw_rules.txt") as rl:
            data=rl.read()
            dangers=data.split('\n')
            if ip.dstip in dangers[1:]:
                log.warning("Illegal access to %s from %s,BLOCKED!"%(ip.dstip,ip.srcip))
                drop()
                return
```

```
#######################################################################
  if not self.transparent: # 2
    if packet.type == packet.LLDP_TYPE or packet.dst.isBridgeFiltered():
      drop() # 2a
      return
  if packet.dst.is_multicast:
    flood() # 3a
  else:
    if packet.dst not in self.macToPort: # 4
      flood("Port for %s unknown -- flooding" % (packet.dst,)) # 4a
    else:
      port = self.macToPort[packet.dst]
      if port == event.port: # 5
        # 5a
        log.warning("Same port for packet from %s -> %s on %s.%s.  Drop."
            % (packet.src, packet.dst, dpid_to_str(event.dpid), port))
        drop(10)
        return
      # 6
      log.debug("installing flow for %s.%i -> %s.%i" %
              (packet.src, event.port, packet.dst, port))
      msg = of.ofp_flow_mod()
      msg.match = of.ofp_match.from_packet(packet, event.port)
      msg.idle_timeout = 10
      msg.hard_timeout = 30
      msg.actions.append(of.ofp_action_output(port = port))
      msg.data = event.ofp # 6a
      self.connection.send(msg)
```

```
class l2_learning (object):
  """
  Waits for OpenFlow switches to connect and makes them learning switches.
  """
  def __init__ (self, transparent):
    core.openflow.addListeners(self)
    self.transparent = transparent
  def _handle_ConnectionUp (self, event):
    log.debug("Connection %s" % (event.connection,))
    LearningSwitch(event.connection, self.transparent)


def launch (transparent=False, hold_down=_flood_delay):
  """
  Starts an L2 learning switch.
  """
  try:
    global _flood_delay
    _flood_delay = int(str(hold_down), 10)
    assert _flood_delay >= 0
  except:
    raise RuntimeError("Expected hold-down to be a number")
  core.registerNew(l2_learning, str_to_bool(transparent))
```