

TABLE OF CONTENTS

1 Introduction	3
2 Design Requirements	3
3 Network Design	8
4 Network Topology Creation in Mininet	10
4.1 Mininet Design	10
4.2 Demonstration of running a server	15
4.3 Implementation	19
4.4 Internet connection to hosts	22
5 Network Configuration and Application Development	23
5.1 VLAN switching	23
5.2 L2L3 switching and routing	37
5.3 Firewall Application	46
5.4 Implementation Benefits in P4 switches	46
5.4 SNORT IDS	46
5.5 Practical open flow switches for implementation of this topology	54
6 References	64
7 Appendices 1 – 6	65

INTRODUCTION

The document describes the implementation of the SDN enterprise network for NewGen Networks Pty. Ltd. We have designed a network diagram based on the requirements of the NewGen Networks Pty. Ltd. Our design is considered for future growth and is fully scalable based on the three-tier hierarchical network topology. We have provided redundancy in our design just in case of failure, another device or switch takes over the failed one. We have used VLANs to isolate users on different floors for management and security. With the help of VLANs, we can separate the users in both the same floor and different floor to make it easy to set firewall rules with VLAN names or IP addresses. We have defined a separate VLAN for wireless connectivity in each floor, and users can use their mobiles or laptops to connect to the network by connecting to that VLAN. and preventing unauthorized access inside the network. Based on the requirements, we have enabled the firewall to define some certain rules to prevent unauthorized access to or from a private network. Moreover, we have enabled the SNORT intrusion detection system in order to detect intrusion in the network and also collect important logs for the sake of both troubleshooting and documentation.

DESIGN REQUIREMENTS

The requirements given in the question can be broken down as given below:

- The enterprise has a three-floor structure
- **First Floor has the following networks:**

- Seven offices have one host each, which is connected via ethernet to a server room.
- The server room consists of all the networking infrastructure needed for both the Intranet and Internet connectivity.
- Seminar room with a PC for the instructor and 40 lab PCs which are used by the students.
 - ✓ The Instructor PC must access to the entire Intranet and Internet
 - ✓ The lab PCs should only have access to the Internet
- **Second Floor (Software House) has the following networks:**
 - Ten cubicles with 1 PC each - access to the server room and internet
 - Seven offices have one host each, which is connected via ethernet to a server room.
- **Third Floor has the following networks:**
 - R&D lab - 30 computers with access to 5 dedicated servers in the server room - access to the internet.
 - Four offices - access to the internet.
 - One office – access to R&D server room and internet.

The number of hosts and access requirements are shown in the below table. The blanks are those which are not explicitly defined, and we assumed them to be 'Yes' for server access and 'Yes' for intranet access if it is an office space.

Floor Number	Item Name	Number of Hosts	Server access	Dedicated server access	Internet access	Intranet access
Floor 1	Office No.1	7	Yes	No	Yes	Yes

Floor 1	Seminar room	40		No	Yes	No
Floor 1	Seminar room	1	Yes	No	Yes	Yes
Floor 2	Cubicles	10	Yes	No	Yes	
Floor 2	Office No.2	7	Yes	No		
Floor 3	R&D lab	30		Yes	Yes	
Floor 3	Office No.3	4	No	No	Yes	
Floor 3	Office No.3	1	Yes	Yes	Yes	

- **Main Requirements:**

- Design the topology of the enterprise network
- Implement the topology in mininet
- Switching and/or Routing application to provide full connectivity, both Internally and externally to the internet.
- Implement Security and Firewall policies for protecting the network. Install rules, which stops certain types of applications/ports from being opened. Also, develop rules which limit bandwidth for certain parts of the network.
- Develop a visualization tool which shows the live state of the network and packet flow information.
- Develop an IDS (Intrusion detection system) for your network and illustrate the operation of your IDS using sample scenarios and illustrations.
- Modify your design to create a high availability architecture, both in terms of switches and controller (Hint: You can use a controller which provides high availability).

- **Additional Requirements:**

- Wireless access throughout the building on all floors through WiFi.

- Recommendations for restricted internet usage with blocked sites like social media.
- Implement network security to minimize external attacks
- Draw insights of implementing the SDN using P4 technology

The main requirements are shown in the table below.

Requirements Number	Requirements
SDN01	Detailed design document for the network
SDN02	Implement the designed topology in mininet
SDN03	Configure the network
SDN04	Implement routing and switching applications for connectivity
SDN05	Security and firewall policy implementation
SDN06	Implementation benefits in P4
SDN07.1	Develop visualization tool for portraying live network states
SDN07.2	Develop an Intrusion Detection System
SDN07.3	Modify design with high availability controllers

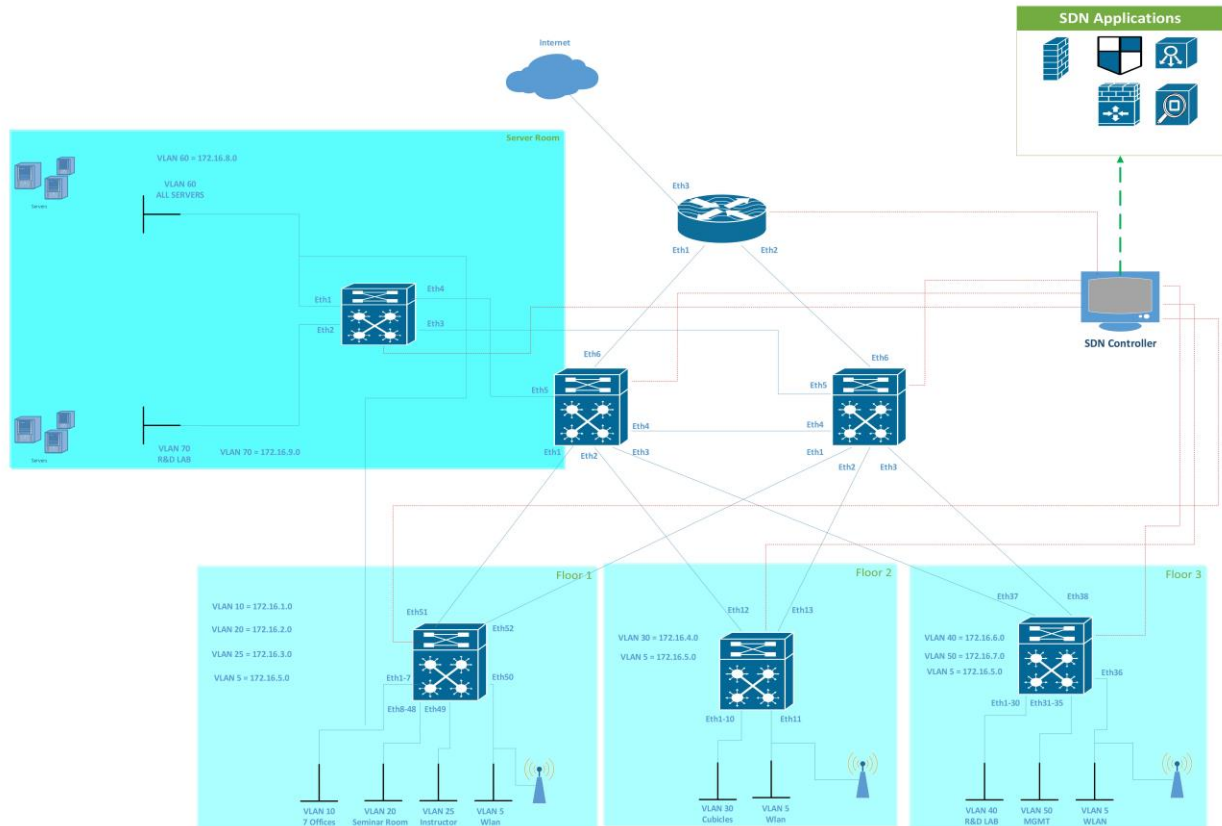
The requirements have been converted to requirement specifications in the below table.

Requirements Number	Specifications Number	Requirements
SDN01	RS01	Identify the tool for network diagram and design
	RS02	Identify design of VLANs in each floor
SDN02	RS03	Identify mininet topology creation using VLANs

	RS04	Identify server creation in mininet
SDN03	RS05	Identify MAC and IP address assignment
SDN04	RS06	Identify Ryu App to run routing and switching apps
SDN05	RS07	Identify Ryu App to run a firewall and add rules
SDN06	RS08	Research on the benefits of P4
SDN07.1	RS09	Identify OpenStack visualization tool
SDN07.2	RS10	Identify and implement SNORT - IDS
SDN07.3	RS11	Identify benefits of zookeeper for high availability

PART 1: NETWORK DESIGN

Microsoft Visio 2019 has been used to design the network topology. The overall network topology is shown below.



We have divided the above network into 11 VLANs. Based on the network diagram, the specifications of Layer 2 and 3 design are as follow:

Floor 1:

We have added 4 VLANS for the floor one. VLAN5 is used commonly to all the floors for providing WiFi access.

VLAN Number	VLAN Name	VLAN Subnet	Interfaces	Number of Hosts

VLAN 10	7 Offices	172.16.1.0	Eth1-7	7
VLAN 20	Seminar Room	172.16.2.0	Eth8-48	40
VLAN 30	Instructor PC	172.16.3.0	Eth49	1
VLAN 5	Wireless	172.16.5.0	Eth50	-

Floor 2:

Floor 2 has been divided into two VLANs.

VLAN Number	VLAN Name	VLAN Subnet	Interfaces	Number of Hosts
VLAN 30	Cubicle	172.16.4.0	Eth1-10	10
VLAN 5	Wireless	172.16.5.0	Eth11	-

Floor 3:

Floor 3 has been divided into three VLANs.

VLAN Number	VLAN Name	VLAN Subnet	Interfaces	Number of Hosts
VLAN 40	R&D LAB	172.16.6.0	Eth1-30	30
VLAN 50	MGMT	172.16.7.0	Eth31-35	5
VLAN 5	Wireless	172.16.5.0	Eth36	-

Server Room:

The server room has two VLANs – One is for the common servers and one for the dedicated servers. We have assumed the VLAN60 has ten common servers because it is not given in the question. VLAN 70 has five dedicated servers.

VLAN Number	VLAN Name	VLAN Subnet	Servers
VLAN 60	All Servers	172.16.8.0	DNS, DHCP, AD, FTP, etc
VLAN 70	R&D LAB	172.16.9.0	Based on the Requirements

We have used seven switches (multi-layer switches) in this topology. The switches are connected as shown below

Switch Number	Description	Connected to other switches
s1	Floor 1 VLANs	s4, s5
s2	Floor 2 VLANs	s4, s5
s3	Floor 3 VLANs	s4, s5
s4	Redundant Switch 1	s1, s2, s3, s6, s7
s5	Redundant Switch 2	s1, s2, s3, s6, s7
s6	Server switch	s4, s5
s7	Internet switch	s4, s5

PART 2: NETWORK TOPOLOGY DESIGN IN MININET

4.1 Mininet design of the topology:

We are using mid-level API in python to develop the custom topology in mininet i.e., by using definitions of functions directly in python. The only exception is the VLANHost class that we are using to define the VLANs. The entire mininet topology code is available in Appendix 1.

We initially import all the library files that are required for the implementation of this topology, as shown below.

```
#!/usr/bin/python
import sys
from functools import partial
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.topo import SingleSwitchTopo
from mininet.log import setLogLevel, info
from mininet.node import Host
from mininet.topo import Topo
from mininet.util import quietRun
from mininet.log import error
from mininet.node import OVSSwitch, Controller, RemoteController
from mininet.node import Controller, OVSKernelSwitch, RemoteController
import json
from mininet.moduledeps import pathCheck
import ipaddress

from sys import exit
import os.path
from subprocess import Popen, STDOUT, PIPE
```

The imported libraries are a lengthy list because we are running the topology directly from python. We have explained some of the main libraries we have utilized and their purpose below.

1. sys library is used because we call some system functionalities to print log level info and set up interfaces
2. functools library is utilized functional tool utilities
3. All mininet libraries are used for adding hosts, switches, links, configuring the links and so.
4. json is used for running an HTTP server.
5. sub process is used to run commands directly in the switch and hosts from this program.

The design of VLANs in mininet has been done using the below class. This VLANHost(Host) class will basically create a new VLAN layer 2 interface with the VLAN base ID that is sent.

The default “physical” interface IP is removed using

“ifconfig <interface name> inet 0”

Then, the new VLAN interface is created using

“vconfig add <interface name> <vlan base>”

```
class VLANHost(Host):
    # Host connected to Vlan interface
    def config(self, vlan=100, **params):
        r = super(VLANHost, self).config(**params)

        intf = self.defaultIntf()
        # remove IP from default, "physical" interface
        self.cmd('ifconfig %s inet 0' % intf)
        # create VLAN interface
        self.cmd('vconfig add %s %d' % (intf, vlan))
        # assign the host's IP to the VLAN interface
        self.cmd('ifconfig %s.%d inet %s' % (intf, vlan, params['ip']))
        # update the intf name and host's intf map
        newName = '%s.%d' % (intf, vlan)
        # update the (Mininet) interface to refer to VLAN interface name
        intf.name = newName
        # add VLAN interface to host's name to intf map
        self.nameToIntf[newName] = intf

    return r
```

The main function that defines all the hosts, servers, controllers, and switches is called `VLAN_Mininet()`. Initially, we create an object for the mininet called `net` with just a remote controller. The actual settings of the remote controller (`c1`) is defined in the second line. The program will try to connect port=6633 with IP=127.0.0.1 to connect to the controller. This is just defined in the code here so that when we run the Ryu switching and routing app, the switches in the mininet topology will automatically connect to the Ryu controller.

```
67 def VLAN_Mininet():
68     net = Mininet(controller=partial(RemoteController))
69     c1 = net.addController('c1', port=6633, ip='127.0.0.1')
```

We add the open flow switches using the below code and enable IP forwarding in them to handle IP packets also.

```

info('*****add switches\n')
s1 = net.addSwitch('s1')
s1.cmd('sysctl -w net.ipv4.ip_forward=1')
s2 = net.addSwitch('s2')
s2.cmd('sysctl -w net.ipv4.ip_forward=1')
s3 = net.addSwitch('s3')
s3.cmd('sysctl -w net.ipv4.ip_forward=1')
s4 = net.addSwitch('s4', )
s4.cmd('sysctl -w net.ipv4.ip_forward=1')
s5 = net.addSwitch('s5', )
s5.cmd('sysctl -w net.ipv4.ip_forward=1')
s6 = net.addSwitch('s6', )
s6.cmd('sysctl -w net.ipv4.ip_forward=1')
s7 = net.addSwitch('s7', )
s7.cmd('sysctl -w net.ipv4.ip_forward=1')

```

After this, we define hosts for different VLANs as shown below. Initially, all the variables for different VLAN bases are defined.

```

k=9 #Total number of VLANs
n=112
#Floor 1 VLANs
vlanBase1=10
vlanBase2=20
vlanBase3=25
vlanBase4=5
#Floor 2 VLANs
vlanBase5=30
vlanBase6=5
#Floor 3 VLANs
vlanBase7=40
vlanBase8=50
vlanBase9=5
#Common Server VLANs
vlanBase10=60
# Dedicated Server VLANs
vlanBase11=70

```

These variables are assigned to another temporary variable to be used to add host functionality.

```

vlan1 = vlanBase1
vlan2 = vlanBase2
vlan3 = vlanBase3
vlan4 = vlanBase4
vlan5 = vlanBase5
vlan6 = vlanBase6
vlan7 = vlanBase7
vlan8 = vlanBase8
vlan9 = vlanBase9
vlan10 = vlanBase10
vlan11 = vlanBase11

```

The hosts are added using the below “for” loops.

```
info('***Adding hosts of first floor (hf) and linking them to Switch 1\n')
j=1
for j in range(n):
    if (j>0 and j <= 7):
        name = 'hf%d' % (j)
        h = net.addHost(name, cls=VLANHost, vlan=vlan1, mac="01:00:00:00:00:%d" % (j), ip='172.16.1.%d/24' % (j + 1), defaultRoute='via 172.16.10.%d' % (j + 1))
        net.addLink(h, s1)
        ip = "172.16.10.%d" % (j+1)
        s1.cmdPrint("ifconfig s1-eth%d %s netmask 255.255.255.0" % (j,ip) )

    elif (j >= 8 and j < 49):
        name = 'hf%d' % (j)
        h = net.addHost(name, cls=VLANHost, vlan=vlan2, mac="01:00:00:00:00:%d" % (j), ip='172.16.2.%d/24' % (j - 6), defaultRoute='via 172.16.20.%d' % (j - 6))
        net.addLink(h, s1)
        ip = "172.16.20.%d" % (j-6)
        s1.cmdPrint("ifconfig s1-eth%d %s netmask 255.255.255.0" % (j,ip) )

    elif (j == 49):
        name = 'hf%d' % (j)
        h = net.addHost(name, cls=VLANHost, vlan=vlan3, mac="01:00:00:00:00:%d" % (j), ip='172.16.3.%d/24' % (j - 47), defaultRoute='via 172.16.30.%d' % (j - 47))
        net.addLink(h, s1)
        ip = "172.16.30.%d" % (j-47)
        s1.cmdPrint("ifconfig s1-eth%d %s netmask 255.255.255.0" % (j,ip) )

    elif (j == 50):
        name = 'hf%d' % (j)
        h = net.addHost(name, cls=VLANHost, vlan=vlan4, mac="01:00:00:00:00:%d" % (j), ip='172.16.5.%d/24' % (j - 48), defaultRoute='via 172.16.50.%d' % (j - 48))
        net.addLink(h, s1)
        ip = "172.16.50.%d" % (j-48)
        s1.cmdPrint("ifconfig s1-eth%d %s netmask 255.255.255.0" % (j,ip) )
```

Dividing and explaining this code, first, we add the VLAN1 for seven offices. The parameters inside add host function are explained below:

1. name is the name of the host. We define
 - a. first floor hosts as ‘hf%d’
 - b. second floor as ‘hs%d’
 - c. third floor as ‘ht%d’
 where %d will be replaced by the host numbers like 1,2,3 and so on.
2. cls = VLANHost which will call our initially defined VLAN class that configures the VLAN interfaces in the hosts.
3. VLAN parameter will receive the VLAN base number.
4. MAC addresses starting from 01:00:00:00:00:01 to 01:00:00:00:00:07
5. IP addresses starting from 172.16.1.2 to 172.16.1.8 with default routes from 172.16.10.2 to 172.16.10.8 respectively.

This means that the IP address on the interface of the switch s1 will be configured as, for example, as s1-eth1 = 172.16.10.2, s1-eth2 = 172.16.10.3 and so on.

We use the command

“ifconfig s1-eth1 172.16.10.2 netmask 255.255.255.0” to configure that particular interface with this IP address.

```
s1.cmdPrint("ifconfig s1-eth%d %s netmask 255.255.255.0" % (j,ip) )
```

The same method is extended to second and third floors as below.

```
info('***Adding hosts of second floor (hs) and linking them to Switch 2\n')
j=1
for j in range(n):
    if (j>0 and j <= 10):
        name = 'hs%d' % (j)
        h = net.addHost(name, cls=VLANHost, vlan=vlan5, mac="02:00:00:00:00:0%d" % (j), ip='172.16.4.%d/24' % (j + 1), defaultRoute='via 172.16.40.%d' % (j + 1))
        net.addLink(h, s2)
        ip = "172.16.40.%d" % (j+1)
        s2.cmdPrint("ifconfig s2-eth%d %s netmask 255.255.255.0" % (j,ip) )
    elif (j ==11):
        name = 'hs%d' % (j)
        h = net.addHost(name, cls=VLANHost, vlan=vlan6, mac="02:00:00:00:00:0%d" % (j), ip='172.16.5.%d/24' % (j - 9), defaultRoute='via 172.16.50.%d' % (j - 9))
        net.addLink(h, s2)
        ip = "172.16.50.%d" % (j-9)
        s2.cmdPrint("ifconfig s2-eth%d %s netmask 255.255.255.0" % (j,ip) )
info('***Adding hosts of third floor (ht) and linking them to switch 3\n')
j=1
for j in range(n):
    if (j>0 and j <= 30):
        name = 'ht%d' % (j)
        h = net.addHost(name, cls=VLANHost, vlan=vlan7, mac="03:00:00:00:00:0%d" % (j), ip='172.16.6.%d/24' % (j + 1), defaultRoute='via 172.16.60.%d' % (j + 1))
        net.addLink(h, s3)
        ip = "172.16.60.%d" % (j+1)
        s3.cmdPrint("ifconfig s3-eth%d %s netmask 255.255.255.0" % (j,ip) )
    elif (j >= 31 and j <= 35):
        name = 'ht%d' % (j)
        h = net.addHost(name, cls=VLANHost, vlan=vlan8, mac="03:00:00:00:00:0%d" % (j), ip='172.16.7.%d/24' % (j + 1), defaultRoute='via 172.16.70.%d' % (j + 1))
        net.addLink(h, s3)
        ip = "172.16.70.%d" % (j+1)
        s3.cmdPrint("ifconfig s3-eth%d %s netmask 255.255.255.0" % (j,ip) )
    elif (j == 36):
        name = 'ht%d' % (j)
        h = net.addHost(name, cls=VLANHost, vlan=vlan9, mac="03:00:00:00:00:0%d" % (j), ip='172.16.5.%d/24' % (j - 34), defaultRoute='via 172.16.50.%d' % (j - 34))
        net.addLink(h, s3)
        ip = "172.16.50.%d" % (j-34)
        s3.cmdPrint("ifconfig s3-eth%d %s netmask 255.255.255.0" % (j,ip) )
```

After the hosts are added for all three floors, we add the servers in the server room. We add ten common servers in VLANBASE 60 and five dedicated servers in VLANBASE 70. Servers are also hosts in mininet.

```
info('***Adding common, dedicated servers and linking them to switch 6\n')
j = 1
for j in range(n):
    if (j>0 and j <= 10):
        name = 'server%d' % (j)
        h = net.addHost(name, cls=VLANHost, vlan=vlan10, mac="11:00:00:00:00:0%d" % (j), ip='172.16.8.%d/24' % (j + 1), defaultRoute='via 172.16.80.%d' % (j + 1))
        net.addLink(h, s6)
        ip = "172.16.80.%d" % (j+1)
        s6.cmdPrint("ifconfig s6-eth%d %s netmask 255.255.255.0" % (j,ip) )
        starthttp(h)
    elif (j >= 11 and j <= 15):
        name = 'dserver%d' % (j)
        h = net.addHost(name, cls=VLANHost, vlan=vlan11, mac="11:00:00:00:00:0%d" % (j), ip='172.16.9.%d/24' % (j - 9), defaultRoute='via 172.16.90.%d' % (j - 9))
        net.addLink(h, s6)
        ip = "172.16.90.%d" % (j-9)
        s6.cmdPrint("ifconfig s6-eth%d %s netmask 255.255.255.0" % (j,ip) )
        starthttp(h)
```

2.2 Demonstration of running the server:

A simple HTTP server is used to demonstrate how the server works to simulate a real-time server environment. We use the `starthttp(host)` method to run a server

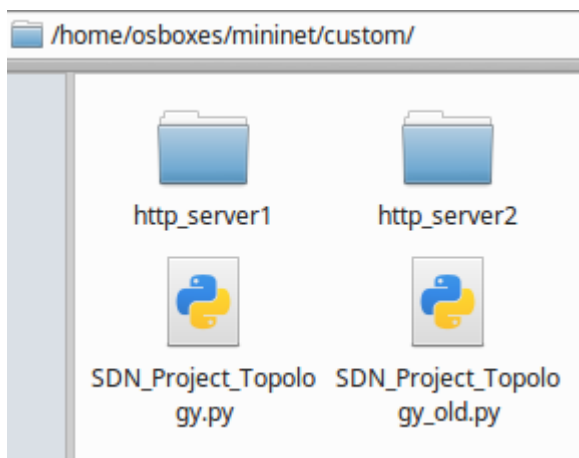
program from the host in mininet. Stophttp() method will kill all the running servers when mininet() is closed.

```
def starthttp( host ) :
    "Start simple Python web server on hosts"
    info( '*** Starting SimpleHTTPServer on host', host, '\n' )
    host.cmd( 'cd ./http_%s/; nohup python2.7 ./webserver.py &' % (host.name) )

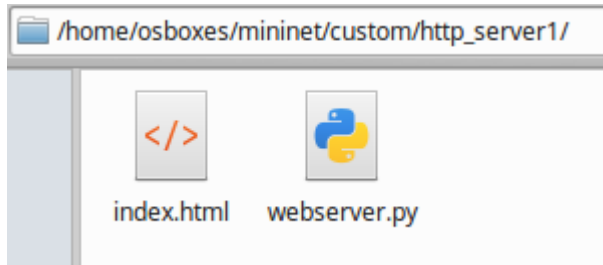
def stophttp() :
    "Stop simple Python web servers"
    info( '*** Shutting down stale SimpleHTTPServers',
        quietRun( "pkill -9 -f SimpleHTTPServer" ), '\n' )
    info( '*** Shutting down stale webservers',
        quietRun( "pkill -9 -f webserver.py" ), '\n' )
```

The starthttp(host) function will simply call the webserver.py file under http_<hostname> folder. For example, if we call starthttp(server1) then http_server1 folder will be accessed and the webserver.py file inside that folder will be run.

Obviously we have to create these folder structures inside mininet before using the starthttp() function in the program. The folder structure is as shown below. The folders are present in the same location (mininet/custom) as the SDN_topology.py file.



Inside the folder, we have an index.html file to show contents of the page and webserver.py file.



The index.html file just shows the content present inside when an HTTP GET request is made by the clients. The index.html code has the below code.

```
<html>
<head><title> This is Server1 </title></head>
<body>
Congratulations! <br/>
Your router successfully route your packets to server1. <br/>
</body>
</html>
```

The webserver.py file is an http server python program that waits for a client request. It is just a simple HTTP server that serves at port 80 as shown below.

```
import SimpleHTTPServer
import SocketServer

class MininetHttpRequestHandler(SimpleHTTPServer.SimpleHTTPRequestHandler):
    # Disable logging DNS lookups
    def address_string(self):
        return str(self.client_address[0])

PORT = 80

Handler = MininetHttpRequestHandler
httpd = SocketServer.TCPServer(("", PORT), Handler)
print "Server1: httpd serving at port", PORT
httpd.serve_forever()
```

The links between switches are added using the addLink method as shown below.


```
# Adding links between switches
info('***Adding links between switches\n')
net.addLink(s1, s4)
net.addLink(s1, s5)
net.addLink(s2, s4)
net.addLink(s2, s5)
net.addLink(s3, s4)
net.addLink(s4, s5)
net.addLink(s6, s5)
net.addLink(s6, s4)
net.addLink(s7, s4)
net.addLink(s7, s5)
```

The controller is started, and all the switches are started by connecting to the controller as shown below.

```
c1.start()
s1.start( [c1] )
s2.start( [c1] )
s3.start( [c1] )
s4.start( [c1] )
s5.start( [c1] )
s6.start( [c1] )
s7.start( [c1] )
```

The actual mininet is built when `net.build()` is called. The `CLI(net)` will show the log outputs on the terminal window. `Stophttp()` and `net.stop()` will be called when the mininet is stopped or interrupted to end the program.

```
info('***starting network\n')
net.build()
info('*****Starting switches\n')
info('*****post configure switches and hosts\n')
CLI(net)
stophttp()
net.stop()

if __name__ == '__main__':
    setLogLevel('info')

    if not quietRun('which vconfig'):
        error("Cannot find command 'vconfig'\nThe package",
              "'vlan' is required in Ubuntu or Debian,",
              "or 'vconfig' in Fedora\n")
        exit()
    else:
        VLAN_Mininet()
```

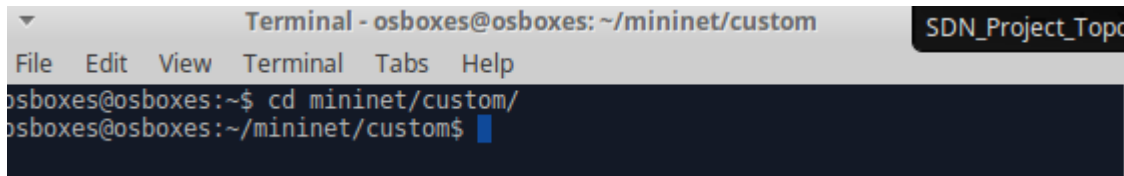
The small code bit at the end is the code used for unit testing of this topology.

4.3 Implementation

It can run as an independent python file inside the mininet/custom folder using :

sudo -E python SDN_Project_Topology.py

1. Navigate to mininet/custom folder

A screenshot of a terminal window titled "Terminal - osboxes@osboxes: ~/mininet/custom". The window has a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal shows the command "cd mininet/custom/" being executed, changing the prompt from "osboxes@osboxes:~\$" to "osboxes@osboxes:~/mininet/custom\$". A blue cursor is visible at the end of the prompt. A tab labeled "SDN_Project_Topo" is visible in the top right corner.

```
Terminal - osboxes@osboxes: ~/mininet/custom
File Edit View Terminal Tabs Help
osboxes@osboxes:~$ cd mininet/custom/
osboxes@osboxes:~/mininet/custom$
```

2. Run the topology using the above command

```
Terminal
File Edit View Terminal Tabs Help
osboxes@osboxes:~$ cd mininet/custom/
osboxes@osboxes:~/mininet/custom$ sudo -E python SDN_Project_Topology.py
[sudo] password for osboxes:
Unable to contact the remote controller at 127.0.0.1:6633
****add switches
***Adding hosts of first floor (hf) and linking them to Switch 1
*** s1 : ('ifconfig s1-eth1 172.16.10.2 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth2 172.16.10.3 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth3 172.16.10.4 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth4 172.16.10.5 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth5 172.16.10.6 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth6 172.16.10.7 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth7 172.16.10.8 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth8 172.16.20.2 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth9 172.16.20.3 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth10 172.16.20.4 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth11 172.16.20.5 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth12 172.16.20.6 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth13 172.16.20.7 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth14 172.16.20.8 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth15 172.16.20.9 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth16 172.16.20.10 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth17 172.16.20.11 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth18 172.16.20.12 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth19 172.16.20.13 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth20 172.16.20.14 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth21 172.16.20.15 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth22 172.16.20.16 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth23 172.16.20.17 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth24 172.16.20.18 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth25 172.16.20.19 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth26 172.16.20.20 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth27 172.16.20.21 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth28 172.16.20.22 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth29 172.16.20.23 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth30 172.16.20.24 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth31 172.16.20.25 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth32 172.16.20.26 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth33 172.16.20.27 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth34 172.16.20.28 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth35 172.16.20.29 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth36 172.16.20.30 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth37 172.16.20.31 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth38 172.16.20.32 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth39 172.16.20.33 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth40 172.16.20.34 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth41 172.16.20.35 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth42 172.16.20.36 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth43 172.16.20.37 netmask 255.255.255.0',)
```

```

*** s1 : ('ifconfig s1-eth43 172.16.20.37 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth44 172.16.20.38 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth45 172.16.20.39 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth46 172.16.20.40 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth47 172.16.20.41 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth48 172.16.20.42 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth49 172.16.30.2 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth50 172.16.50.2 netmask 255.255.255.0',)
***Adding hosts of second floor (hs) and linking them to Switch 2
*** s2 : ('ifconfig s2-eth1 172.16.40.2 netmask 255.255.255.0',)
*** s2 : ('ifconfig s2-eth2 172.16.40.3 netmask 255.255.255.0',)
*** s2 : ('ifconfig s2-eth3 172.16.40.4 netmask 255.255.255.0',)
*** s2 : ('ifconfig s2-eth4 172.16.40.5 netmask 255.255.255.0',)
*** s2 : ('ifconfig s2-eth5 172.16.40.6 netmask 255.255.255.0',)
*** s2 : ('ifconfig s2-eth6 172.16.40.7 netmask 255.255.255.0',)
*** s2 : ('ifconfig s2-eth7 172.16.40.8 netmask 255.255.255.0',)
*** s2 : ('ifconfig s2-eth8 172.16.40.9 netmask 255.255.255.0',)
*** s2 : ('ifconfig s2-eth9 172.16.40.10 netmask 255.255.255.0',)
*** s2 : ('ifconfig s2-eth10 172.16.40.11 netmask 255.255.255.0',)
*** s2 : ('ifconfig s2-eth11 172.16.50.2 netmask 255.255.255.0',)
***Adding hosts of third floor (ht) and linking them to switch 3
*** s3 : ('ifconfig s3-eth1 172.16.60.2 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth2 172.16.60.3 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth3 172.16.60.4 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth4 172.16.60.5 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth5 172.16.60.6 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth6 172.16.60.7 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth7 172.16.60.8 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth8 172.16.60.9 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth9 172.16.60.10 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth10 172.16.60.11 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth11 172.16.60.12 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth12 172.16.60.13 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth13 172.16.60.14 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth14 172.16.60.15 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth15 172.16.60.16 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth16 172.16.60.17 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth17 172.16.60.18 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth18 172.16.60.19 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth19 172.16.60.20 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth20 172.16.60.21 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth21 172.16.60.22 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth22 172.16.60.23 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth23 172.16.60.24 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth24 172.16.60.25 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth25 172.16.60.26 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth26 172.16.60.27 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth27 172.16.60.28 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth28 172.16.60.29 netmask 255.255.255.0',)

```

```

*** s3 : ('ifconfig s3-eth27 172.16.60.28 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth28 172.16.60.29 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth29 172.16.60.30 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth30 172.16.60.31 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth31 172.16.70.32 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth32 172.16.70.33 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth33 172.16.70.34 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth34 172.16.70.35 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth35 172.16.70.36 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth36 172.16.50.2 netmask 255.255.255.0',)
***Adding common, dedicated servers and linking them to switch 6
*** s6 : ('ifconfig s6-eth1 172.16.80.2 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth2 172.16.80.3 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth3 172.16.80.4 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth4 172.16.80.5 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth5 172.16.80.6 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth6 172.16.80.7 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth7 172.16.80.8 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth8 172.16.80.9 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth9 172.16.80.10 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth10 172.16.80.11 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth11 172.16.50.2 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth12 172.16.50.3 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth13 172.16.50.4 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth14 172.16.50.5 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth15 172.16.50.6 netmask 255.255.255.0',)
***Adding links between switches
***starting network
*** Configuring hosts
hf1 hf2 hf3 hf4 hf5 hf6 hf7 hf8 hf9 hf10 hf11 hf12 hf13 hf14 hf15 hf16 hf17 hf18 hf19 hf20 hf21 hf22 hf23 hf24 hf25 hf26 hf27 hf28 hf29 hf30 hf31 hf32 hf33 hf34 hf35 hf36 hf37 hf38
hf39 hf40 hf41 hf42 hf43 hf44 hf45 hf46 hf47 hf48 hf49 hf50 hs1 hs2 hs3 hs4 hs5 hs6 hs7 hs8 hs9 hs10 hs11 ht1 ht2 ht3 ht4 ht5 ht6 ht7 ht8 ht9 ht10 ht11 ht12 ht13 ht14 ht15 ht16 ht
17 ht18 ht19 ht20 ht21 ht22 ht23 ht24 ht25 ht26 ht27 ht28 ht29 ht30 ht31 ht32 ht33 ht34 ht35 ht36 server1 server2 server3 server4 server5 server6 server7 server8 server9 server10 d
server11 dserver12 dserver13 dserver14 dserver15
****Starting switches
****post configure switches and hosts
*** Starting CLI:
mininet>

```

As we can observe in the above screenshots the topology in the mininet gets created.

- The first floor hosts are hf1 to hf50
- The second-floor hosts are hs1 to hs11
- The third-floor hosts are ht1 to ht36
- The common servers are server1 to server10
- The dedicated servers are dserver1 to dserver5

Their corresponding IP addresses are allocated as per the topology designed with VLANs.

4.4 Providing internet connection to the hosts:

The following commands is used to provide internet connectivity to the hosts.

```

ifconfig h1-eth0 inet 0.0.0.0
dhclient h1-eth0

```

When we ping google server it successfully pings.

```
*** h1 : ('dhclient h1-eth0',)
*** Starting CLI:
mininet> h1 ping -c 1 www.google.com
PING www.google.com (74.125.226.112) 56(84) bytes of data.
64 bytes from yyz08s13-in-f16.1e100.net (74.125.226.112): icmp_req=1 ttl=55 time=11.6 ms

--- www.google.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 11.692/11.692/11.692/0.000 ms
```

When the free ethernet interface is configured to dhclient the internet connectivity is established in individual hosts. However, in our topology the switch is connecting to the internet hosts.

One way is, we will add one interface on the switch 7 and enable NAT functionality using the below commands.

```
net.addNAT().configDefault()
```

Other way is we can add host with internet to switch 7 and make it as a internet host.

PART 3: NETWORK CONFIGURATION AND APPLICATION DEVELOPMENT

There are two ways in which connectivity can be established in this network we have designed.

5.1. Switching app for the trunk and access VLAN switches:

The entire application has been written in VlanSwitchSDN.py file. The entire code is explained at the end. Usually, the switches are configured manually on their ports using the below commands in VLAN configurations.

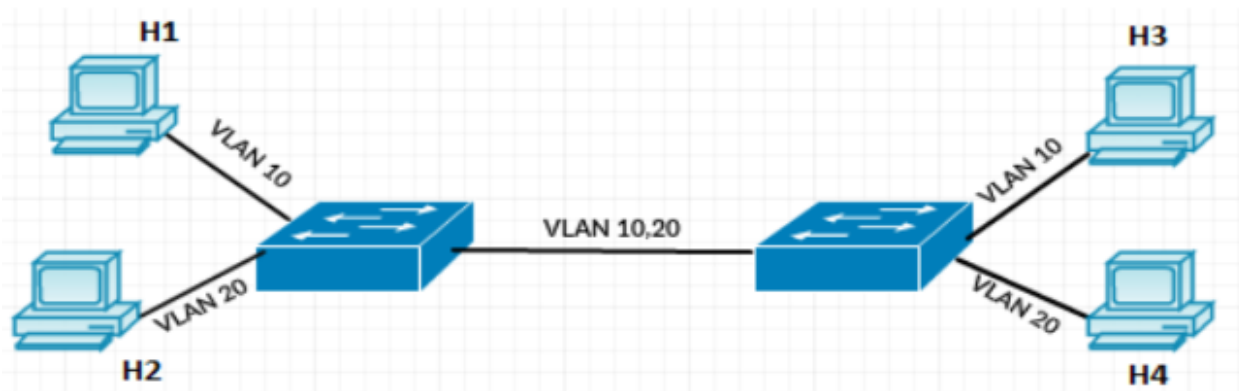
For access ports,

sudo ovs-vsctl set port [port] tag = <vlanbase>

For trunking ports,

sudo ovs-vsctl set port [ports] tag = vlanbase1,vlanbase2

For example, if we consider the below simple topology,



The left switch is S1 and right switch is S2

S1.cmd("sudo ovs-vsctl set port [s1-eth1] tag = 10)

Similarly we can configure other switches manually.

In our design and implementation of mininet, we already tagged the VLAN on the interface name from python itself.

The purpose of using ryu controller is to avoid manual configuration all these interfaces of the switches. We developed a ryu app that updates the incoming and outgoing interfaces of the switch automatically in a flow table. The concept of the app is presented below.

We are considering two scenarios only for simplicity, and our entire network is all VLAN network. This can then be extended to our entire topology.

Scenario 1 – Each switch has only one trunk line:

In this case, our assumption is all the packets will be VLAN tagged when they arrive and leave the trunk switches given that all hosts are VLAN based.

Scenario 2 – Each switch can have multiple trunk lines:

In this case, our assumption is the packets are tagged at trunk when they come or leave but untagging will not be done to all the packets that are incoming.

In both these cases, this app works perfectly to perform switching.

The basic flow actions that need to be performed are:

1. Add VLAN tag to the incoming packets/frames if not tagged already by matching the 'in-port', 'tag-status' and send it in the output trunk port.
2. Tagged packets destined for hosts connected to another switch must be maintained in the tagged state and relayed.
3. Tagged packets destined for hosts connected to the same switch must be matched with destination MAC address and VLAN ID, untagged and send it output port.

Let us get into the code and explain the functionalities:

We basically use four RYU OFP v1.3 flow actions namely,

- `OFPActionSetField(vlanid="vid")` – Puts TCI field's vid value with value given
- `OFPMatch(vlanid=0x100|"vid")` – Matches VLANID. 'vid' is an integer value
- `OFPActionPushVlan(ethertype=33024,type=None,len=None)` – Pushes a VLAN tag
- `OFPActionPopVlan(type=None,len=None)` – Pops a VLAN tag from the outer part.

The three main dictionaries in the mac to port table are as shown below:

```
VLAN ONLY WITH SINGLE TRUNK LINE

port_vlan = {1:{1:[10],2:[20],3:[10,20]},2:{1:[10],2:[20],3:[10,20]}} #port_vlan[a][b]=c => 'a'= dpid, 'b'= port number,'c'= VLAN ID

access= {1:[1,2],2:[1,2]} #access[a]=[B] => 'a' = dpid ,'[B]'=List of ports configured as Access Ports

trunk = {1:[3],2:[3]} #trunk[a]=[B] => 'a' = dpid ,'[B]'=List of ports configured as Trunk Ports

-----
VLAN ONLY NETWORK WITH MULTI TRUNK LINES

port_vlan = {1:{1:[10],2:[20],3:[30],4:[10,20,30]},2:{1:[10],2:[40],3:[10,20,30],4:[20,40],5:[10,30]},3:{1:[20],2:[40],3:[40],4:[20,40]}, 4:{1:[30],2:[10],3:[10,30]} }

access= {1:[1,2,3],2:[1,2],3:[1,2,3],4:[1,2]} #access[a]=[B] => 'a' = dpid ,'[B]'=List of ports configured as Access Ports

trunk = {1:[4],2:[3,4,5],3:[4],4:[3]} #trunk[a]=[B] => 'a' = dpid ,'[B]'=List of ports configured as Trunk Ports
```

- port_vlan consists of the mapping between portID, port number, and VLAN ID
- Access consists of the datapath id(dpid) and access ports configured in the network.
- Trunk consists of the datapath id and trunk ports configured in the network.

Let us explain the code of this VLAN switching app,

Initially we import all the libraries required as shown below.

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import vlan
from ryu.lib.packet import ether_types
```

- As we already know, ryu.base.app_manager loads all the ryu applications.
- Open flow event definitions are contained in ofp_event
- Ofproto_v1_3 helps choose the OFP version

- In this code, we are handling ethernet, VLAN, packet parsers and ethernet type predefined libraries because we are only dealing with ethernet, vlan, and packets in our open flow switch.

We define our global variable dictionaries, which are basically the tables that we will update when the frames are exchanged.

```
#GLOBAL VARIABLES
port_vlan = {2:{3:[" "],1:[30],2:[20],4:[20,30]},3:{4:[20,30],1:[30],2:[20],3:[" "]}} #port_vlan[a][b]=c => 'a' = dpid, 'b' = port number, 'c' = VLAN ID
access = {2:[1,2],3:[1,2]} #access[a]=[B] => 'a' = dpid, '[B]' = List of ports configured as Access Ports
trunk = {2:[4],3:[4]} #trunk[a]=[B] => 'a' = dpid, '[B]' = List of ports configured as Trunk Ports
```

We already explained these three dictionaries before. The VLAN switch class is defined, and it is the child of Ryu App, and the open flow protocol version is set to 1.3. We further define a constructor to instantiate the app.

```
class VlanSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(VlanSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
```

First, let us install the table, miss flow entry. Using the decorator @set_ev_cls, we define the switch features handler function, which basically sets a flow in the switch that all packets are forwarded to the controller since we don't know what to do initially with the packets with no flow installed on the switches. Ofproto.OFPP_CONTROLLER signifies that. The add_flow method helps in adding flow to the OpenFlow switch. Note that the priority is set to 0 because this may be the last applicable flow action when no other entry is present in the flow table.

```

@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    match = parser.OFPMatch()
    actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                     ofproto.OFPCML_NO_BUFFER)]
    self.add_flow(datapath, 0, match, actions)

```

We define the `add_flow()` method below. This method will add the datapath, priority, match, actions, and buffer id according to the parameter values that are passed. The flow mod message is created and then sent in the datapath to configure the switch.

```

def add_flow(self, datapath, priority, match, actions, buffer_id=None):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                         actions)]
    if buffer_id:
        mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                priority=priority, match=match,
                                instructions=inst)
    else:
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                match=match, instructions=inst)
    datapath.send_msg(mod)

```

We then define the `vlan_members` function. This will define which ports are trunk ports and which ports are access ports in the switch in two different lists.

```

def vlan_members(self,dpid,in_port,src_vlan):

    B=[]
    self.access_ports = []
    self.trunk_ports = []

    if src_vlan == "NULL":
        return

    for item in port_vlan[dpid]:
        vlans=port_vlan[dpid][item]
        if src_vlan in vlans and item!=in_port:
            B.append(item)

    for port in B:
        if port in access[dpid]:
            self.access_ports.append(port)
        else:
            self.trunk_ports.append(port)

```

The `getActionsArrayTrunk` method is used to create an actions array for all the packets received in trunk ports. This method determines which output ports the VLAN ID must be popped or not. (That depends on the trunk or access ports as already explained before)

```

def getActionsArrayTrunk(self,out_port_access,out_port_trunk,parser):
    actions= [ ]

    for port in out_port_trunk:
        actions.append(parser.OFPActionOutput(port))

    actions.append(parser.OFPActionPopVlan())

    for port in out_port_access:
        actions.append(parser.OFPActionOutput(port))

    return actions

```

We then define the array of access ports and the corresponding actions to be done when the packet is received in the access ports. 33024 is the ethernet type 802.1Q used here.

```
def getActionsArrayAccess(self,out_port_access,out_port_trunk,src_vlan, parser):
    actions= [ ]

    for port in out_port_access:
        actions.append(parser.OFPActionOutput(port))

    actions.append(parser.OFPActionPushVlan(33024))
    actions.append(parser.OFPActionSetField(vlan_vid=src_vlan))

    for port in out_port_trunk:
        actions.append(parser.OFPActionOutput(port))

    return actions
```

Then we go ahead and define the packet handler class with decorator @set_ev_cls tied to the main dispatcher. This class is where the incoming packets will be handled. The message length is statically defined, as shown below. It can be increased when the limit is reached.

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    # If you hit this you might want to increase
    # the "miss_send_length" of your switch
    if ev.msg.msg_len < ev.msg.total_len:
        self.logger.debug("packet truncated: only %s of %s bytes",
                           ev.msg.msg_len, ev.msg.total_len)
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']
```

The variables for dpid, L3 pkt, L2 data frames, and vlan_header are defined as shown below.

```

#SWITCH ID
dpid = datapath.id

pkt = packet.Packet(msg.data)
eth = pkt.get_protocols(ethernet.ethernet)[0]
vlan_header = pkt.get_protocols(vlan.vlan)  #If packet is tagged,then this will have non-null value

```

We next define the header handling for the 802.1Q VLAN tagged and untagged frame.

```

if eth.ethertype == ether_types.ETH_TYPE_8021Q :    #Checking for VLAN Tagged Packet
    vlan_header_present = 1
    src_vlan=vlan_header[0].vid
elif dpid not in port_vlan:
    vlan_header_present = 0
    in_port_type = "NORMAL SWITCH "                #NORMAL NON-VLAN L2 SWITCH
    src_vlan = "NULL"
elif port_vlan[dpid][in_port][0]== " " or in_port in trunk[dpid]:
    vlan_header_present = 0
    in_port_type = "NORMAL UNTAGGED"                #NATIVE VLAN PACKET
    src_vlan = "NULL"
else:
    vlan_header_present = 0
    src_vlan=port_vlan[dpid][in_port][0]           # STORE VLAN ASSOCIATION FOR THE IN PORT

```

The LLDP packets are forwarded as shown below

```

if eth.ethertype == ether_types.ETH_TYPE_LLDP:
    # ignore lldp packet
    return

dst = eth.dst
src = eth.src

```

The filling of the MAC to Port table is shown in the below lines of code. MAC address is learned to avoid flooding next time. In_port's and out_port's VLAN members are determined. The table consists of the MACaddress, in_port, out_port and says if the port is trunk port or access port.

```

#CREATE NEW DICTIONARY ENTRY IF IT DOES NOT EXIST
self.mac_to_port.setdefault(dpid, {})

self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

# learn a mac address to avoid FLOOD next time.
self.mac_to_port[dpid][src] = in_port

#Determine which ports are members of in_port's VLAN
self.vlan_members(dpid,in_port,src_vlan)
out_port_type = " "

if dst in self.mac_to_port[dpid]:          #MAC ADDRESS TABLE CREATION
    out_port_unknown = 0
    out_port = self.mac_to_port[dpid][dst]
    if src_vlan!="NULL":
        if out_port in access[dpid]:
            out_port_type = "ACCESS"
        else:
            out_port_type = "TRUNK"
    else :
        out_port_type = "NORMAL"
else:
    out_port_unknown = 1
    out_port_access = self.access_ports      #List of All Access Ports Which are Members of Same VLAN (to Flood the Traffic)
    out_port_trunk = self.trunk_ports        #List of All Trunk Ports Which are Members of Same VLAN (to Flood the Traffic)

```

The flow entries are added in this part of the code. The actions that we discussed, in the beginning, is done in this. The Popping and pushing of VLAN tags if they tagged and directly forwarding if they untagged is all taken care in this segment.

```

if out_port_unknown!=1:                    # IF OUT PORT IS KNOWN
    if vlan_header_present and out_port_type == "ACCESS" :          #If VLAN Tagged and needs to be sent out through ACCESS port
        match = parser.OFPMatch(in_port=in_port, eth_dst=dst, vlan_vid=(0x1000 | src_vlan))
        actions = [parser.OFPACTIONPopVlan(), parser.OFPACTIONOutput(out_port)] # STRIP VLAN TAG and SEND TO OUTPUT PORT
    elif vlan_header_present and out_port_type == "TRUNK" :
        match = parser.OFPMatch(in_port=in_port, eth_dst=dst, vlan_vid=(0x1000 | src_vlan))
        actions = [parser.OFPACTIONOutput(out_port)]                #SEND THROUGH TRUNK PORT AS IS
    elif vlan_header_present!=1 and out_port_type == "TRUNK" :
        match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
        actions = [parser.OFPACTIONPushVlan(33024), parser.OFPACTIONSetField(vlan_vid=src_vlan), parser.OFPACTIONOutput(out_port)]
    else:
        match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
        actions = [parser.OFPACTIONOutput(out_port)]

# verify if we have a valid buffer_id, if avoid yes to send both
# flow_mod & packet_out
if msg.buffer_id != ofproto.OFP_NO_BUFFER:
    self.add_flow(datapath, 1, match, actions, msg.buffer_id)
    return
else:
    self.add_flow(datapath, 1, match, actions)

else :                                     #FOR FLOODING ACTIONS
    if vlan_header_present:                 #IF TAGGED
        actions = self.getActionsArrayTrunk(out_port_access,out_port_trunk,parser)
    elif vlan_header_present==0 and src_vlan!="NULL":          #IF UNTAGGED BUT GENERATED FROM VLAN ASSOCIATED PORT
        actions = self.getActionsArrayAccess(out_port_access,out_port_trunk,src_vlan,parser)
    elif in_port_type == "NORMAL UNTAGGED":          #IF UNTAGGED AND BELONGING TO NATIVE VLAN (CAPTURED ON A VLAN AWARE SWITCH)
        actions = self.getActionsNormalUntagged(dpid,in_port,parser)
    else :
        actions = [parser.OFPACTIONOutput(ofproto.OFPP_FLOOD)]          # FOR NORMAL NON-VLAN L2 SWITCH

```

The actions of the untagged and tagged packets are resolved by calling their appropriate functions, as shown below. The data path is set using the last line in the code.

```
# verify if we have a valid buffer_id, if avoid yes to send both
# flow_mod & packet_out
if msg.buffer_id != ofproto.OFP_NO_BUFFER:
    self.add_flow(datapath, 1, match, actions, msg.buffer_id)
    return
else:
    self.add_flow(datapath, 1, match, actions)

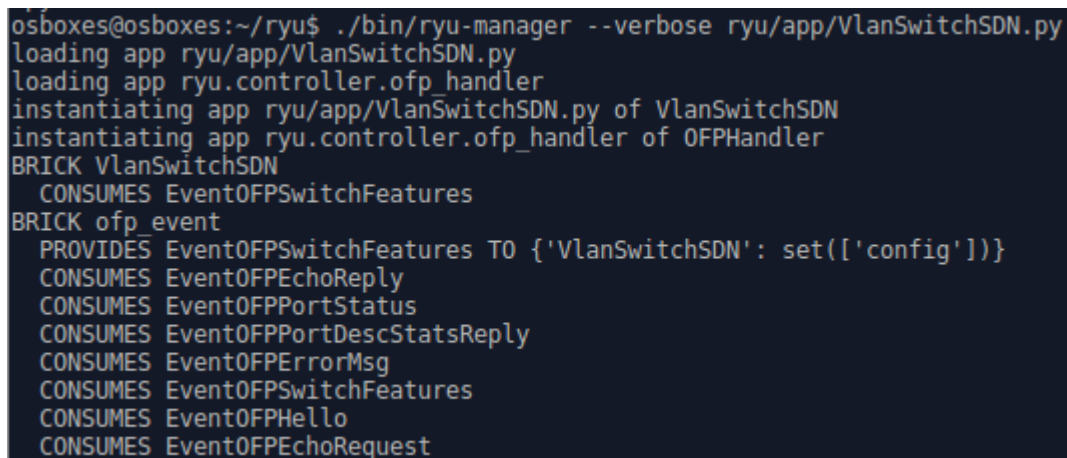
else :
    #FOR FLOODING ACTIONS
    if vlan_header_present:
        #IF TAGGED
        actions = self.getActionsArrayTrunk(out_port_access,out_port_trunk,parser)
    elif vlan_header_present==0 and src_vlan!= "NULL":
        #IF UNTAGGED BUT GENERATED FROM VLAN ASSOCIATED PORT
        actions = self.getActionsArrayAccess(out_port_access,out_port_trunk,src_vlan, parser)
    elif in_port_type == "NORMAL UNTAGGED":
        #IF UNTAGGED AND BELONGING TO NATIVE VLAN (CAPTURED ON A VLAN AWARE SWITCH)
        actions = self.getActionsNormalUntagged(dpid,in_port,parser)
    else :
        actions = [parser.OFPActionOutput(ofproto.OFPP_FLOOD)]
        # FOR NORMAL NON-VLAN L2 SWITCH

data = None
if msg.buffer_id == ofproto.OFP_NO_BUFFER:
    data = msg.data

out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                          in_port=in_port, actions=actions, data=data)
datapath.send_msg(out)
```

Implementation of the app in VM:

We first run the Ryu application as shown below



```
osboxes@osboxes: ~/.ryu$ ./bin/ryu-manager --verbose ryu/app/VlanSwitchSDN.py
loading app ryu/app/VlanSwitchSDN.py
loading app ryu.controller.ofp_handler
instantiating app ryu/app/VlanSwitchSDN.py of VlanSwitchSDN
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK VlanSwitchSDN
  CONSUMES EventOFPSwitchFeatures
BRICK ofp_event
  PROVIDES EventOFPSwitchFeatures TO {'VlanSwitchSDN': set(['config'])}
  CONSUMES EventOFPEchoReply
  CONSUMES EventOFPPortStatus
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPErrormsg
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPHello
  CONSUMES EventOFPEchoRequest
```

Then run the mininet topology,


```

osboxes@osboxes:~/mininet/custom$ sudo python -E SDN_Project_Topology.py
[sudo] password for osboxes:
****add switches
****Adding hosts of first floor (hf) and linking them to Switch 1
*** s1 : ('ifconfig s1-eth1 172.16.10.2 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth2 172.16.10.3 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth3 172.16.10.4 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth4 172.16.10.5 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth5 172.16.10.6 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth6 172.16.10.7 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth7 172.16.10.8 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth8 172.16.20.2 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth9 172.16.20.3 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth10 172.16.20.4 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth11 172.16.20.5 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth12 172.16.20.6 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth13 172.16.20.7 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth14 172.16.20.8 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth15 172.16.20.9 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth16 172.16.20.10 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth17 172.16.20.11 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth18 172.16.20.12 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth19 172.16.20.13 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth20 172.16.20.14 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth21 172.16.20.15 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth22 172.16.20.16 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth23 172.16.20.17 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth24 172.16.20.18 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth25 172.16.20.19 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth26 172.16.20.20 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth27 172.16.20.21 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth28 172.16.20.22 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth29 172.16.20.23 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth30 172.16.20.24 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth31 172.16.20.25 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth32 172.16.20.26 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth33 172.16.20.27 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth34 172.16.20.28 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth35 172.16.20.29 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth36 172.16.20.30 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth37 172.16.20.31 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth38 172.16.20.32 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth39 172.16.20.33 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth40 172.16.20.34 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth41 172.16.20.35 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth42 172.16.20.36 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth43 172.16.20.37 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth44 172.16.20.38 netmask 255.255.255.0')
*** s1 : ('ifconfig s1-eth45 172.16.20.39 netmask 255.255.255.0')

```

```

*** s1 : ('ifconfig s1-eth46 172.16.20.40 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth47 172.16.20.41 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth48 172.16.20.42 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth49 172.16.30.2 netmask 255.255.255.0',)
*** s1 : ('ifconfig s1-eth50 172.16.50.2 netmask 255.255.255.0',)
***Adding hosts of second floor (hs) and linking them to Switch 2
*** s2 : ('ifconfig s2-eth1 172.16.40.2 netmask 255.255.255.0',)
*** s2 : ('ifconfig s2-eth2 172.16.40.3 netmask 255.255.255.0',)
*** s2 : ('ifconfig s2-eth3 172.16.40.4 netmask 255.255.255.0',)
*** s2 : ('ifconfig s2-eth4 172.16.40.5 netmask 255.255.255.0',)
*** s2 : ('ifconfig s2-eth5 172.16.40.6 netmask 255.255.255.0',)
*** s2 : ('ifconfig s2-eth6 172.16.40.7 netmask 255.255.255.0',)
*** s2 : ('ifconfig s2-eth7 172.16.40.8 netmask 255.255.255.0',)
*** s2 : ('ifconfig s2-eth8 172.16.40.9 netmask 255.255.255.0',)
*** s2 : ('ifconfig s2-eth9 172.16.40.10 netmask 255.255.255.0',)
*** s2 : ('ifconfig s2-eth10 172.16.40.11 netmask 255.255.255.0',)
*** s2 : ('ifconfig s2-eth11 172.16.50.2 netmask 255.255.255.0',)
***Adding hosts of third floor (ht) and linking them to switch 3
*** s3 : ('ifconfig s3-eth1 172.16.60.2 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth2 172.16.60.3 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth3 172.16.60.4 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth4 172.16.60.5 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth5 172.16.60.6 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth6 172.16.60.7 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth7 172.16.60.8 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth8 172.16.60.9 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth9 172.16.60.10 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth10 172.16.60.11 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth11 172.16.60.12 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth12 172.16.60.13 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth13 172.16.60.14 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth14 172.16.60.15 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth15 172.16.60.16 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth16 172.16.60.17 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth17 172.16.60.18 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth18 172.16.60.19 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth19 172.16.60.20 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth20 172.16.60.21 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth21 172.16.60.22 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth22 172.16.60.23 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth23 172.16.60.24 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth24 172.16.60.25 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth25 172.16.60.26 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth26 172.16.60.27 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth27 172.16.60.28 netmask 255.255.255.0',)

```

```

*** s3 : ('ifconfig s3-eth27 172.16.60.28 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth28 172.16.60.29 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth29 172.16.60.30 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth30 172.16.60.31 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth31 172.16.70.32 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth32 172.16.70.33 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth33 172.16.70.34 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth34 172.16.70.35 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth35 172.16.70.36 netmask 255.255.255.0',)
*** s3 : ('ifconfig s3-eth36 172.16.50.2 netmask 255.255.255.0',)
***Adding common, dedicated servers and linking them to switch 6
*** s6 : ('ifconfig s6-eth1 172.16.80.2 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth2 172.16.80.3 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth3 172.16.80.4 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth4 172.16.80.5 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth5 172.16.80.6 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth6 172.16.80.7 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth7 172.16.80.8 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth8 172.16.80.9 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth9 172.16.80.10 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth10 172.16.80.11 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth11 172.16.50.2 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth12 172.16.50.3 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth13 172.16.50.4 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth14 172.16.50.5 netmask 255.255.255.0',)
*** s6 : ('ifconfig s6-eth15 172.16.50.6 netmask 255.255.255.0',)
***Adding links between switches
***starting network
*** Configuring hosts
hf1 hf2 hf3 hf4 hf5 hf6 hf7 hf8 hf9 hf10 hf11 hf12 hf13 hf14 hf15 hf16 hf17 hf18
hf19 hf20 hf21 hf22 hf23 hf24 hf25 hf26 hf27 hf28 hf29 hf30 hf31 hf32 hf33 hf34
hf35 hf36 hf37 hf38 hf39 hf40 hf41 hf42 hf43 hf44 hf45 hf46 hf47 hf48 hf49 hf50
hs1 hs2 hs3 hs4 hs5 hs6 hs7 hs8 hs9 hs10 hs11 ht1 ht2 ht3 ht4 ht5 ht6 ht7 ht8 h
t9 ht10 ht11 ht12 ht13 ht14 ht15 ht16 ht17 ht18 ht19 ht20 ht21 ht22 ht23 ht24 ht
25 ht26 ht27 ht28 ht29 ht30 ht31 ht32 ht33 ht34 ht35 ht36 server1 server2 server
3 server4 server5 server6 server7 server8 server9 server10 dserver11 dserver12 d
server13 dserver14 dserver15
****Starting switches
****post configure switches and hosts
*** Starting CLI:

```

The switch gets connected to the ryu controller app as shown below, and the data flows are configured.

```

CONSUMES EventOfPEchoRequest
connected socket:<eventlet.greenio.base.GreenSocket object at 0xb630b20c> address:('127.0.0.1', 49510)
connected socket:<eventlet.greenio.base.GreenSocket object at 0xb630be6c> address:('127.0.0.1', 49514)

```

When we ping all, if the VLANs were configured correctly, the hosts will be able to ping only within VLANs directly as shown below.

```
mininet> hs1 ping hs2
PING 172.16.4.3 (172.16.4.3) 56(84) bytes of data.
64 bytes from 172.16.4.3: icmp_seq=1 ttl=64 time=0.301 ms
64 bytes from 172.16.4.3: icmp_seq=2 ttl=64 time=0.030 ms
64 bytes from 172.16.4.3: icmp_seq=3 ttl=64 time=0.029 ms
64 bytes from 172.16.4.3: icmp_seq=4 ttl=64 time=0.044 ms
64 bytes from 172.16.4.3: icmp_seq=5 ttl=64 time=0.036 ms
^C
--- 172.16.4.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4101ms
rtt min/avg/max/mdev = 0.029/0.088/0.301/0.106 ms
mininet> hf1 ping hf2
PING 172.16.1.3 (172.16.1.3) 56(84) bytes of data.
64 bytes from 172.16.1.3: icmp_seq=1 ttl=64 time=0.224 ms
64 bytes from 172.16.1.3: icmp_seq=2 ttl=64 time=0.034 ms
64 bytes from 172.16.1.3: icmp_seq=3 ttl=64 time=0.100 ms
64 bytes from 172.16.1.3: icmp_seq=4 ttl=64 time=0.029 ms
64 bytes from 172.16.1.3: icmp_seq=5 ttl=64 time=0.108 ms
64 bytes from 172.16.1.3: icmp_seq=6 ttl=64 time=0.032 ms
^C
--- 172.16.1.3 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5100ms
rtt min/avg/max/mdev = 0.029/0.087/0.224/0.070 ms
mininet> ht1 ping ht2
PING 172.16.6.3 (172.16.6.3) 56(84) bytes of data.
64 bytes from 172.16.6.3: icmp_seq=1 ttl=64 time=0.278 ms
64 bytes from 172.16.6.3: icmp_seq=2 ttl=64 time=0.037 ms
64 bytes from 172.16.6.3: icmp_seq=3 ttl=64 time=0.034 ms
64 bytes from 172.16.6.3: icmp_seq=4 ttl=64 time=0.039 ms
^C
--- 172.16.6.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3055ms
rtt min/avg/max/mdev = 0.034/0.097/0.278/0.104 ms
```

5.2. Routing and switching application :

We utilize Ryu based ARP and IP handling app to control switching and routing in the OpenFlow switch we created. In our design, we have a total of seven switches. Our file in the Ryu/app folder is L2L3switch.py. We explain the code in the routing and switching app below. The initial libraries are defined. The purpose of the libraries has been explained already.

```

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet, arp, ipv4
from ryu.lib.packet import ether_types
import ryu.lib.dpid
import ryu.lib

```

The L2L3switch class has been created as shown below with the initial class definition, switch feature handlers, and packet in handlers functions. The switch feature handlers handle all the L2 forwarding functionalities and Packet in handler will take care of the L3 forwarding functionalities.

```

class L2L3Switch(app_manager.RyuApp): #creating a simple switch as a Ryu App
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(L2L3Switch, self).__init__(*args, **kwargs)
        # simpleL2switch is a child of Ryu App.
        self.mac_to_port = {}
        # {port1:[mac1,ip1],port2:[mac2,ip2],...}
        # #to store the details of switch and connected hosts IP and mac address

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, event): #this method handles switch feature requests and we install initial flow to forward all the packets to the
        print("*** in feature handler *** ")
        # send to the controller
        sendto_controller(self, event)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, event):
        pkt = packet.Packet(data=event.msg.data)
        # creating a packet with msg's data as payload
        eth = pkt.get_protocols(ethernet.ethernet)[0]
        # fetching ethernet dataframe
        if eth.ethertype == ether_types.ETH_TYPE_ARP:
            #handling ARP requests
            handle_ARP(self, event)
            # ARP packet is ethernet dataframe's payload. Ethernet frame has type from as ETH_TYPE_ARP, if true then handle ARP
            # call method to handle ARP packets
        elif eth.ethertype == ether_types.ETH_TYPE_IP:
            handle_IP(self, event)
            #handle IP packet.
            #call method to handle IP packet

```

The Handle_ARP() method will handle the ARP packets. The switch will receive the ARP request via the in_port and checks for matching entries in the mac to the port table with source MAC, destination MAC, and in_port. If the entry is not already matched in the flow table called table miss, the controller instructs the switch to flood the frame to all its output ports.

```

def handle_ARP(self,event):
    # handle ARP packets
    datapath = event.msg.datapath
    # datapath connection
    ofproto = datapath.ofproto
    # ofproto of the datapath
    in_port = event.msg.match['in_port']
    # port through which the switch recieved this packet
    parser = datapath.ofproto_parser
    pkt = packet.Packet(data=event.msg.data)
    eth = pkt.get_protocols(ethernet.ethernet)[0]
    # fetching ethernet dataframe
    arp_pkt = pkt.get_protocol(arp.arp)
    # Fetching ARP dataframe
    self.mac_to_port[in_port] = [arp_pkt.src_mac,arp_pkt.src_ip]
    # MAC, IP storing
    out_port = self.check_mactable(ofproto,'arp',arp_pkt.dst_mac)
    # Check MAC table for destination IP
    actions = [parser.OFPACTIONOutput(out_port)]
    # updating action lists
    match = self.simplematch(parser,eth.src,eth.dst,in_port)
    # Gives Match field
    self.add_flow(datapath, 1, match, actions, buffer_id=None)
    # adding flow in the table

```

The IP packets are handled similarly. The incoming packets are matched for destination IP in the mac table to find the port in which the packets can be sent out. If a hit is found, the packet is forwarded to the output port. Else the controller will follow the same procedure as before, but the controller itself will check IP address, switch port details and configures flows.


```

def handle_IP(self,event):
    # handle IP packets
    datapath = event.msg.datapath
    # datapath connection
    ofproto = datapath.ofproto
    #ofproto of the datapath
    in_port = event.msg.match['in_port']
    # port through which the switch recieved this packet
    parser = datapath.ofproto_parser
    pkt = packet.Packet(data=event.msg.data)
    eth = pkt.get_protocols(ethernet.ethernet)[0]
    # Fetching ethernet dataframe
    ip_pkt = pkt.get_protocol(ipv4.ipv4)
    # fetching IP packet
    out_port = self.check_mactable(ofproto,'ip',ip_pkt.dst)
    # check IP destination in the Table
    match = self.simplematch(parser,eth.src,eth.dst,in_port)
    #
    actions = [parser.OFPACTIONOutput(port=out_port)]
    #
    if event.msg.buffer_id != ofproto.OFP_NO_BUFFER:
        self.add_flow(datapath, 1, match, actions, event.msg.buffer_id)
        # adding a flow in case of no buffer
    else:
        self.add_flow(datapath, 1, match, actions)

```

The check mac table function handles ARP and IP packets separately.

```

def check_mactable(self,ofproto,caller,para):
    # to check if an mac addr or IP addr exists in mac table
    if caller == 'arp':
        # if the calling function is arp, then check mac address
        for p in self.mac_to_port:
            if self.mac_to_port[p][0] == para:
                #[p][0]
                return p
            # return p as outport # if found return
    elif caller == 'ip':
        #if calling function is ip , then check ip addr
        for p in self.mac_to_port:
            if self.mac_to_port[p][1] == para:
                #[p][1]
                return p
            # return corresponding port
    return ofproto.OFPP_FLOOD
    # if no port is found

```

The send to controller function helps to send the packets to the controller when the initial flow is not defined.

```
def sendto_controller(self,event):
    # initial installation of table miss flow
    datapath = event.msg.datapath
    #.
    ofproto = datapath.ofproto
    #.
    parser = datapath.ofproto_parser
    #.
    match = parser.OFPMatch()
    #.
    actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,ofproto.OFPCML_NO_BUFFER)]
    #send to
    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]
    mod = parser.OFPFlowMod(datapath=datapath,priority=0,match=match,instructions=inst)
    datapath.send_msg(mod)
```

Add flow() helps in adding the flow on the switches, switchport_out() will forward the packet in that particular output port, and simple match() will help to perform a simple L2 matching.

```
def add_flow(self, datapath, priority, match, actions, buffer_id=None):
    #
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    idle_timeout=45
    # idle-timeout set to 45 seconds
    hard_timeout=45
    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]
    #forming instructions
    if buffer_id:
        mod = parser.OFPFlowMod(datapath=datapath,buffer_id=buffer_id,priority=priority,idle_timeout=idle_timeout,hard_timeout = hard_timeout,
                                instructions=inst)
    else:
        mod = parser.OFPFlowMod(datapath=datapath,priority=priority,match=match,idle_timeout=idle_timeout, hard_timeout = hard_timeout,instructions=inst)
    self.logger.info("added flow for %s",mod)
    datapath.send_msg(mod)

#### request the packet to be forwarded onto a specific port from the switch ####
def switchport_out(self,pkt,datapath,port):
    #.'accept raw data , serialise it and packetout from a OF switch '' #
    ofproto = datapath.ofproto
    #.
    parser = datapath.ofproto_parser
    #.
    pkt.serialize()
    #. serialise packet (ie convert raw data)
    self.logger.info("packet-out %s" %(pkt,))
    #.
    data = pkt.data
    #.
    actions = [parser.OFPActionOutput(port=port)]
    #.
    out = parser.OFPacketOut(datapath = datapath,buffer_id=ofproto.OFP_NO_BUFFER,in_port=ofproto.OFPP_CONTROLLER,actions=actions,data=data)
    #.
    datapath.send_msg(out)
    #.

def simplematch(self,parser,src,dst,in_port):
    match = parser.OFPMatch(in_port=in_port,eth_dst=dst,eth_src=src)
    #
    return match
```

Implementation in VM with configuring one switch S1:

First, instantiate the switch in RYU app.


```

osboxes@osboxes:~/ryu$ ./bin/ryu-manager --verbose ryu/app/L2L3Switch.py
loading app ryu/app/L2L3Switch.py
loading app ryu.controller.ofp_handler
instantiating app ryu/app/L2L3Switch.py of L2L3Switch
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK L2L3Switch
  CONSUMES EventOFPPacketIn
  CONSUMES EventOFPSwitchFeatures
BRICK ofp_event
  PROVIDES EventOFPPacketIn TO {'L2L3Switch': set(['main'])}
  PROVIDES EventOFPSwitchFeatures TO {'L2L3Switch': set(['config'])}
  CONSUMES EventOFPEchoReply
  CONSUMES EventOFPPortStatus
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPEErrorMsg
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPHello
  CONSUMES EventOFPEchoRequest

```

When we run the topology, the switch gets connected,

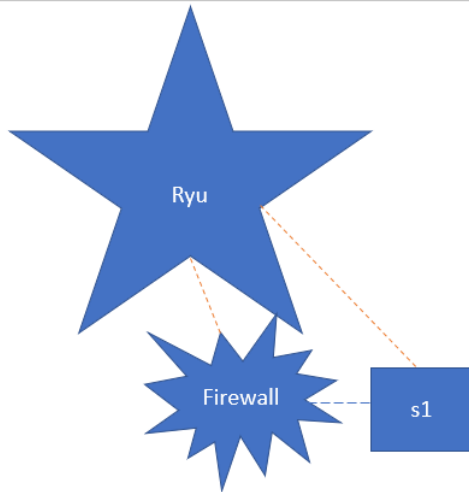
```

instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK L2L3Switch
  CONSUMES EventOFPPacketIn
  CONSUMES EventOFPSwitchFeatures
BRICK ofp_event
  PROVIDES EventOFPPacketIn TO {'L2L3Switch': set(['main'])}
  PROVIDES EventOFPSwitchFeatures TO {'L2L3Switch': set(['config'])}
  CONSUMES EventOFPEchoReply
  CONSUMES EventOFPPortStatus
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPEErrorMsg
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPHello
  CONSUMES EventOFPEchoRequest
connected socket:<eventlet.greenio.base.GreenSocket object at 0xb631bdcc> address:('127.0.0.1', 52540)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0xb6ed308c>
move onto config mode
EVENT ofp_event->L2L3Switch EventOFPSwitchFeatures
switch features ev version=0x4,msg_type=0x6,msg_len=0x20,xid=0xdc2e046aL,OFPSwitchFeatures(auxiliary_id=0,capabilities=79,datapath_id=2,n_buffers=256,n_tables=254)
*** in feature handler ***

```

5.3 .Security and Firewall Implementation

For the firewall implementation and defining firewall rules, we use `rest_firewall.py` in Ryu app. The firewall basically connects to the switches in the topology and helps in configuring the rules in the switch. The `rest_firewall.py` is explained in the later part of this document. We will first show the implementation of a firewall in Ryu. The block diagram will look like this.



The firewall connects to different switches when the `rest_firewall.py` is run as shown below.

```

osboxes@osboxes:~/ryu$ ./bin/ryu-manager --verbose ryu/app/rest_firewall.py
loading app ryu/app/rest_firewall.py
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app ryu/app/rest_firewall.py of RestFirewallAPI
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK dpset
  PROVIDES EventDP TO {'RestFirewallAPI': set(['dpset'])}
  CONSUMES EventOFPPortStatus
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPPStateChange
BRICK RestFirewallAPI
  CONSUMES EventOFPPacketIn
  CONSUMES EventOFPPFlowStatsReply
  CONSUMES EventDP
  CONSUMES EventOFPPStatsReply
BRICK ofp event
  PROVIDES EventOFPPortStatus TO {'dpset': set(['main'])}
  PROVIDES EventOFPSwitchFeatures TO {'dpset': set(['config'])}
  PROVIDES EventOFPPFlowStatsReply TO {'RestFirewallAPI': set(['main'])}
  PROVIDES EventOFPPStatsReply TO {'RestFirewallAPI': set(['main'])}
  PROVIDES EventOFPPStateChange TO {'dpset': set(['main', 'dead'])}
  PROVIDES EventOFPPPacketIn TO {'RestFirewallAPI': set(['main'])}
  CONSUMES EventOFPEchoReply
  CONSUMES EventOFPPortStatus
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPErrorMsg
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPHello
  CONSUMES EventOFPEchoRequest
(8658) wsgi starting up on http://0.0.0.0:8080
  
```

When the topology is run, all the switches will join the firewall. In the below screen shot we can switch 5, 4 and 7 joined the firewall.

```

EVENT ofp_event->dpset EventOFPStateChange
DPSET: register datapath <ryu.controller.controller.Datapath object at 0xb6096e6c>
EVENT dpset->RestFirewallAPI EventDP
Sending message with xid(de3dea9a) to datapath(0000000000000005): version=None,msg_type=None,msg_len=None,
mask=0,flags=0,hard_timeout=0,idle_timeout=0,instructions=[],match=OFPMatch(oxm_fields={}),out_group=4294
Sending message with xid(de3dea9b) to datapath(0000000000000005): version=None,msg_type=None,msg_len=None,
mask=0,flags=0,hard_timeout=0,idle_timeout=0,instructions=[OFPIInstructionActions(actions=[OFPAActionOutput
m_fields={'eth_type': 2054}),out_group=4294967295L,out_port=4294967295L,priority=65534,table_id=0)
Sending message with xid(de3dea9c) to datapath(0000000000000005): version=None,msg_type=None,msg_len=None,
mask=0,flags=0,hard_timeout=0,idle_timeout=0,instructions=[OFPIInstructionActions(actions=[OFPAActionOutput
fields={})),out_group=4294967295L,out_port=4294967295L,priority=0,table_id=0)
[FW][INFO] dpid=0000000000000005: Join as firewall.
connected socket:<eventlet.greenio.base.GreenSocket object at 0xb609b06c> address:('127.0.0.1', 51006)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0xb60b648c>
move onto config mode
EVENT ofp_event->dpset EventOFPStateChange
DPSET: register datapath <ryu.controller.controller.Datapath object at 0xb60a548c>
EVENT dpset->RestFirewallAPI EventDP
Sending message with xid(14274cdf) to datapath(0000000000000004): version=None,msg_type=None,msg_len=None,
mask=0,flags=0,hard_timeout=0,idle_timeout=0,instructions=[],match=OFPMatch(oxm_fields={}),out_group=4294
Sending message with xid(14274ce0) to datapath(0000000000000004): version=None,msg_type=None,msg_len=None,
mask=0,flags=0,hard_timeout=0,idle_timeout=0,instructions=[OFPIInstructionActions(actions=[OFPAActionOutput
m_fields={'eth_type': 2054}),out_group=4294967295L,out_port=4294967295L,priority=65534,table_id=0)
Sending message with xid(14274ce1) to datapath(0000000000000004): version=None,msg_type=None,msg_len=None,
mask=0,flags=0,hard_timeout=0,idle_timeout=0,instructions=[OFPIInstructionActions(actions=[OFPAActionOutput
fields={})),out_group=4294967295L,out_port=4294967295L,priority=0,table_id=0)
[FW][INFO] dpid=0000000000000004: Join as firewall.
connected socket:<eventlet.greenio.base.GreenSocket object at 0xb60b0bac> address:('127.0.0.1', 51008)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0xb60b6aac>
move onto config mode
EVENT ofp_event->dpset EventOFPStateChange
DPSET: register datapath <ryu.controller.controller.Datapath object at 0xb60b034c>
EVENT dpset->RestFirewallAPI EventDP
Sending message with xid(d7510ae7) to datapath(0000000000000007): version=None,msg_type=None,msg_len=None,
mask=0,flags=0,hard_timeout=0,idle_timeout=0,instructions=[],match=OFPMatch(oxm_fields={}),out_group=4294
Sending message with xid(d7510ae8) to datapath(0000000000000007): version=None,msg_type=None,msg_len=None,
mask=0,flags=0,hard_timeout=0,idle_timeout=0,instructions=[OFPIInstructionActions(actions=[OFPAActionOutput
m_fields={'eth_type': 2054}),out_group=4294967295L,out_port=4294967295L,priority=65534,table_id=0)
Sending message with xid(d7510ae9) to datapath(0000000000000007): version=None,msg_type=None,msg_len=None,
mask=0,flags=0,hard_timeout=0,idle_timeout=0,instructions=[OFPIInstructionActions(actions=[OFPAActionOutput
fields={})),out_group=4294967295L,out_port=4294967295L,priority=0,table_id=0)
[FW][INFO] dpid=0000000000000007: Join as firewall.

```

We used a separate python file to send the curl command to configure the rules in the firewall of switches called my firewall.py. For a sample, we configured firewall rules in communication between hosts hf1 and hf2, as shown below. The urllib2 library is utilized to send the HTTP request directly from python rather than using curl command in the API. It the same procedure. The advantage of using python to

run this is. Obviously we can set or remove multiple firewall rules sequentially from just a single file.

```
from ryu.app import rest_firewall
import urllib2

data = '{"nw_src": "172.16.1.2/32", "nw_dst": "172.16.1.3/32", "nw_proto": "ICMP", "actions": "DENY", "priority": "10"}'
url = 'http://localhost:8080/firewall/rules/0000000000000001'
req = urllib2.Request(url, data, {'Content-Type': 'application/json'})
f = urllib2.urlopen(req)
for x in f:
    print(x)
f.close()

data = '{"nw_src": "172.16.1.3/32", "nw_dst": "172.16.1.4/32", "nw_proto": "ICMP", "actions": "ALLOW", "priority": "10"}'
url = 'http://localhost:8080/firewall/rules/0000000000000001'
req = urllib2.Request(url, data, {'Content-Type': 'application/json'})
f = urllib2.urlopen(req)
for x in f:
    print(x)
f.close()
```

The network source IP as 172.16.1.2/32, which is IP of hf1 and destination IP as 172.16.1.3/32, which is IP of hf2 are utilized in the first firewall rule. ICMP protocol communication has been blocked in both these hosts by the firewall in switch 1. It means the hf1 ping hf2 will not work.

The image displays three terminal windows. The top-left window shows the output of a REST API call to add a firewall rule, including details like 'POST /firewall/rules/0000000000000001 HTTP/1.1' and '200 244 0.008698'. The top-right window shows the output of a Mininet CLI, including network configuration, host configuration, and a successful ping from hf1 to hf2 (56(84) bytes of data). The bottom-left window shows the output of a REST API call to add a second firewall rule, including details like 'POST /firewall/rules/0000000000000001 HTTP/1.1' and '200 244 0.008698'.

The left top screen in the firewall, the right top is the mininet and left bottom is the custom made rules set. As expected hf1 cannot ping hf2 because the rule has been set as DENY.

5.3 Implementation Benefits in P4 switches

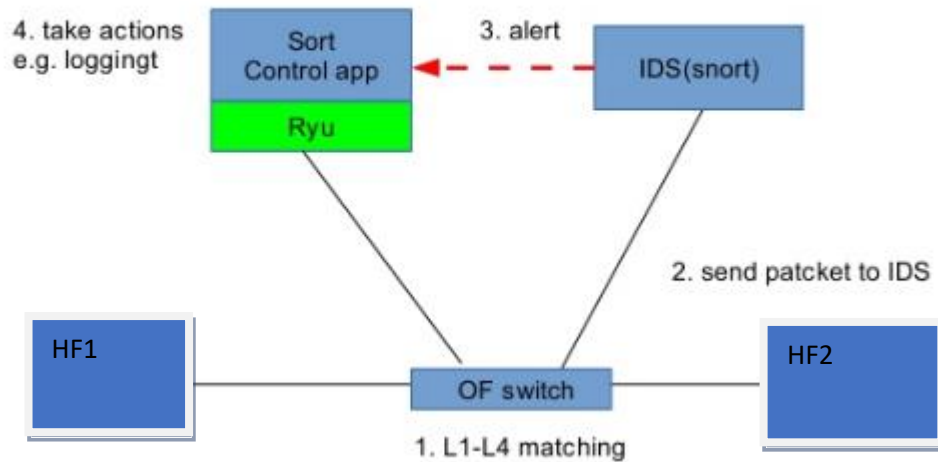
P4 is a technology which helps in data plane programmability independent of the underlying physical switch properties. SDN helps in control plane programming, but the open flow protocol header length keeps increasing as versions get updated. There is only a limited length of messages that can be processed by the underlying traditional physical switches. Moreover, they support only fixed number of protocols. With open flow versions frequently changing P4 provides a completely programmable device which can be reconciled to our needs. These switches enable a new language to configure them called P4 language. The most common benefits of this scheme are :

1. We can design entirely new protocols without bothering about the physical processing capacity of the switches involved.
2. It can remove unwanted protocols that are never used
3. SW based development can be achieved even in data plane completely
4. The reliability on third party vendors on choosing the functionality of the device is removed.
5. The resources utilized for routing table flows is completely flexible with P4 switches which is not possible with OFPS.

5.4 Intrusion Detection System – SNORT

We have deployed an IDS an SDN application in our Ryu controller. Besides the firewall, we designed an Intrusion Detection System (IDS) for the network to enhance security features. An IDS is a type of security software which alerts administrators when someone is trying to compromise information system through malicious activities or security policy violations. An IDS works by examining and monitoring networks and system activities and examine all the vulnerabilities in the network. It also automatically monitors the Internet for any possible threats.

We can run IDS in Ryu with Snort. Snort is a free open source IDS. Ryu receives Snort alert packet via Unix Domain Socket.



Snort application will run from the ryu app folder and an instance will be created. A copy of all the packets traveling between HF1 and HF2 through the OF Switch will be forwarded to snort IDS sniffer, so when the intrusion is detected it will be sent to the ryu controller, as shown in the figure above. So, a simple snort program can be found in the ryu app folder with the name `simple_switch_snort.py`. In the code snort application has only one additional function to report an intrusion that has been detected using `__dump__alert` function, which will be triggered with the decorator event under `snortlib.EventAlert`.

```
@set_ev_cls(snortlib.EventAlert, MAIN_DISPATCHER)
def __dump_alert(self, ev):
    msg = ev.msg

    print('alertmsg: %s' % ''.join(msg.alertmsg))
    self.packet_print(msg.pkt)
```

Logging functionality is enabled using the below function, which prints all the captured IP, ethernet and ICMP packets that were captured during the `__dump_alert()`.

```
def packet_print(self, pkt):
    pkt = packet.Packet(array.array('B', pkt))

    eth = pkt.get_protocol(ethernet.ethernet)
    _ipv4 = pkt.get_protocol(ipv4.ipv4)
    _icmp = pkt.get_protocol(icmp.icmp)

    if _icmp:
        self.logger.info("%r", _icmp)

    if _ipv4:
        self.logger.info("%r", _ipv4)

    if eth:
        self.logger.info("%r", eth)
```

When we run the snort file i.e. `simple_switch_snort.py`, we find the below output.

```
loading app ryu.controller.ofp_handler
instantiating app None of SnortLib
creating context snortlib
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu/app/simple_switch_snort.py of SimpleSwitchSnort
[snort][INFO] {'unixsock': True}
BRICK SimpleSwitchSnort
  CONSUMES EventOFPPacketIn
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventAlert
BRICK snortlib
  PROVIDES EventAlert TO {'SimpleSwitchSnort': set(['main'])}
BRICK ofp event
  PROVIDES EventOFPPacketIn TO {'SimpleSwitchSnort': set(['main'])}
  PROVIDES EventOFPSwitchFeatures TO {'SimpleSwitchSnort': set(['config'])}
  CONSUMES EventOFPEchoReply
  CONSUMES EventOFPPortStatus
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPErrormsg
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPHello
  CONSUMES EventOFPEchoRequest
[snort][INFO] Unix socket start listening...
```

When we run the topology, the switches get connected to the snort application and all the packets travelling through the switches in our topology can be observed and monitored in the snort application.

```

*** s3 : ('ifconfig s3-eth21 172.16.60.22 netmask 255.255.255.0')
*** s3 : ('ifconfig s3-eth22 172.16.60.23 netmask 255.255.255.0')
*** s3 : ('ifconfig s3-eth23 172.16.60.24 netmask 255.255.255.0')
*** s3 : ('ifconfig s3-eth24 172.16.60.25 netmask 255.255.255.0')
*** s3 : ('ifconfig s3-eth25 172.16.60.26 netmask 255.255.255.0')
*** s3 : ('ifconfig s3-eth26 172.16.60.27 netmask 255.255.255.0')
*** s3 : ('ifconfig s3-eth27 172.16.60.28 netmask 255.255.255.0')
*** s3 : ('ifconfig s3-eth28 172.16.60.29 netmask 255.255.255.0')
*** s3 : ('ifconfig s3-eth29 172.16.60.30 netmask 255.255.255.0')
*** s3 : ('ifconfig s3-eth30 172.16.60.31 netmask 255.255.255.0')
*** s3 : ('ifconfig s3-eth31 172.16.70.32 netmask 255.255.255.0')
*** s3 : ('ifconfig s3-eth32 172.16.70.33 netmask 255.255.255.0')
*** s3 : ('ifconfig s3-eth33 172.16.70.34 netmask 255.255.255.0')
*** s3 : ('ifconfig s3-eth34 172.16.70.35 netmask 255.255.255.0')
*** s3 : ('ifconfig s3-eth35 172.16.70.36 netmask 255.255.255.0')
*** s3 : ('ifconfig s3-eth36 172.16.50.2 netmask 255.255.255.0')
***Adding common, dedicated servers and linking them to switches
*** s6 : ('ifconfig s6-eth1 172.16.80.2 netmask 255.255.255.0')
*** s6 : ('ifconfig s6-eth2 172.16.80.3 netmask 255.255.255.0')
*** s6 : ('ifconfig s6-eth3 172.16.80.4 netmask 255.255.255.0')
*** s6 : ('ifconfig s6-eth4 172.16.80.5 netmask 255.255.255.0')
*** s6 : ('ifconfig s6-eth5 172.16.80.6 netmask 255.255.255.0')
*** s6 : ('ifconfig s6-eth6 172.16.80.7 netmask 255.255.255.0')
*** s6 : ('ifconfig s6-eth7 172.16.80.8 netmask 255.255.255.0')
*** s6 : ('ifconfig s6-eth8 172.16.80.9 netmask 255.255.255.0')
*** s6 : ('ifconfig s6-eth9 172.16.80.10 netmask 255.255.255.0')
*** s6 : ('ifconfig s6-eth10 172.16.80.11 netmask 255.255.255.0')
*** s6 : ('ifconfig s6-eth11 172.16.50.2 netmask 255.255.255.0')
*** s6 : ('ifconfig s6-eth12 172.16.50.3 netmask 255.255.255.0')
*** s6 : ('ifconfig s6-eth13 172.16.50.4 netmask 255.255.255.0')
*** s6 : ('ifconfig s6-eth14 172.16.50.5 netmask 255.255.255.0')
*** s6 : ('ifconfig s6-eth15 172.16.50.6 netmask 255.255.255.0')
***Adding links between switches
***starting network
*** Configuring hosts
hf1 hf2 hf3 hf4 hf5 hf6 hf7 hf8 hf9 hf10 hf11 hf12 hf13 hf14 hf15 hf16 hf17 hf18 hf19 hf20 hf21 hf22 hf23 hf24 hf25 hf26 hf27 hf28 hf29 hf30 hf31 hf32 hf33 hf34 hf35 hf36 hf37 hf38 hf39 hf40 hf41 hf42 hf43 hf44 hf45 hf46 hf47 hf48 hf49 hf50 hf51 hf52 hf53 hf54 hf55 hf56 hf57 hf58 hf59 hf60 hf61 hf62 hf63 hf64 hf65 hf66 hf67 hf68 hf69 hf70 hf71 hf72 hf73 hf74 hf75 hf76 hf77 hf78 hf79 hf80 hf81 hf82 hf83 hf84 hf85 hf86 hf87 hf88 hf89 hf90 hf91 hf92 hf93 hf94 hf95 hf96 hf97 hf98 hf99 hf100 hf101 hf102 hf103 hf104 hf105 hf106 hf107 hf108 hf109 hf110 hf111 hf112 hf113 hf114 hf115 hf116 hf117 hf118 hf119 hf120 hf121 hf122 hf123 hf124 hf125 hf126 hf127 hf128 hf129 hf130 hf131 hf132 hf133 hf134 hf135 hf136 hf137 hf138 hf139 hf140 hf141 hf142 hf143 hf144 hf145 hf146 hf147 hf148 hf149 hf150 hf151 hf152 hf153 hf154 hf155 hf156 hf157 hf158 hf159 hf160 hf161 hf162 hf163 hf164 hf165 hf166 hf167 hf168 hf169 hf170 hf171 hf172 hf173 hf174 hf175 hf176 hf177 hf178 hf179 hf180 hf181 hf182 hf183 hf184 hf185 hf186 hf187 hf188 hf189 hf190 hf191 hf192 hf193 hf194 hf195 hf196 hf197 hf198 hf199 hf200 hf201 hf202 hf203 hf204 hf205 hf206 hf207 hf208 hf209 hf210 hf211 hf212 hf213 hf214 hf215 hf216 hf217 hf218 hf219 hf220 hf221 hf222 hf223 hf224 hf225 hf226 hf227 hf228 hf229 hf230 hf231 hf232 hf233 hf234 hf235 hf236 hf237 hf238 hf239 hf240 hf241 hf242 hf243 hf244 hf245 hf246 hf247 hf248 hf249 hf250 hf251 hf252 hf253 hf254 hf255 hf256 hf257 hf258 hf259 hf260 hf261 hf262 hf263 hf264 hf265 hf266 hf267 hf268 hf269 hf270 hf271 hf272 hf273 hf274 hf275 hf276 hf277 hf278 hf279 hf280 hf281 hf282 hf283 hf284 hf285 hf286 hf287 hf288 hf289 hf290 hf291 hf292 hf293 hf294 hf295 hf296 hf297 hf298 hf299 hf300 hf301 hf302 hf303 hf304 hf305 hf306 hf307 hf308 hf309 hf310 hf311 hf312 hf313 hf314 hf315 hf316 hf317 hf318 hf319 hf320 hf321 hf322 hf323 hf324 hf325 hf326 hf327 hf328 hf329 hf330 hf331 hf332 hf333 hf334 hf335 hf336 hf337 hf338 hf339 hf340 hf341 hf342 hf343 hf344 hf345 hf346 hf347 hf348 hf349 hf350 hf351 hf352 hf353 hf354 hf355 hf356 hf357 hf358 hf359 hf360 hf361 hf362 hf363 hf364 hf365 hf366 hf367 hf368 hf369 hf370 hf371 hf372 hf373 hf374 hf375 hf376 hf377 hf378 hf379 hf380 hf381 hf382 hf383 hf384 hf385 hf386 hf387 hf388 hf389 hf390 hf391 hf392 hf393 hf394 hf395 hf396 hf397 hf398 hf399 hf400 hf401 hf402 hf403 hf404 hf405 hf406 hf407 hf408 hf409 hf410 hf411 hf412 hf413 hf414 hf415 hf416 hf417 hf418 hf419 hf420 hf421 hf422 hf423 hf424 hf425 hf426 hf427 hf428 hf429 hf430 hf431 hf432 hf433 hf434 hf435 hf436 hf437 hf438 hf439 hf440 hf441 hf442 hf443 hf444 hf445 hf446 hf447 hf448 hf449 hf450 hf451 hf452 hf453 hf454 hf455 hf456 hf457 hf458 hf459 hf460 hf461 hf462 hf463 hf464 hf465 hf466 hf467 hf468 hf469 hf470 hf471 hf472 hf473 hf474 hf475 hf476 hf477 hf478 hf479 hf480 hf481 hf482 hf483 hf484 hf485 hf486 hf487 hf488 hf489 hf490 hf491 hf492 hf493 hf494 hf495 hf496 hf497 hf498 hf499 hf500 hf501 hf502 hf503 hf504 hf505 hf506 hf507 hf508 hf509 hf510 hf511 hf512 hf513 hf514 hf515 hf516 hf517 hf518 hf519 hf520 hf521 hf522 hf523 hf524 hf525 hf526 hf527 hf528 hf529 hf530 hf531 hf532 hf533 hf534 hf535 hf536 hf537 hf538 hf539 hf540 hf541 hf542 hf543 hf544 hf545 hf546 hf547 hf548 hf549 hf550 hf551 hf552 hf553 hf554 hf555 hf556 hf557 hf558 hf559 hf560 hf561 hf562 hf563 hf564 hf565 hf566 hf567 hf568 hf569 hf570 hf571 hf572 hf573 hf574 hf575 hf576 hf577 hf578 hf579 hf580 hf581 hf582 hf583 hf584 hf585 hf586 hf587 hf588 hf589 hf590 hf591 hf592 hf593 hf594 hf595 hf596 hf597 hf598 hf599 hf600 hf601 hf602 hf603 hf604 hf605 hf606 hf607 hf608 hf609 hf610 hf611 hf612 hf613 hf614 hf615 hf616 hf617 hf618 hf619 hf620 hf621 hf622 hf623 hf624 hf625 hf626 hf627 hf628 hf629 hf630 hf631 hf632 hf633 hf634 hf635 hf636 hf637 hf638 hf639 hf640 hf641 hf642 hf643 hf644 hf645 hf646 hf647 hf648 hf649 hf650 hf651 hf652 hf653 hf654 hf655 hf656 hf657 hf658 hf659 hf660 hf661 hf662 hf663 hf664 hf665 hf666 hf667 hf668 hf669 hf670 hf671 hf672 hf673 hf674 hf675 hf676 hf677 hf678 hf679 hf680 hf681 hf682 hf683 hf684 hf685 hf686 hf687 hf688 hf689 hf690 hf691 hf692 hf693 hf694 hf695 hf696 hf697 hf698 hf699 hf700 hf701 hf702 hf703 hf704 hf705 hf706 hf707 hf708 hf709 hf710 hf711 hf712 hf713 hf714 hf715 hf716 hf717 hf718 hf719 hf720 hf721 hf722 hf723 hf724 hf725 hf726 hf727 hf728 hf729 hf730 hf731 hf732 hf733 hf734 hf735 hf736 hf737 hf738 hf739 hf740 hf741 hf742 hf743 hf744 hf745 hf746 hf747 hf748 hf749 hf750 hf751 hf752 hf753 hf754 hf755 hf756 hf757 hf758 hf759 hf760 hf761 hf762 hf763 hf764 hf765 hf766 hf767 hf768 hf769 hf770 hf771 hf772 hf773 hf774 hf775 hf776 hf777 hf778 hf779 hf780 hf781 hf782 hf783 hf784 hf785 hf786 hf787 hf788 hf789 hf790 hf791 hf792 hf793 hf794 hf795 hf796 hf797 hf798 hf799 hf800 hf801 hf802 hf803 hf804 hf805 hf806 hf807 hf808 hf809 hf810 hf811 hf812 hf813 hf814 hf815 hf816 hf817 hf818 hf819 hf820 hf821 hf822 hf823 hf824 hf825 hf826 hf827 hf828 hf829 hf830 hf831 hf832 hf833 hf834 hf835 hf836 hf837 hf838 hf839 hf840 hf841 hf842 hf843 hf844 hf845 hf846 hf847 hf848 hf849 hf850 hf851 hf852 hf853 hf854 hf855 hf856 hf857 hf858 hf859 hf860 hf861 hf862 hf863 hf864 hf865 hf866 hf867 hf868 hf869 hf870 hf871 hf872 hf873 hf874 hf875 hf876 hf877 hf878 hf879 hf880 hf881 hf882 hf883 hf884 hf885 hf886 hf887 hf888 hf889 hf890 hf891 hf892 hf893 hf894 hf895 hf896 hf897 hf898 hf899 hf900 hf901 hf902 hf903 hf904 hf905 hf906 hf907 hf908 hf909 hf910 hf911 hf912 hf913 hf914 hf915 hf916 hf917 hf918 hf919 hf920 hf921 hf922 hf923 hf924 hf925 hf926 hf927 hf928 hf929 hf930 hf931 hf932 hf933 hf934 hf935 hf936 hf937 hf938 hf939 hf940 hf941 hf942 hf943 hf944 hf945 hf946 hf947 hf948 hf949 hf950 hf951 hf952 hf953 hf954 hf955 hf956 hf957 hf958 hf959 hf960 hf961 hf962 hf963 hf964 hf965 hf966 hf967 hf968 hf969 hf970 hf971 hf972 hf973 hf974 hf975 hf976 hf977 hf978 hf979 hf980 hf981 hf982 hf983 hf984 hf985 hf986 hf987 hf988 hf989 hf990 hf991 hf992 hf993 hf994 hf995 hf996 hf997 hf998 hf999 hf1000
server1 server2 server3 server4 server5 server6 server7 server8 server9 server10 server11 server12 server13 server14 server15 dserver1 dserver2 dserver3 dserver4 dserver5 dserver6 dserver7 dserver8 dserver9 dserver10 dserver11 dserver12 dserver13 dserver14 dserver15
***Starting switches
***post configure switches and hosts
*** Starting CLI:
mininet>

```

Snort can also be independently run as IDS application in the computer systems. This can also be called inside the application layer of the host. We, explained this functionality in the further sections of this document.

Snort can be configured to run in three modes:

1- Sniffer Mode

Reading the packets and display them in the monitor.

2- Packet Logger Mode

Logs the packets to disk

3- Network Intrusion Detection Service (NIDS)

Performing detection and analysis on the network traffic.

After successful installation of Snort, we can check the version and make sure it is correctly installed on our system by issuing the command:

Snort -V


```

osboxes@osboxes:/usr/local/bin$ snort -V

o",_~
'    '

-*> Snort++ <*-
Version 3.0.0 (Build 255) from 2.9.11
By Martin Roesch & The Snort Team
http://snort.org/contact#team
Copyright (C) 2014-2019 Cisco and/or its affiliates. All rights reserved.
Copyright (C) 1998-2013 Sourcefire, Inc., et al.
Using DAQ version 2.2.2
Using LuaJIT version 2.0.4
Using OpenSSL 1.0.2g 1 Mar 2016
Using libpcap version 1.7.4
Using PCRE version 8.38 2015-11-23
Using ZLIB version 1.2.8
Using Hyperscan version 4.7.0 2019-05-27

osboxes@osboxes:/usr/local/bin$

```

To make the best of snort, we have installed some additional rules. The following screenshots show the installed rules on our system:

```

rule counts
  total rules loaded: 829
    text rules: 829
  option chains: 829
  chain headers: 46
-----
port rule counts
      tcp      udp      icmp      ip
any      63       3        0        0
src     124       3        0        0
dst     539      98        0        0
both      0        1        0        0
total    726     105        0        0
-----
flowbits
      defined: 20
      not checked: 11
      not set: 3
-----
service rule counts - tcp      to-srv  to-cli
      dns:          1        0
      ftp:          7        2
      ftp-data:      0        8
      http:         485      92
      imap:          0        8
      irc:           4        1
      netbios-ssn:   15        1
      pop3:          0        8
      smtp:         16        0
      ssl:          14       31
      telnet:        1        0
      total:        543     151

```

```

service rule counts - udp      to-srv  to-cli
                        dns:      88      2
                        http:     4       0
                        total:    92      2
-----
fast pattern port groups      src      dst      any
packet:                       12      24      2
-----
fast pattern service groups  to-srv  to-cli
packet:                       10      6
key:                          1       0
header:                       1       4
body:                         1       0
file:                         2       4
-----
search engine
      instances: 64
      patterns: 1135
      pattern chars: 23239
      num states: 18288
      num match states: 1117
      memory scale: KB
      total memory: 423.624
      pattern memory: 49.2334
      match list memory: 91.4375
      transition memory: 276.703
-----
pcap DAQ configured to passive.

Snort successfully validated the configuration (with 0 warnings).
o")~ Snort exiting
osboxes@osboxes:~/snort_src/snort3-community-rules$

```

Setting up rules

In this part, we are defining two different rules to illustrate some certain types of traffic which IDS captures. We installed OpenAppID to specify the application layer traffic. With the help of OpenAppID, we can create rules to operate on application layer traffic.

After installation of OpenAppID in snort, we are going to define to separate rules for this illustration:

First Rule:

```

alert TCP any any -> any any ( msg:"Facebook traffic Seen";
appids:"Facebook";sid:10000001; )

```

This rule is going to Alert administrator that someone is trying to access Facebook. Based on the requirements, we are trying to minimize distractions from social media. Hence, this rule can be applied for all social media websites and applications besides firewall rules.

Second Rule:

```
alert icmp any any -> any any (msg:"ICMP Traffic Detected";sid:10000002;)
```

This is simply an alert administrator about ICMP or issuing ping command.

To observe the result, we need to run Snort in detection mode on an interface. The following command is going to enable detection mode:

```
sudo snort -c /usr/local/etc/snort/snort.lua -R /usr/local/etc/snort/rules/local.rules \ -
i ens34 -A alert_fast -s 65535 -k none
```

Once we issue this command, the interface is switching to a listening state for any protocols:

```
pcap DAQ configured to passive.
Commencing packet processing
++ [0] ens34
05/28-06:10:43.647761 [**] [116:414:1] "(ipv4) IPv4 packet to broadcast dest address" [**] [Priority: 3] {UDP}
192.168.75.1:57043 -> 255.255.255.255:3289
05/28-06:10:48.612693 [**] [116:414:1] "(ipv4) IPv4 packet to broadcast dest address" [**] [Priority: 3] {UDP}
192.168.75.1:57043 -> 255.255.255.255:3289
05/28-06:10:53.612994 [**] [116:414:1] "(ipv4) IPv4 packet to broadcast dest address" [**] [Priority: 3] {UDP}
192.168.75.1:57043 -> 255.255.255.255:3289
05/28-06:11:13.035223 [**] [1:10000002:0] "ICMP Traffic Detected" [**] [Priority: 0] {ICMP} fe80::4ee:7fd9:834:
9030 -> ff02::1:ffcc:332
05/28-06:11:36.260987 [**] [116:414:1] "(ipv4) IPv4 packet to broadcast dest address" [**] [Priority: 3] {UDP}
192.168.75.1:61984 -> 255.255.255.255:3289
```

As soon as we issue the command `wget facebook.com` in another terminal, we can observe the alert outputs output:

```

osboxes@osboxes:~$ wget facebook.com
--2019-05-28 06:04:06-- http://facebook.com/
Resolving facebook.com (facebook.com)... 157.240.8.35, 2a03:2880:f119:8083:face:b00c:0:25de
Connecting to facebook.com (facebook.com)|157.240.8.35|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://facebook.com/ [following]
--2019-05-28 06:04:07-- https://facebook.com/
Connecting to facebook.com (facebook.com)|157.240.8.35|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://www.facebook.com/ [following]
--2019-05-28 06:04:07-- https://www.facebook.com/
Resolving www.facebook.com (www.facebook.com)... 157.240.8.35, 2a03:2880:f119:8083:face:b00c:0:25de
Connecting to www.facebook.com (www.facebook.com)|157.240.8.35|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://www.facebook.com/unsupportedbrowser [following]
--2019-05-28 06:04:07-- https://www.facebook.com/unsupportedbrowser
Reusing existing connection to www.facebook.com:443.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: 'index.html.2'

index.html.2          [ <=>                ] 51.14K  --.-KB/s    in 0.05s
2019-05-28 06:04:07 (957 KB/s) - 'index.html.2' saved [52372]

```

Alerts Output:

```

05/28-04:41:07.284672 [**] [1:10000001:0] "Facebook trafic Seen" [**] [Priority: 0] [AppID: Facebook] {TCP} 157
.240.8.35:443 -> 192.168.75.134:58954
05/28-04:41:07.284697 [**] [1:10000001:0] "Facebook trafic Seen" [**] [Priority: 0] [AppID: Facebook] {TCP} 192
.168.75.134:58954 -> 157.240.8.35:443
05/28-04:41:07.285482 [**] [1:10000001:0] "Facebook trafic Seen" [**] [Priority: 0] [AppID: Facebook] {TCP} 157
.240.8.35:443 -> 192.168.75.134:58954
05/28-04:41:07.285751 [**] [1:10000001:0] "Facebook trafic Seen" [**] [Priority: 0] [AppID: Facebook] {TCP} 157
.240.8.35:443 -> 192.168.75.134:58954
05/28-04:41:07.285789 [**] [1:10000001:0] "Facebook trafic Seen" [**] [Priority: 0] [AppID: Facebook] {TCP} 192
.168.75.134:58954 -> 157.240.8.35:443
05/28-04:41:07.290249 [**] [1:10000001:0] "Facebook trafic Seen" [**] [Priority: 0] [AppID: Facebook] {TCP} 157
.240.8.35:443 -> 192.168.75.134:58954
05/28-04:41:07.290402 [**] [1:10000001:0] "Facebook trafic Seen" [**] [Priority: 0] [AppID: Facebook] {TCP} 157
.240.8.35:443 -> 192.168.75.134:58954
05/28-04:41:07.290636 [**] [1:10000001:0] "Facebook trafic Seen" [**] [Priority: 0] [AppID: Facebook] {TCP} 192
.168.75.134:58954 -> 157.240.8.35:443
05/28-04:41:07.291233 [**] [1:10000001:0] "Facebook trafic Seen" [**] [Priority: 0] [AppID: Facebook] {TCP} 157
.240.8.35:443 -> 192.168.75.134:58954
05/28-04:41:07.291298 [**] [1:10000001:0] "Facebook trafic Seen" [**] [Priority: 0] [AppID: Facebook] {TCP} 192
.168.75.134:58954 -> 157.240.8.35:443
05/28-04:41:07.291712 [**] [1:10000001:0] "Facebook trafic Seen" [**] [Priority: 0] [AppID: Facebook] {TCP} 157
.240.8.35:443 -> 192.168.75.134:58954
05/28-04:41:07.295784 [**] [1:10000001:0] "Facebook trafic Seen" [**] [Priority: 0] [AppID: Facebook] {TCP} 192
.168.75.134:58954 -> 157.240.8.35:443
05/28-04:41:07.296313 [**] [1:10000001:0] "Facebook trafic Seen" [**] [Priority: 0] [AppID: Facebook] {TCP} 157
.240.8.35:443 -> 192.168.75.134:58954
05/28-04:41:07.312907 [**] [1:10000001:0] "Facebook trafic Seen" [**] [Priority: 0] [AppID: Facebook] {TCP} 157
.240.8.35:443 -> 192.168.75.134:58954
05/28-04:41:07.313062 [**] [1:10000001:0] "Facebook trafic Seen" [**] [Priority: 0] [AppID: Facebook] {TCP} 192
.168.75.134:58954 -> 157.240.8.35:443
05/28-04:41:17.165517 [**] [116:414:1] "(ipv4) IPv4 packet to broadcast dest address" [**] [Priority: 3] {UDP}
192.168.75.1:58286 -> 255.255.255.255:3289

```

It is simply the same for the second rule, by issuing ping command:

Ping 8.8.8.8

```

05/28-04:43:52.897528 [**] [112:1:1] "(arp_spoof) unicast ARP request" [**] [Priority: 3] {ARP} ->
05/28-04:43:54.375347 [**] [116:414:1] "(ipv4) IPv4 packet to broadcast dest address" [**] [Priority: 3] {UDP}
192.168.75.1:64399 -> 255.255.255.255:3289
05/28-04:43:57.232710 [**] [1:10000002:0] "ICMP Traffic Detected" [**] [Priority: 0] {ICMP} 192.168.75.134 -> 8
.8.8.8
05/28-04:43:57.242805 [**] [1:10000002:0] "ICMP Traffic Detected" [**] [Priority: 0] {ICMP} 8.8.8.8 -> 192.168.
75.134
05/28-04:43:58.234339 [**] [1:10000002:0] "ICMP Traffic Detected" [**] [Priority: 0] {ICMP} 192.168.75.134 -> 8
.8.8.8
05/28-04:43:58.242520 [**] [1:10000002:0] "ICMP Traffic Detected" [**] [Priority: 0] {ICMP} 8.8.8.8 -> 192.168.
75.134
05/28-04:43:59.235984 [**] [1:10000002:0] "ICMP Traffic Detected" [**] [Priority: 0] {ICMP} 192.168.75.134 -> 8
.8.8.8
05/28-04:43:59.244212 [**] [1:10000002:0] "ICMP Traffic Detected" [**] [Priority: 0] {ICMP} 8.8.8.8 -> 192.168.
75.134
05/28-04:43:59.332230 [**] [116:414:1] "(ipv4) IPv4 packet to broadcast dest address" [**] [Priority: 3] {UDP}
192.168.75.1:64399 -> 255.255.255.255:3289
05/28-04:44:00.238432 [**] [1:10000002:0] "ICMP Traffic Detected" [**] [Priority: 0] {ICMP} 192.168.75.134 -> 8
.8.8.8
05/28-04:44:00.248036 [**] [1:10000002:0] "ICMP Traffic Detected" [**] [Priority: 0] {ICMP} 8.8.8.8 -> 192.168.
75.134
05/28-04:44:01.286122 [**] [1:10000002:0] "ICMP Traffic Detected" [**] [Priority: 0] {ICMP} 192.168.75.134 -> 8
.8.8.8
05/28-04:44:01.294769 [**] [1:10000002:0] "ICMP Traffic Detected" [**] [Priority: 0] {ICMP} 8.8.8.8 -> 192.168.
75.134
05/28-04:44:02.345795 [**] [1:10000002:0] "ICMP Traffic Detected" [**] [Priority: 0] {ICMP} 192.168.75.134 -> 8
.8.8.8
05/28-04:44:02.355687 [**] [1:10000002:0] "ICMP Traffic Detected" [**] [Priority: 0] {ICMP} 8.8.8.8 -> 192.168.
75.134

```

5.5 Practical open flow switches for implementation of this topology:

Our design is a hybrid model and makes use of both hardware and software SDN enabled switches with two different leading companies in SDN and SDWAN technology. For the internal connectivity we have used HP SDN enabled switches with two different series model and for the external connectivity to the Internet and remote site for the sake of future growth, we have used Cisco Meraki SDWAN with some interesting features which will be very useful in order to control the flow and some traffic shaping. As we have mentioned, we used three-tier hierarchy to manage, troubleshoot, and maintain the network more efficiently. At the same time, because our network is SDN based, we have automatically three-tier architecture. The first tier is physical infrastructure, which includes all hardware devices such as HP switches and cabling required to establish connectivity. The second tier is our HPE VAN SDN Controller software which

provides a unified control point in an OpenFlow-enabled network. The third tier is an application layer which directs specific functions through the controller.

Physical Hardware Specifications

For the first tier of our architecture, we have chosen the HP 3800 series. This 3800-switch series is a fully managed Gigabit Ethernet switches available in both 24 and 48 ports. The key features of this switch are as follow:

- 1- Open Flow
- 2- Layer 3 stackable
- 3- Low-Latency
- 4- Offer 10GbE
- 5- Supports 802.1Q, 8.2.1v, 802.3ad, RPVST+ and much more
- 6- RIP, OSPF, BGP, and static route

In our design, we used 3800 for all three floors in the first tier of the architecture, which is commonly called access layer as all these switches are connected to the workstations. The reason why we used these 3800 series is that it supports open flow, which means we can use HP VAN Controller to manage all these switches in a centralized manner. Furthermore, it supports layer four optimization, which means the controller can manage this switch through layer 1 to 4. It has many features such as QoS, bandwidth shaping, and security which bring many benefits into our design.



For the second tier of our design, we have used the HP 5500 EI switch series, which is combined of layer 2 and three features. The reason why we have chosen these series 5500 EI is because of its high availability feature which provides us with fast switching mechanism that is ideal for the second tier of our design which is usually called as a distribution layer. These switches provide both IPv4 and IPv6 protocol. It also provides traffic prioritization to enhance applications.

The key features of these series switches are as follow:

- 1- Open Flow
- 2- QoS
- 3- SNMP, sFlow, LLDP and VLAN
- 4- 10 GbE
- 5- MSTP, RSTP, VRRP, and IGMP
- 6- ARP, DHCP, UDP
- 7- IPv4 and IPv6 Routing
- 8- Security such as RADIUS, BPDU Guard, STP root Guard, etc

For our server room, we used the same HP 550 EI series as well because it provides fast switching features. This switch provides reliability and multi-service support capabilities for robust switching at the edge or aggregation layer of enterprises.



HPE VAN Controller

The HPE VAN Controller provides an extensive platform for developing and running SDN applications. It provides a unified control point in an open flow enabled network. It is a software-based controller which simplifies management, provisioning, and orchestration. It provides APIs to dynamically link business requirements to network infrastructure via either java programming or RESTful control interfaces. It is quite fit for a campus network and data centers. The key features of this controller are as follow:

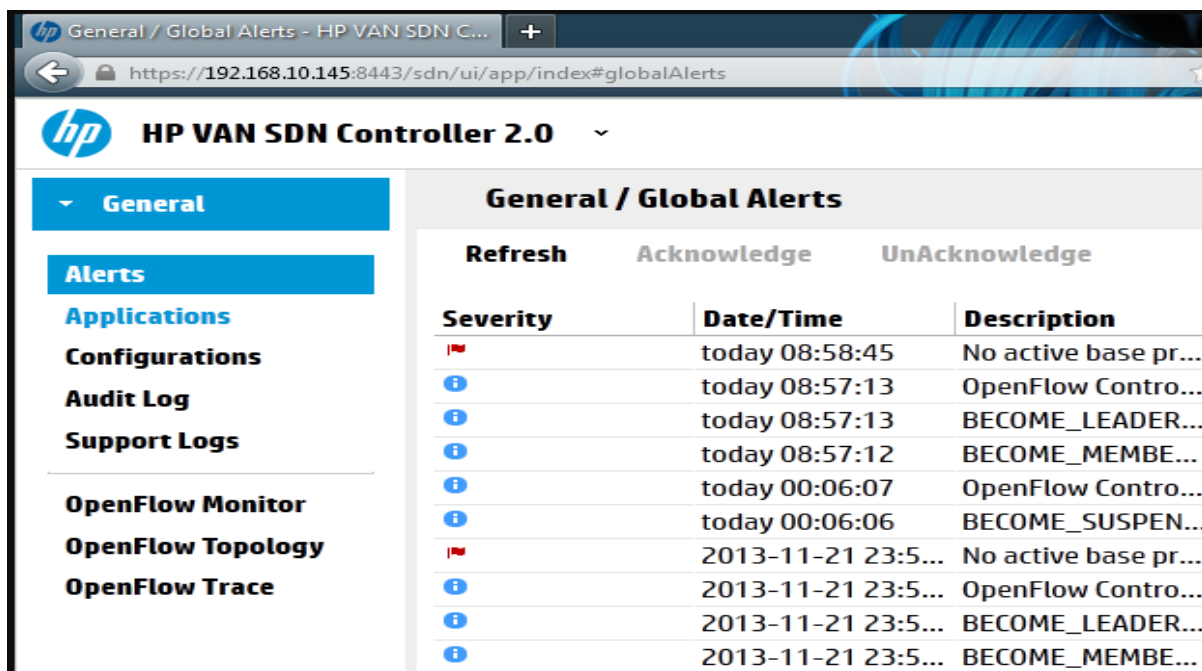
- 1- A base controller platform for Linux/Java and open flow 1.0/1.3.
- 2- Operation with either a local or remote keystone server
- 3- Embedded Java applications, extensible Java RESTful API, and HPE Intelligent Management Centre.
- 4- Proactive and Reactive flow processing
- 5- Northbound APIs: RESTful https Interface
- 6- Delivers security at multiple levels
- 7- Utilizes LLDP messages to discover physical links between switches in the control domain

8- Creates a network graph based on information from the link service

After installing HPE VAN SDN Controller, we can log in to the controller with the default provided username and password.



Once you log in, the basic dashboard view is very simple.



Internet and Remote Site Connectivity

As we have mentioned in the introduction, we used a hybrid model, which means we have made use of both hardware and software to design this network. We used Cisco Meraki MX 64, which provides 100% cloud managed security and SDWAN. The reason we have used MX series in our design is that it provides many security features as well as SDWAN solution. The price is quite reasonable, and it is easy to learn to work with its friendly dashboard. The interesting feature about these MX series is that you control the WAN link via the cloud which means once you login to the dashboard, you are in the Meraki cloud and the controller is based in the cloud and make it easy to control all the devices from the cloud with the help of a cloud-based controller.



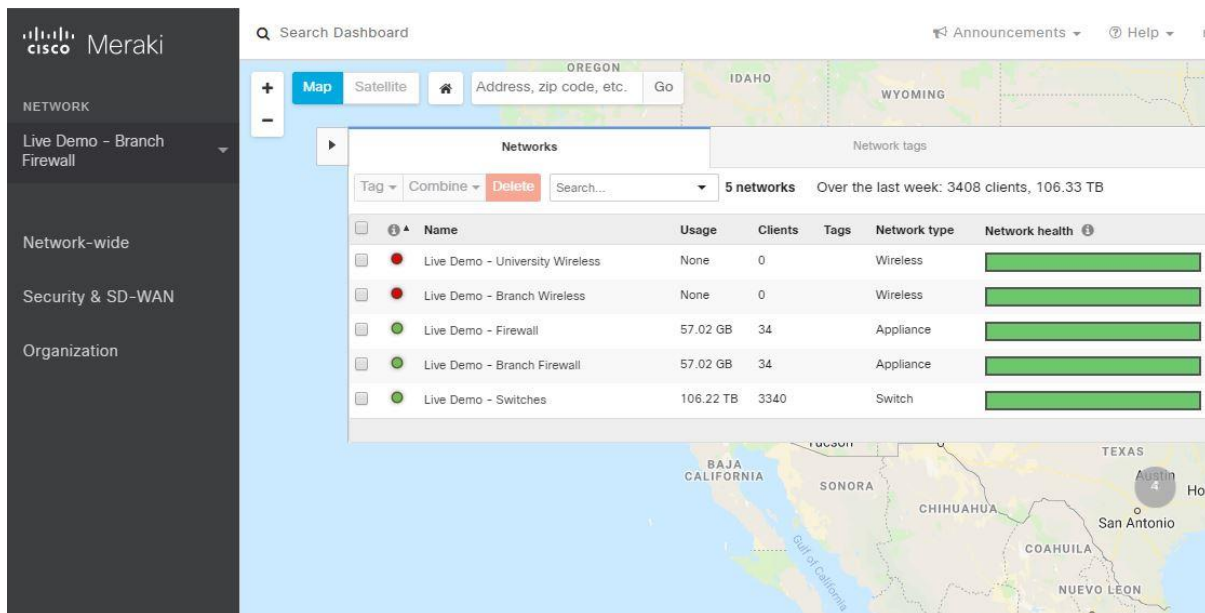
Some of the key features of the MX series are as follow:

- 1- Stateful Firewall
- 2- Cloud-based centralized management
- 3- Manage centrally over the Web
- 4- Site-to-site VPN

- 5- Identity-based Policy
- 6- Client VPN (IPsec)
- 7- Traffic shaping and Application Management
- 8- IDS/IPS
- 9- Advanced Malware Protection (AMP)
- 10-Content Filtering

Dashboard Overview

In this section, we are going through the dashboard and its capabilities.



The above image is the whole view of the dashboard. Once you login to the dashboard, you can observe your network(s) and their locations in the map. This window provides you with some information such as your type network types, usage, number of clients, and network health.

If you click on Security & SDWAN, you are prompted to the SDWAN and traffic shaping:

The screenshot shows the Cisco Meraki dashboard interface. On the left is a dark sidebar with the Meraki logo and navigation menu items: NETWORK, Live Demo - Branch Firewall, Network-wide, Security & SD-WAN (highlighted), and Organization. The main content area is titled 'SD-WAN & traffic shaping'. Below this is the 'Uplink configuration' section, which lists three uplinks: WAN 1 (500 Mbps), WAN 2 (500 Mbps), and Cellular (unlimited). Each uplink has a 'details' link. Below the uplink configuration is the 'Uplink statistics' section, which includes a table for testing connectivity. The table has columns for 'Test Connectivity to', 'Description', 'Default', and 'Actions'. The first row shows a test to '8.8.8.8' with the description 'Google', which is the default and has a 'Test' button. There is also a link to 'Add a destination'.

In the above image, you can configure uplink and its bandwidth. As you can see, we can choose the bandwidth between WAN 1 and WAN 2 and choose one of them as the primary up link with more bandwidth and the other one as the alternative link with lower bandwidth. We can test the statuses of the link by entering the destination IP address and see the results.

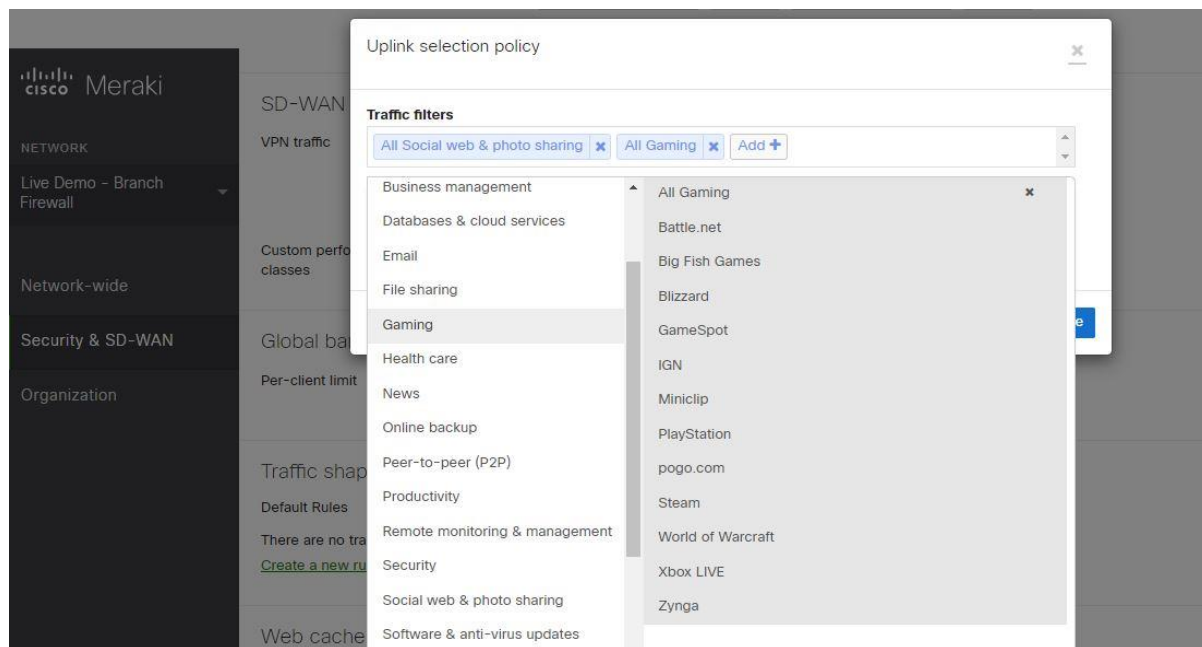
We can load balance between links as we have mentioned and activate VPN tunnel over all the available uplinks:

The screenshot shows the 'Uplink selection' configuration page in the Cisco Meraki dashboard. The sidebar is the same as in the previous image. The main content area is titled 'Uplink selection' and contains 'Global preferences'. Under 'Global preferences', there are three settings: 'Primary uplink' (set to WAN 1), 'Load balancing' (set to Enabled), and 'Active-Active AutoVPN' (set to Enabled). The 'Load balancing' section has a description: 'Traffic will be spread across both uplinks in the proportions specified above. Management traffic to the Meraki cloud will use the primary uplink.' The 'Active-Active AutoVPN' section has a description: 'Create VPN tunnels over all of the available uplinks (primary and secondary).'

We can set up flow preference by choosing specific protocols, source IP, destination IP, source port, destination port, preferred uplinks, and associated actions:

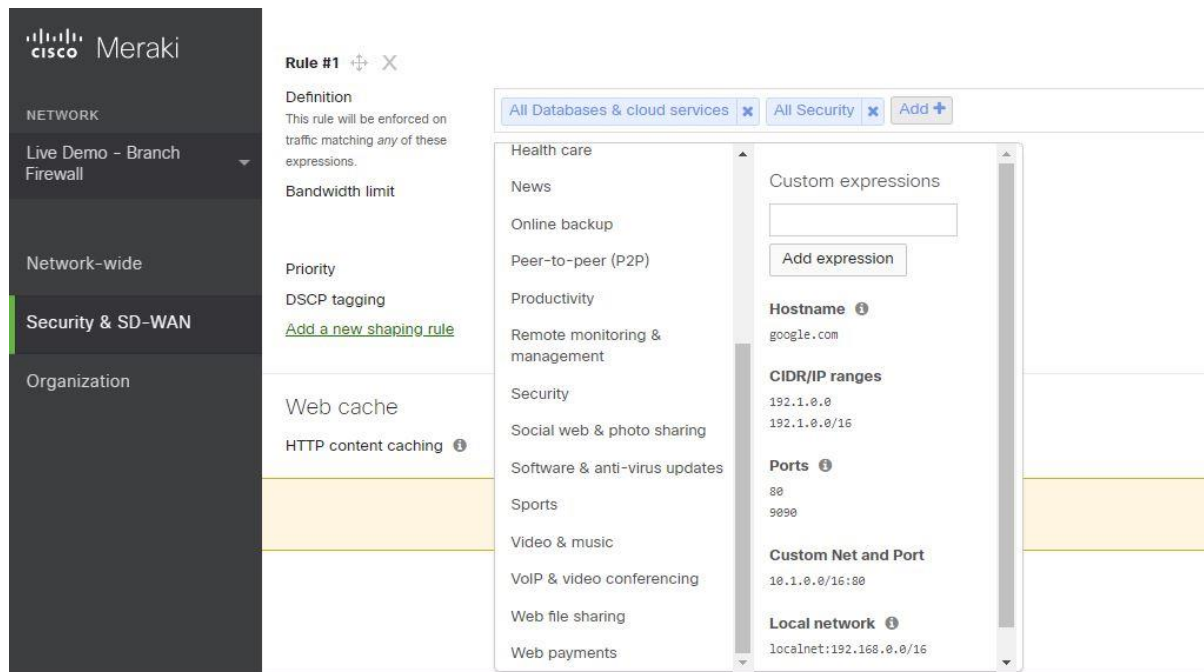


As per the requirements of our design, we can set some specific policies to prevent some access to social media and gaming:

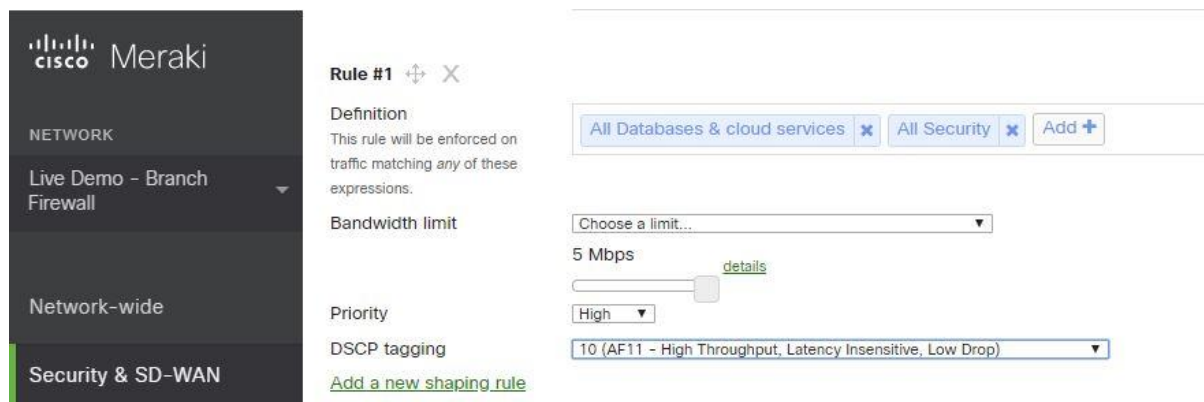


As you can see from the above window, we can filter some certain traffics such as social web and gaming to reduce distraction in an enterprise to speed up productivity.

With MX series, we can do traffic shaping and set some rules to prioritize specific traffic:



In the above window, we choose our certain types of traffic that we want to prioritize.



And then we choose the bandwidth limit by setting up its priority.

As you have seen, the dashboard is easy to learn and quite user-friendly, which helps to speed up the process of implementation as well as educating stuff to learn how to work with this dashboard.

To recap, with this hybrid design, we can easily scale the network for future growth with proper redundancy in case of failover. Both HPE VAN SDN Controller and Cisco Meraki MX series provide centrally managing network

through layer 1 to layer four as well as security features which help us to protect the network.

REFERENCES

- CISCO APPLIANCES 2019, Sydney, viewed 29 May 2019
<<https://meraki.cisco.com/products/appliances/mx64>>
- HP support 2019, Sydney, viewed 29 May 2019
<https://support.hpe.com/hpsc/doc/public/display?docId=emr_na-c03967699>
- IT in stock 2019, *e-commerce website*, Sydney, viewed 29 May 2019,
<<https://www.itinstock.com/hp-procurve-3800-24g-2sfp-2-x-10gbe-l3-switch-j9575a-2x-psu-j9581a-j9575aaba-42903-p.asp>>
- Network geek Stuff 2019, US, Sydney, viewed 29 May 2019
<<https://networkgeekstuff.com/networking/playing-with-the-new-hp-sdn-controller-including-getting-started-guide-with-open-vswitch-in-gns3/>>
- Nautil Communications 2019, Sydney, viewed 29 May 2019
<<https://www.nautilusnet.com/products/jd374a-hpe-5500-24g-sfp-ei-switch-with-2-interface-slots-managed-l3.html>>
- UTS: Engineering 42027, Software Defined Networks, Lecture slides, Faculty of engineering & information technology, Sydney
- UTS: Engineering 42027, Software Defined Networks, Lab handouts, Faculty of engineering & information technology, Sydney

APPENDIX 1: SDN_Project_Topology.py

MININET TOPOLOGY:

```
#!/usr/bin/python
import sys
from functools import partial
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.topo import SingleSwitchTopo
from mininet.log import setLogLevel, info
from mininet.node import Host
from mininet.topo import Topo
from mininet.util import quietRun
from mininet.log import error
from mininet.node import OVSSwitch, Controller, RemoteController
from mininet.node import Controller, OVSKernelSwitch, RemoteController
import json
from mininet.moduledeps import pathCheck
import ipaddress

from sys import exit
import os.path
from subprocess import Popen, STDOUT, PIPE

class VLANHost(Host):
    # Host connected to Vlan interface
    def config(self, vlan=100, **params):
        r = super(VLANHost, self).config(**params)

        intf = self.defaultIntf()
        # remove IP from default "physical" interface
        self.cmd('ifconfig %s inet 0' % intf)
        # create VLAN interface
        self.cmd('vconfig add %s %d' % (intf, vlan))
        # assign the host's IP to the VLAN interface
        self.cmd('ifconfig %s.%d inet %s' % (intf, vlan, params['ip']))
        # update the intf name and host's intf map
        newName = '%s.%d' % (intf, vlan)
```



```

    # update the (Mininet) interface to refer to VLAN interface name
    intf.name = newName
    # add VLAN interface to host's name to intf map
    self.nameToIntf[newName] = intf

    return r

```

```
hosts = {'vlan': VLANHost}
```

```
def starthttp( host ):
```

```
    "Start simple Python web server on hosts"
```

```
    info( '*** Starting SimpleHTTPServer on host', host, '\n' )
```

```
    host.cmd( 'cd ./http_%s/; nohup python2.7 ./webserver.py &' % (host.name) )
```

```
def stophttp():
```

```
    "Stop simple Python web servers"
```

```
    info( '*** Shutting down stale SimpleHTTPServers',
```

```
        quietRun( "pkill -9 -f SimpleHTTPServer" ), '\n' )
```

```
    info( '*** Shutting down stale webserver',
```

```
        quietRun( "pkill -9 -f webserver.py" ), '\n' )
```

```
def VLAN_Mininet():
```

```
    net = Mininet(controller=partial(RemoteController))
```

```
    c1 = net.addController( 'c1', port=6633, ip='127.0.0.1' )
```

```
    k=9 #Total number of VLANS
```

```
    n=112
```

```
    #Floor 1 VLANS
```

```
    vlanBase1=10
```

```
    vlanBase2=20
```

```
    vlanBase3=25
```

```
    vlanBase4=5
```

```
#Floor 2 VLANs
vlanBase5=30
vlanBase6=5
#Floor 3 VLANs
vlanBase7=40
vlanBase8=50
vlanBase9=5
#Common Server VLANs
vlanBase10=60
# Dedicated Server VLANs
vlanBase11=70
info('*****add switches\n')
s1 = net.addSwitch('s1', ip='172.16.1.1/24')
s1.cmd('sysctl -w net.ipv4.ip_forward=1')
s2 = net.addSwitch('s2', ip='172.16.4.1/24')
s2.cmd('sysctl -w net.ipv4.ip_forward=1')
s3 = net.addSwitch('s3', ip='172.16.6.1/24')
s3.cmd('sysctl -w net.ipv4.ip_forward=1')
s4 = net.addSwitch('s4', )
s4.cmd('sysctl -w net.ipv4.ip_forward=1')
s5 = net.addSwitch('s5', )
s5.cmd('sysctl -w net.ipv4.ip_forward=1')
s6 = net.addSwitch('s6', )
s6.cmd('sysctl -w net.ipv4.ip_forward=1')
s7 = net.addSwitch('s7', )
s7.cmd('sysctl -w net.ipv4.ip_forward=1')
vlan1 = vlanBase1
vlan2 = vlanBase2
vlan3 = vlanBase3
vlan4 = vlanBase4
vlan5 = vlanBase5
vlan6 = vlanBase6
vlan7 = vlanBase7
```

```

vlan8 = vlanBase8
vlan9 = vlanBase9
vlan10 = vlanBase10
vlan11 = vlanBase11
info('***Adding hosts of first floor (hf) and linking them to Switch 1\n')
j=1
] for j in range(n):
]   if (j>0 and j <= 7):
       name = 'hf%d' % (j)
       h = net.addHost(name, cls=VLANHost, vlan=vlan1, mac="01:00:00:00:00:%d" % (j), ip='172.16.1.%d/24' % (j + 1), defaultRoute='via 172.16.10.%d' % (j + 1))
       net.addLink(h, s1)
       #s1.cmdPrint("ifconfig s1-eth%d 172.16.1.1 netmask 255.255.255.0" % (j)) #info('****adding IP addresses to the interfaces of the L3 switch 1')
       #a = s1.cmdPrint("ifconfig s1-eth%d" % (j))
       ip = "172.16.10.%d" % (j+1)
       s1.cmdPrint("ifconfig s1-eth%d %s netmask 255.255.255.0" % (j,ip) )

       .
]   elif (j >= 8 and j < 49):
       name = 'hf%d' % (j)
       h = net.addHost(name, cls=VLANHost, vlan=vlan2, mac="01:00:00:00:00:%d" % (j), ip='172.16.2.%d/24' % (j - 6), defaultRoute='via 172.16.20.%d' % (j - 6))
       net.addLink(h, s1)
       #s1.cmdPrint("ifconfig s1-eth%d 172.16.2.1 netmask 255.255.255.0" % (j))
       ip = "172.16.20.%d" % (j-6)
       s1.cmdPrint("ifconfig s1-eth%d %s netmask 255.255.255.0" % (j,ip) )

       .
]   elif (j == 49):
       name = 'hf%d' % (j)
       h = net.addHost(name, cls=VLANHost, vlan=vlan3, mac="01:00:00:00:00:%d" % (j), ip='172.16.3.%d/24' % (j - 47), defaultRoute='via 172.16.30.%d' % (j - 47))
       net.addLink(h, s1)
       #s1.cmdPrint("ifconfig s1-eth%d 172.16.3.1 netmask 255.255.255.0" % (j))
       ip = "172.16.30.%d" % (j-47)
       s1.cmdPrint("ifconfig s1-eth%d %s netmask 255.255.255.0" % (j,ip) )

       .
]

```

```

elif (j == 50):
    name = 'hf%d' % (j)
    h = net.addHost(name, cls=VLANHost, vlan=vlan4, mac="01:00:00:00:00:00" % (j), ip='172.16.5.%d/24' % (j - 48), defaultRoute='via 172.16.50.%d' % (j - 48))
    net.addLink(h, s1)
    #s1.cmdPrint("ifconfig s1-eth%d 172.16.5.1 netmask 255.255.255.0" % (j))
    ip = "172.16.50.%d" % (j-48)
    s1.cmdPrint("ifconfig s1-eth%d %s netmask 255.255.255.0" % (j,ip) )
info("***Adding hosts of second floor (hs) and linking them to Switch 2\n")
j=1

```

```

for j in range(n):
    if (j>0 and j <= 10):
        name = 'hs%d' % (j)
        h = net.addHost(name, cls=VLANHost, vlan=vlan5, mac="02:00:00:00:00:00" % (j), ip='172.16.4.%d/24' % (j + 1), defaultRoute='via 172.16.40.%d' % (j + 1))
        net.addLink(h, s2)
        #s2.cmdPrint("ifconfig s2-eth%d 172.16.4.1 netmask 255.255.255.0" % (j))
        ip = "172.16.40.%d" % (j+1)
        s2.cmdPrint("ifconfig s2-eth%d %s netmask 255.255.255.0" % (j,ip) )
    elif (j == 11):
        name = 'hs%d' % (j)
        h = net.addHost(name, cls=VLANHost, vlan=vlan6, mac="02:00:00:00:00:00" % (j), ip='172.16.5.%d/24' % (j - 9), defaultRoute='via 172.16.50.%d' % (j - 9))
        net.addLink(h, s2)
        #s2.cmdPrint("ifconfig s2-eth%d 172.16.5.1 netmask 255.255.255.0" % (j))
        ip = "172.16.50.%d" % (j-9)
        s2.cmdPrint("ifconfig s2-eth%d %s netmask 255.255.255.0" % (j,ip) )
info("***Adding hosts of third floor (ht) and linking them to switch 3\n")
j=1

```

```

for j in range(n):
    if (j>0 and j <= 30):
        name = 'ht%d' % (j)
        h = net.addHost(name, cls=VLANHost, vlan=vlan7, mac="03:00:00:00:00:0%d" % (j), ip='172.16.6.%d/24' % (j + 1), defaultRoute='via 172.16.60.%d' % (j + 1))
        net.addLink(h, s3)
        #s3.cmdPrint("ifconfig s3-eth%d 172.16.6.1 netmask 255.255.255.0" % (j))
        ip = "172.16.60.%d" % (j+1)
        s3.cmdPrint("ifconfig s3-eth%d %s netmask 255.255.255.0" % (j,ip) )
    elif (j >= 31 and j <= 35):
        name = 'ht%d' % (j)
        h = net.addHost(name, cls=VLANHost, vlan=vlan8, mac="03:00:00:00:00:0%d" % (j), ip='172.16.7.%d/24' % (j + 1), defaultRoute='via 172.16.70.%d' % (j + 1))
        net.addLink(h, s3)
        ip = "172.16.70.%d" % (j+1)
        s3.cmdPrint("ifconfig s3-eth%d %s netmask 255.255.255.0" % (j,ip) )
        #s3.cmdPrint("ifconfig s3-eth%d 172.16.7.1 netmask 255.255.255.0" % (j))
    elif (j == 36):
        name = 'ht%d' % (j)
        h = net.addHost(name, cls=VLANHost, vlan=vlan9, mac="03:00:00:00:00:0%d" % (j), ip='172.16.5.%d/24' % (j - 34), defaultRoute='via 172.16.50.%d' % (j - 34))
        net.addLink(h, s3)
        ip = "172.16.50.%d" % (j-34)
        s3.cmdPrint("ifconfig s3-eth%d %s netmask 255.255.255.0" % (j,ip) )
        #s3.cmdPrint("ifconfig s3-eth%d 172.16.5.1 netmask 255.255.255.0" % (j))
info('***Adding common, dedicated servers and linking them to switch 6\n')
j = 1
for j in range(n):
    if (j>0 and j <= 10):
        name = 'server%d' % (j)
        h = net.addHost(name, cls=VLANHost, vlan=vlan10, mac="11:00:00:00:00:0%d" % (j), ip='172.16.8.%d/24' % (j + 1), defaultRoute='via 172.16.80.%d' % (j + 1))
        net.addLink(h, s6)
        #s6.cmdPrint("ifconfig s6-eth%d 172.16.80.1 netmask 255.255.255.0" % (j+1))
        ip = "172.16.80.%d" % (j+1)
        s6.cmdPrint("ifconfig s6-eth%d %s netmask 255.255.255.0" % (j,ip) )
        #starthttp(h)

```

```

elif (j >= 11 and j <= 15):
    name = 'dserver%d' % (j)
    h = net.addHost(name, cls=VLANHost, vlan=vlan11, mac="11:00:00:00:00:0d" % (j), ip='172.16.9.%d/24' % (j - 9), defaultRoute='via 172.16.90.%d' % (j - 9))
    net.addLink(h, s6)
    #s6.cmdPrint("ifconfig s6-eth%d 172.16.90.1 netmask 255.255.255.0" % (j+1))
    ip = "172.16.50.%d" % (j-9)
    s6.cmdPrint("ifconfig s6-eth%d %s netmask 255.255.255.0" % (j,ip) )
    #starthttp(h)

# Adding links between switches
info('***Adding links between switches\n')
net.addLink(s1, s4)
net.addLink(s1, s5)
net.addLink(s2, s4)
net.addLink(s2, s5)
net.addLink(s3, s4)
net.addLink(s4, s5)
net.addLink(s6, s5)
net.addLink(s6, s4)
net.addLink(s7, s4)
net.addLink(s7, s5)

#Starting the servers
#starthttp(server1)
#starthttp(dserver1)

#Configuring flows in the S1
#net.cmdPrint("sh ovs-ofctl add-flow s1 in_port=5,dl_vlan=10,action=strip_vlan,output=1")

# Connecting Switch 7 to the internet
#s7.cmdPrint("ifconfig s7-eth3 o")
#s7.cmdPrint("dhclient s7-eth3")

```

```
c1.start()
s1.start( [c1] )
s2.start( [c1] )
s3.start( [c1] )
s4.start( [c1] )
s5.start( [c1] )
s6.start( [c1] )
s7.start( [c1] )

info('***starting network\n')
net.build()
info('****Starting switches\n')
info('****post configure switches and hosts\n')
CLI(net)
stophttp()
net.stop()

if __name__ == '__main__':
    setLogLevel('info')

    if not quietRun('which vconfig'):
        error("Cannot find command 'vconfig'\nThe package",
            "'vlan' is required in Ubuntu or Debian,",
            "or 'vconfig' in Fedora\n")
        exit()
    else:
        VLAN_Mininet()
```

APPENDIX 2: Web Server.py and Index.html

Webserver.py

```
import SimpleHTTPServer
import SocketServer

class MininetHttpRequestHandler(SimpleHTTPServer.SimpleHTTPRequestHandler):
    # Disable logging DNS lookups
    def address_string(self):
        return str(self.client_address[0])

PORT = 80

Handler = MininetHttpRequestHandler
httpd = SocketServer.TCPServer(("", PORT), Handler)
print "Server1: httpd serving at port", PORT
httpd.serve_forever()
```

Index.html

```
<html>
  <head><title> This is Server1 </title></head>
  <body>
    Congratulations! <br/>
    Your router successfully route your packets to server1. <br/>
  </body>
</html>
```


APPENDIX 3: VlanSwitchSDN.py

```

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import vlan
from ryu.lib.packet import ether_types

#GLOBAL VARIABLES
port_vlan = {2:{3:[" "],1:[30],2:[20],4:[20,30]},3:{4:[20,30],1:[30],2:[20],3:[" " ]}} #port_vlan[a][b]=c => 'a'= dpid, 'b'= port number, 'c'= VLAN ID

access= {2:[1,2],3:[1,2]} #access[a]=[B] => 'a' = dpid, '[B]'=List of ports configured as Access Ports

trunk = {2:[4],3:[4]} #trunk[a]=[B] => 'a' = dpid, '[B]'=List of ports configured as Trunk Ports

class VlanSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(VlanSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # install table-miss flow entry
        #
        # We specify NO BUFFER to max_len of the output action due to

```

```

# We specify NO BUFFER to max_len of the output action due to
# OVS bug. At this moment, if we specify a lesser number, e.g.,
# 128, OVS will send Packet-In with invalid buffer_id and
# truncated packet data. In that case, we cannot output packets
# correctly. The bug has been fixed in OVS v2.1.0.
match = parser.OFPMatch()
actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                  ofproto.OFPCML_NO_BUFFER)]
self.add_flow(datapath, 0, match, actions)

def add_flow(self, datapath, priority, match, actions, buffer_id=None):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    inst = [parser.OFPInstructionActions(ofproto.OFPI_APPLY_ACTIONS,
                                         actions)]
    if buffer_id:
        mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                priority=priority, match=match,
                                instructions=inst)
    else:
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                match=match, instructions=inst)
    datapath.send_msg(mod)

def vlan_members(self, dpid, in_port, src_vlan):
    B=[]
    self.access_ports = []
    self.trunk_ports = []

```

```

if src_vlan == "NULL":
    return

for item in port_vlan[dpid]:
    vlans=port_vlan[dpid][item]
    if src_vlan in vlans and item!=in_port:
        B.append(item)

for port in B:
    if port in access[dpid]:
        self.access_ports.append(port)
    else:
        self.trunk_ports.append(port)

#-----#

def getActionsArrayTrunk(self,out_port_access,out_port_trunk,parser):
    actions= [ ]

    for port in out_port_trunk:
        actions.append(parser.OFPActionOutput(port))

    actions.append(parser.OFPActionPopVlan())

    for port in out_port_access:
        actions.append(parser.OFPActionOutput(port))

    return actions

```

```
def getActionsArrayAccess(self,out_port_access,out_port_trunk,src_vlan, parser):
    actions= [ ]

    for port in out_port_access:
        actions.append(parser.OFPACTIONOutput(port))

    actions.append(parser.OFPACTIONPushVlan(33024))
    actions.append(parser.OFPACTIONSetField(vlan_vid=src_vlan))

    for port in out_port_trunk:
        actions.append(parser.OFPACTIONOutput(port))

    return actions

def getActionsNormalUntagged(self,dpid,in_port,parser):
    actions= [ ]

    for port in port_vlan[dpid]:
        if port_vlan[dpid][port][0]==" " and port!=in_port:
            actions.append(parser.OFPACTIONOutput(port))

    if dpid in trunk:

        for port in trunk[dpid]:
            if port!=in_port:
                actions.append(parser.OFPACTIONOutput(port))

    return actions
```

```

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    # If you hit this you might want to increase
    # the "miss_send_length" of your switch
    if ev.msg.msg_len < ev.msg.total_len:
        self.logger.debug("packet truncated: only %s of %s bytes",
                           ev.msg.msg_len, ev.msg.total_len)
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']

    #SWITCH ID
    dpid = datapath.id

    pkt = packet.Packet(msg.data)
    eth = pkt.get_protocols(ethernet.ethernet)[0]
    vlan_header = pkt.get_protocols(vlan.vlan) #If packet is tagged, then this will have non-null value

    if eth.ethertype == ether_types.ETH_TYPE_8021Q :    #Checking for VLAN Tagged Packet
        vlan_header_present = 1
        src_vlan=vlan_header[0].vid
    elif dpid not in port_vlan:
        vlan_header_present = 0
        in_port_type = "NORMAL SWITCH "          #NORMAL NON-VLAN L2 SWITCH
        src_vlan = "NULL"

```

```

elif port_vlan[dpid][in_port][0]== " " or in_port in trunk[dpid]:
    vlan_header_present = 0
    in_port_type = "NORMAL UNTAGGED"          #NATIVE VLAN PACKET
    src_vlan = "NULL"
else:
    vlan_header_present = 0
    src_vlan=port_vlan[dpid][in_port][0]      # STORE VLAN ASSOCIATION FOR THE IN PORT

if eth.ethertype == ether_types.ETH_TYPE_LLDP:
    # ignore lldp packet
    return

```

```

dst = eth.dst
src = eth.src

```

```

#CREATE NEW DICTIONARY ENTRY IF IT DOES NOT EXIST

```

```

self.mac_to_port.setdefault(dpid, {})

```

```

self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

```

```

# learn a mac address to avoid FLOOD next time.

```

```

self.mac_to_port[dpid][src] = in_port

```

```

#Determine which ports are members of in_port's VLAN

```

```

self.vlan_members(dpid,in_port,src_vlan)

```

```

out_port_type = " "

```

```

if dst in self.mac_to_port[dpid]:          #MAC ADDRESS TABLE CREATION
    out_port_unknown = 0
    out_port = self.mac_to_port[dpid][dst]
    if src_vlan!="NULL":
        if out_port in access[dpid]:
            out_port_type = "ACCESS"
        else:
            out_port_type = "TRUNK"
        else:
            out_port_type = "NORMAL"
    else:
        out_port_unknown = 1
        out_port_access = self.access_ports    #List of All Access Ports Which are Members of Same VLAN (to Flood the Traffic)
        out_port_trunk = self.trunk_ports      #List of All Trunk Ports Which are Members of Same VLAN (to Flood the Traffic)

```

```

##### -----FLOW ENTRY ADDITION SEGMENT -----#####

```

```

if out_port_unknown!=1:                    # IF OUT PORT IS KNOWN
    if vlan_header_present and out_port_type == "ACCESS" :          #If VLAN Tagged and needs to be sent out through ACCESS port
        match = parser.OFPMatch(in_port=in_port, eth_dst=dst, vlan_vid=(0x1000 | src_vlan))
        actions = [parser.OFPACTIONPopVlan(), parser.OFPACTIONOutput(out_port)] # STRIP VLAN TAG and SEND TO OUTPUT PORT
    elif vlan_header_present and out_port_type == "TRUNK" :
        match = parser.OFPMatch(in_port=in_port, eth_dst=dst, vlan_vid=(0x1000 | src_vlan))
        actions = [parser.OFPACTIONOutput(out_port)]                #SEND THROUGH TRUNK PORT AS IS
    elif vlan_header_present!=1 and out_port_type == "TRUNK" :
        match = parser.OFPMatch(in_port=in_port, eth_dst=dst)

```

```

    data = None
    if msg.buffer_id == ofproto.OFP_NO_BUFFER:
        data = msg.data

    out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                              in_port=in_port, actions=actions, data=data)
    datapath.send_msg(out)

```

APPENDIX 4: L2L3Switch.py

```

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet, arp, ipv4
from ryu.lib.packet import ether_types
import ryu.lib.dpid
import ryu.lib

class L2L3Switch(app_manager.RyuApp): #creating a simple switch as a Ryu App
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(L2L3Switch, self).__init__(*args, **kwargs)
        # simpleL2switch is a child of Ryu App.
        self.mac_to_port = {}
        #[port1:[mac1.ip1].port2:[mac2.ip2]....]
        #to store the details of switch and connected hosts IP and mac address

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, event): #this method handles switch feature requests and we install initial flow to forward all the packets to the controller incase of a table miss.
        print("**** in feature handler **** ")
        # send to the controller
        sendto_controller(self, event)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, event):
        pkt = packet.Packet(data=event.msg.data)
        # creating a packet with msg's data as payload
        eth = pkt.get_protocols(ethernet.ethernet)[0]
        # fetching ethernet dataframe
        if eth.ethertype == ether_types.ETH_TYPE_ARP:
            #handling ARP requests

```

```

        handle_ARP(self, event)
        # ARP packet is ethernet dataframe's payload. Ethernet frame has type from as ETH_TYPE_ARP, if true then handle ARP
        # call method to handle ARP packets
    elif eth.ethertype == ether_types.ETH_TYPE_IP:
        handle_IP(self, event)
        #handle IP packet.
        #call method to handle IP packet

def handle_ARP(self, event):
    # handle ARP packets
    datapath = event.msg.datapath
    # datapath connection
    ofproto = datapath.ofproto
    #ofproto of the datapath
    in_port = event.msg.match['in_port']
    # port through which the switch recieved this packet
    parser = datapath.ofproto_parser
    pkt = packet.Packet(data=event.msg.data)
    eth = pkt.get_protocols(ethernet.ethernet)[0]
    # fetching ethernet dataframe
    arp_pkt = pkt.get_protocol(arp.arp)
    # Fetching ARP dataframe
    self.mac_to_port[in_port] = [arp_pkt.src_mac, arp_pkt.src_ip]
    # MAC, IP storing
    out_port = self.check_mactable(ofproto, 'arp', arp_pkt.dst_mac)
    # Check MAC table for destination IP
    actions = [parser.OFPACTIONOutput(out_port)]
    # updating action lists
    match = self.simplermatch(parser, eth.src, eth.dst, in_port)
    # Gives Match field
    self.add_flow(datapath, 1, match, actions, buffer_id=None)
    # adding flow in the table

```



```

def handle_IP(self,event):
    # handle IP packets
    datapath = event.msg.datapath
    # datapath connection
    ofproto = datapath.ofproto
    # ofproto of the datapath
    in_port = event.msg.match['in_port']
    # port through which the switch recieved this packet
    parser = datapath.ofproto_parser
    pkt = packet.Packet(data=event.msg.data)
    eth = pkt.get_protocols(ethernet.ethernet)[0]
    # Fetching ethernet dataframe
    ip_pkt = pkt.get_protocol(ipv4.ipv4)
    # fetching IP packet
    out_port = self.check_mactable(ofproto,'ip',ip_pkt.dst)
    # check IP destination in the Table
    match = self.simplematch(parser,eth.src,eth.dst,in_port)
    # .
    actions = [parser.OFPActionOutput(port=out_port)]
    #
    if event.msg.buffer_id != ofproto.OFP_NO_BUFFER:
        self.add_flow(datapath, 1, match, actions, event.msg.buffer_id)
        # adding a flow in case of no buffer
    else:
        self.add_flow(datapath, 1, match, actions)

def check_mactable(self,ofproto,caller,para):
    # to check if an mac addr or IP addr exists in mac table
    if caller == 'arp':
        # if the calling function is arp, then check mac address
        for p in self.mac_to_port:
            if self.mac_to_port[p][0] == para:
                # [p][0]
                return p
        # return p as output # if found return

```

```

    elif caller == 'ip':
        #if calling function is ip , then check ip addr
        for p in self.mac_to_port:
            if self.mac_to_port[p][1] == para:
                #[p][1]
                return p
            # return corresponding port
        return ofproto.OFPP_FLOOD
    # if no port is found

def sendto_controller(self,event):
    # initial installation of table miss flow
    datapath = event.msg.datapath
    #.
    ofproto = datapath.ofproto
    #.
    parser = datapath.ofproto_parser
    #.
    match = parser.OFPMatch()
    #.
    actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,ofproto.OFPCML_NO_BUFFER)]
    #send to
    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]
    mod = parser.OFPFlowMod(datapath=datapath,priority=0,match=match,instructions=inst)
    datapath.send_msg(mod)

def add_flow(self, datapath, priority, match, actions, buffer_id=None):
    #
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    idle_timeout=45
    # idle-timeout set to 45 seconds

```

```

hard_timeout=45
inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]
#forming instructions
if buffer_id:
    mod = parser.OFPFlowMod(datapath=datapath,buffer_id=buffer_id,priority=priority,idle_timeout=idle_timeout,hard_timeout = hard_timeout, match=match,instructions=inst)
else:
    mod = parser.OFPFlowMod(datapath=datapath,priority=priority,match=match,idle_timeout=idle_timeout, hard_timeout = hard_timeout,instructions=inst)
self.logger.info("added flow for %s",mod)
datapath.send_msg(mod)

#### request the packet to be forwarded onto a specific port from the switch ####
def switchport_out(self,pkt,datapath,port):
    #.'accept raw data , serialise it and packetout from a OF switch ''' #
    ofproto = datapath.ofproto
    #.
    parser = datapath.ofproto_parser
    #.
    pkt.serialize()
    #. serialise packet (ie convert raw data)
    self.logger.info("packet-out %s" %(pkt,))
    #.
    data = pkt.data
    #.
    actions = [parser.OFPActionOutput(port=port)]
    #.
    out = parser.OFPPacketOut(datapath = datapath,buffer_id=ofproto.OFP_NO_BUFFER,in_port=ofproto.OFPP_CONTROLLER,actions=actions,data=data)
    #.
    datapath.send_msg(out)
    #.

def simplematch(self,parser,src,dst,in_port):
    match = parser.OFPMatch(in_port=in_port,eth_dst=dst,eth_src=src)
    #
    return match

```

APPENDIX 5: rest_firewall.py

```
import logging
import json

from ryu.app.wsgi import ControllerBase
from ryu.app.wsgi import Response
from ryu.app.wsgi import WSGIApplication
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller import dpset
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.exception import OFPUnknownVersion
from ryu.lib import mac
from ryu.lib import dpid as dpid_lib
from ryu.lib import ofctl_v1_0
from ryu.lib import ofctl_v1_2
from ryu.lib import ofctl_v1_3
from ryu.lib.packet import packet
from ryu.ofproto import ether
from ryu.ofproto import inet
from ryu.ofproto import ofproto_v1_0
from ryu.ofproto import ofproto_v1_2
from ryu.ofproto import ofproto_v1_2_parser
from ryu.ofproto import ofproto_v1_3
from ryu.ofproto import ofproto_v1_3_parser
```

```
SWITCHID_PATTERN = dpid_lib.DPID_PATTERN + r'|all'
VLANID_PATTERN = r'[0-9]{1,4}|all'

REST_ALL = 'all'
REST_SWITCHID = 'switch_id'
REST_VLANID = 'vlan_id'
REST_RULE_ID = 'rule_id'
REST_STATUS = 'status'
REST_LOG_STATUS = 'log_status'
REST_STATUS_ENABLE = 'enable'
REST_STATUS_DISABLE = 'disable'
REST_COMMAND_RESULT = 'command_result'
REST_ACL = 'access_control_list'
REST_RULES = 'rules'
REST_COOKIE = 'cookie'
REST_PRIORITY = 'priority'
REST_MATCH = 'match'
REST_IN_PORT = 'in_port'
REST_SRC_MAC = 'dl_src'
REST_DST_MAC = 'dl_dst'
REST_DL_TYPE = 'dl_type'
REST_DL_TYPE_ARP = 'ARP'
REST_DL_TYPE_IPV4 = 'IPv4'
REST_DL_TYPE_IPV6 = 'IPv6'
REST_DL_VLAN = 'dl_vlan'
REST_SRC_IP = 'nw_src'
REST_DST_IP = 'nw_dst'
REST_SRC_IPV6 = 'ipv6_src'
REST_DST_IPV6 = 'ipv6_dst'
REST_NW_PROTO = 'nw_proto'
REST_NW_PROTO_TCP = 'TCP'
REST_NW_PROTO_UDP = 'UDP'
REST_NW_PROTO_ICMP = 'ICMP'
```

```

REST_NW_PROTO_ICMPV6 = 'ICMPv6'
REST_TP_SRC = 'tp_src'
REST_TP_DST = 'tp_dst'
REST_ACTION = 'actions'
REST_ACTION_ALLOW = 'ALLOW'
REST_ACTION_DENY = 'DENY'
REST_ACTION_PACKETIN = 'PACKETIN'

STATUS_FLOW_PRIORITY = ofproto_v1_3_parser.UINT16_MAX
ARP_FLOW_PRIORITY = ofproto_v1_3_parser.UINT16_MAX - 1
LOG_FLOW_PRIORITY = 0
ACL_FLOW_PRIORITY_MIN = LOG_FLOW_PRIORITY + 1
ACL_FLOW_PRIORITY_MAX = ofproto_v1_3_parser.UINT16_MAX - 2

VLANID_NONE = 0
VLANID_MIN = 2
VLANID_MAX = 4094
COOKIE_SHIFT_VLANID = 32

```

```

class RestFirewallAPI(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_0.OFP_VERSION,
                    ofproto_v1_2.OFP_VERSION,
                    ofproto_v1_3.OFP_VERSION]

    _CONTEXTS = {'dpset': dpset.DPSet,
                 'wsgi': WSGIApplication}

    def __init__(self, *args, **kwargs):
        super(RestFirewallAPI, self).__init__(*args, **kwargs)

```

```
# logger configure
FirewallController.set_logger(self.logger)

self.dpset = kwargs['dpset']
wsgi = kwargs['wsgi']
self.waiters = {}
self.data = {}
self.data['dpset'] = self.dpset
self.data['waiters'] = self.waiters

mapper = wsgi.mapper
wsgi.registry['FirewallController'] = self.data
path = '/firewall'
requirements = {'switchid': SWITCHID_PATTERN,
                'vlanid': VLANID_PATTERN}

# for firewall status
uri = path + '/module/status'
mapper.connect('firewall', uri,
               controller=FirewallController, action='get_status',
               conditions=dict(method=['GET']))

uri = path + '/module/enable/{switchid}'
mapper.connect('firewall', uri,
               controller=FirewallController, action='set_enable',
               conditions=dict(method=['PUT']),
               requirements=requirements)

uri = path + '/module/disable/{switchid}'
mapper.connect('firewall', uri,
               controller=FirewallController, action='set_disable',
               conditions=dict(method=['PUT']),
               requirements=requirements)
```

```
# for firewall logs
uri = path + '/log/status'
mapper.connect('firewall', uri,
               controller=FirewallController, action='get_log_status',
               conditions=dict(method=['GET']))

uri = path + '/log/enable/{switchid}'
mapper.connect('firewall', uri,
               controller=FirewallController, action='set_log_enable',
               conditions=dict(method=['PUT']),
               requirements=requirements)

uri = path + '/log/disable/{switchid}'
mapper.connect('firewall', uri,
               controller=FirewallController, action='set_log_disable',
               conditions=dict(method=['PUT']),
               requirements=requirements)

# for no VLAN data
uri = path + '/rules/{switchid}'
mapper.connect('firewall', uri,
               controller=FirewallController, action='get_rules',
               conditions=dict(method=['GET']),
               requirements=requirements)

mapper.connect('firewall', uri,
               controller=FirewallController, action='set_rule',
               conditions=dict(method=['POST']),
               requirements=requirements)

mapper.connect('firewall', uri,
               controller=FirewallController, action='delete_rule',
               conditions=dict(method=['DELETE']),
               requirements=requirements)
```



```

# for VLAN data
uri += '/{vlanid}'
mapper.connect('firewall', uri, controller=FirewallController,
               action='get_vlan_rules',
               conditions=dict(method=['GET']),
               requirements=requirements)

mapper.connect('firewall', uri, controller=FirewallController,
               action='set_vlan_rule',
               conditions=dict(method=['POST']),
               requirements=requirements)

mapper.connect('firewall', uri, controller=FirewallController,
               action='delete_vlan_rule',
               conditions=dict(method=['DELETE']),
               requirements=requirements)

def stats_reply_handler(self, ev):
    msg = ev.msg
    dp = msg.datapath

    if dp.id not in self.waiters:
        return
    if msg.xid not in self.waiters[dp.id]:
        return
    lock, msgs = self.waiters[dp.id][msg.xid]
    msgs.append(msg)

    flags = 0
    if dp.ofproto.OFP_VERSION == ofproto_v1_0.OFP_VERSION or \
       dp.ofproto.OFP_VERSION == ofproto_v1_2.OFP_VERSION:
        flags = dp.ofproto.OFPSF_REPLY_MORE
    elif dp.ofproto.OFP_VERSION == ofproto_v1_3.OFP_VERSION:

```

```

        flags = dp.ofproto.OFPMPPF_REPLY_MORE

    if msg.flags & flags:
        return
    del self.waiters[dp.id][msg.xid]
    lock.set()

@set_ev_cls(dpset.EventDP, dpset.DPSET_EV_DISPATCHER)
def handler_datapath(self, ev):
    if ev.enter:
        FirewallController.regist_ofs(ev.dp)
    else:
        FirewallController.unregist_ofs(ev.dp)

# for OpenFlow version1.0
@set_ev_cls(ofp_event.EventOFPPFlowStatsReply, MAIN_DISPATCHER)
def stats_reply_handler_v1_0(self, ev):
    self.stats_reply_handler(ev)

# for OpenFlow version1.2 or later
@set_ev_cls(ofp_event.EventOFPPStatsReply, MAIN_DISPATCHER)
def stats_reply_handler_v1_2(self, ev):
    self.stats_reply_handler(ev)

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def packet_in_handler(self, ev):
    FirewallController.packet_in_handler(ev.msg)

class FirewallOfsList(dict):
    def __init__(self):
        super(FirewallOfsList, self).__init__()

```

```

] def get_ofs(self, dp_id):
]     if len(self) == 0:
]         raise ValueError('firewall sw is not connected.')
.
]     dps = {}
]     if dp_id == REST_ALL:
]         dps = self
]     else:
]         try:
]             dpid = dpid_lib.str_to_dpid(dp_id)
]         except:
]             raise ValueError('Invalid switchID.')
.
]         if dpid in self:
]             dps = {dpid: self[dpid]}
]         else:
]             msg = 'firewall sw is not connected. : switchID=%s' % dp_id
]             raise ValueError(msg)
.
]     return dps
.

] class FirewallController(ControllerBase):
.
]     _OFS_LIST = FirewallOfsList()
]     _LOGGER = None
.
]     def __init__(self, req, link, data, **config):
]         super(FirewallController, self).__init__(req, link, data, **config)
]         self.dpset = data['dpset']
]         self.waiters = data['waiters']

```

```

@classmethod
def set_logger(cls, logger):
    cls._LOGGER = logger
    cls._LOGGER.propagate = False
    hdlr = logging.StreamHandler()
    fmt_str = '[FW][%(levelname)s] %(message)s'
    hdlr.setFormatter(logging.Formatter(fmt_str))
    cls._LOGGER.addHandler(hdlr)

@staticmethod
def regist_ofs(dp):
    dpid_str = dpid_lib.dpid_to_str(dp.id)
    try:
        f_ofs = Firewall(dp)
    except OFPUnknownVersion as message:
        FirewallController._LOGGER.info('dpid=%s: %s',
                                         dpid_str, message)
    return

FirewallController._OFS_LIST.setdefault(dp.id, f_ofs)

f_ofs.set_disable_flow()
f_ofs.set_arp_flow()
f_ofs.set_log_enable()
FirewallController._LOGGER.info('dpid=%s: Join as firewall.',
                                dpid_str)

@staticmethod
def unregist_ofs(dp):
    if dp.id in FirewallController._OFS_LIST:
        del FirewallController._OFS_LIST[dp.id]
        FirewallController._LOGGER.info('dpid=%s: Leave firewall.',
                                         dpid_lib.dpid_to_str(dp.id))

```

```

# GET /firewall/module/status
def get_status(self, req, **_kwargs):
    return self._access_module(REST_ALL, 'get_status',
                                waiters=self.waiters)

# POST /firewall/module/enable/{switchid}
def set_enable(self, req, switchid, **_kwargs):
    return self._access_module(switchid, 'set_enable_flow')

# POST /firewall/module/disable/{switchid}
def set_disable(self, req, switchid, **_kwargs):
    return self._access_module(switchid, 'set_disable_flow')

# GET /firewall/log/status
def get_log_status(self, dummy, **_kwargs):
    return self._access_module(REST_ALL, 'get_log_status',
                                waiters=self.waiters)

# PUT /firewall/log/enable/{switchid}
def set_log_enable(self, dummy, switchid, **_kwargs):
    return self._access_module(switchid, 'set_log_enable',
                                waiters=self.waiters)

# PUT /firewall/log/disable/{switchid}
def set_log_disable(self, dummy, switchid, **_kwargs):
    return self._access_module(switchid, 'set_log_disable',
                                waiters=self.waiters)

def _access_module(self, switchid, func, waiters=None):
    try:
        dps = self._OFS_LIST.get_ofs(switchid)
    except ValueError as message:
        return Response(status=400, body=str(message))

```

```
msgs = []
for f_ofs in dps.values():
    function = getattr(f_ofs, func)
    msg = function() if waiters is None else function(waiters)
    msgs.append(msg)

body = json.dumps(msgs)
return Response(content_type='application/json', body=body)

# GET /firewall/rules/{switchid}
def get_rules(self, req, switchid, **_kwargs):
    return self._get_rules(switchid)

# GET /firewall/rules/{switchid}/{vlanid}
def get_vlan_rules(self, req, switchid, vlanid, **_kwargs):
    return self._get_rules(switchid, vlan_id=vlanid)

# POST /firewall/rules/{switchid}
def set_rule(self, req, switchid, **_kwargs):
    return self._set_rule(req, switchid)

# POST /firewall/rules/{switchid}/{vlanid}
def set_vlan_rule(self, req, switchid, vlanid, **_kwargs):
    return self._set_rule(req, switchid, vlan_id=vlanid)

# DELETE /firewall/rules/{switchid}
def delete_rule(self, req, switchid, **_kwargs):
    return self._delete_rule(req, switchid)

# DELETE /firewall/rules/{switchid}/{vlanid}
def delete_vlan_rule(self, req, switchid, vlanid, **_kwargs):
    return self._delete_rule(req, switchid, vlan_id=vlanid)
```

```
def _get_rules(self, switchid, vlan_id=VLANID_NONE):
    try:
        dps = self._OFS_LIST.get_ofs(switchid)
        vid = FirewallController._conv_toint_vlanid(vlan_id)
    except ValueError as message:
        return Response(status=400, body=str(message))

    msgs = []
    for f_ofs in dps.values():
        rules = f_ofs.get_rules(self.waiters, vid)
        msgs.append(rules)

    body = json.dumps(msgs)
    return Response(content_type='application/json', body=body)

def _set_rule(self, req, switchid, vlan_id=VLANID_NONE):
    try:
        rule = req.json if req.body else {}
    except ValueError:
        FirewallController._LOGGER.debug('invalid syntax %s', req.body)
        return Response(status=400)

    try:
        dps = self._OFS_LIST.get_ofs(switchid)
        vid = FirewallController._conv_toint_vlanid(vlan_id)
    except ValueError as message:
        return Response(status=400, body=str(message))

    msgs = []
    for f_ofs in dps.values():
        try:
            msg = f_ofs.set_rule(rule, self.waiters, vid)
```

APPENDIX 6: my_firewall.py

```
from ryu.app import rest_firewall
import urllib2

data = '{"nw_src": "172.16.1.2/32", "nw_dst": "172.16.1.3/32", "nw_proto": "ICMP", "actions": "ALLOW", "priority": "10"}'
url = 'http://localhost:8080/firewall/rules/0000000000000001'
req = urllib2.Request(url, data, {'Content-Type': 'application/json'})
f = urllib2.urlopen(req)
for x in f:
    print(x)
f.close()

data = '{"nw_src": "172.16.1.3/32", "nw_dst": "172.16.1.4/32", "nw_proto": "ICMP", "actions": "ALLOW", "priority": "10"}'
url = 'http://localhost:8080/firewall/rules/0000000000000001'
req = urllib2.Request(url, data, {'Content-Type': 'application/json'})
f = urllib2.urlopen(req)
for x in f:
    print(x)
f.close()
```