A Report on,

**DIFFERENT APPLICATIONS OF SDN USING MININET**

CMPE 210

Fall 2017



Under the guidance of,

Prof. ANDREW BOND

Submitted by,

Team: UNFILTERED

SANJEEV RAO PALAMAND

(011442275)

KISHORE KUMAR BHEEMAPPA HANABAR

(011103690)

**ABSTRACT**

In traditional networks, the control and the data plane are combined in the network node. In this approach, once the flow management has been defined, the only way to make an adjustment to the policy is via reconfiguration of the devices. With increasing use of mobile devices, the scaling of networks as per changing traffic demands is restricted by this approach. Software Defined Networks (SDN) is a process where the control is moved out of the individual network nodes and into the separate centralized controller. The controller can hence, exploit complete knowledge of the network to optimize flow management and support user requirements. A custom network topology is created in our project and by using a centralized SDN controller, the network is controlled. After programming the network in tune with our application, packets are generated to test different applications that we developed on the SDN. This project demonstrates the different capabilities of SDN, by implementing 3 major network applications- SDN switch as a hub, SDN switch as a Firewall, and Selective Routing using SDN.

**INTRODUCTION**

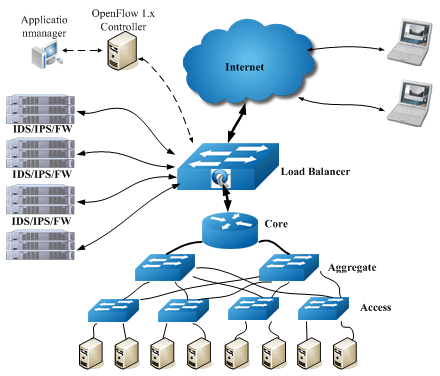
SDN

It is a network architecture approach that enables the network to centrally and intelligently control and program using software applications. The control and data plane are separated from each other and no longer exist on the same device. Programming of network behavior is controlled centrally by using software applications.

SDN provides a significant improvement in 4 fundamental areas of networking, such as follows:

1. Network programmability: Network behavior is controlled by a software which doesn’t not reside in the physical device which provide connectivity. Because of this architecture advantage it can support new services added to the network and scaling is no longer an issue.
2. Logically centralize intelligence and control: It’s centralized control plain contains full information about the entire network where as in traditional network control process is distributed so SDN provides facilities like bandwidth management, restoration, security, etc.
3. Abstraction of the network: SDN technology abstracted the technology which is laying under software which provides physical connection from network control.
4. Openness: Provides multi-vendor interoperability and fostering vendor-neutral ecosystem. The open APIs support a wide range of applications, including cloud orchestration, OSS/BSS, SaaS, and business-critical networked apps. In addition, intelligent software can control hardware from multiple vendors with open programmatic interfaces like OpenFlow.

The core of our project is SDN and there are few tools that we used to achieve it. They are: Mininet, POX Controller, and Scapy. They are briefly explained below. Figure 1 shows an SDN architecture that is explained in [1].



**Figure 1** Software Defined Networking (SDN) [1]

Mininet

Mininet is a software [emulator](http://whatis.techtarget.com/definition/emulator) for [prototyping](http://searchcio-midmarket.techtarget.com/definition/prototype) a large network on a single machine. It can be used to quickly create a realistic virtual network running an actual [kernel](http://searchenterpriselinux.techtarget.com/definition/kernel), [switch](http://searchtelecom.techtarget.com/definition/switch), software, and any application [code](http://whatis.techtarget.com/definition/code) on a personal computer.

Mininet allows the user to quickly create, interact with, customize and share a [software-defined network](http://whatis.techtarget.com/definition/software-defined-networking-SDN) (SDN) prototype to simulate a [network topology](http://whatis.techtarget.com/definition/network-topology) that uses [Openﬂow](http://whatis.techtarget.com/definition/OpenFlow) switches.

POX

POX is an open source development platform for Python-based control applications, such as Open Flow SDN controllers. The mininet topology can be controlled using various controllers and we decided to go with POX because of its adaptability to the application we were developing.

POX features:

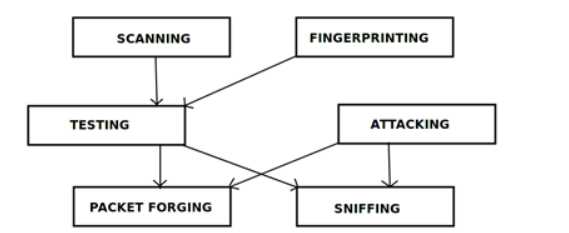
* Open source.
* Python based Open Flow interface.
* Reusable sample components for path selection, topology discovery, etc.
* Specifically targets Linux, Mac OS, and Windows.

Scapy

Scapy is a Python interpreter that enables you to create, forge, or decode packets on the network, to capture packets and analyze them, to dissect the packets, etc. It also allows you to inject packets into the network. It supports a wide number of network protocols and it can handle and manipulate wireless communication packets.

Scapy can be used to perform the jobs done by many network tools, such as nmap, hping, arpscan, and tshark (the command line of wireshark).

The concept behind Scapy is that it is cable of sending and receiving packets and it can sniff packets. The packets to be sent can be created easily using the built-in options and the received packets can be dissected. Sniffing of packets helps in understanding what communication is taking place on the network.



**Figure 2** Architecture of Scapy [2]

**IMPLEMENTATION**

We have implemented our project in the mininet environment on the Linux OS. A custom topology with 7 SDN switches and 8 hosts is created. Each switch is connected to 2 hosts. The nodes are all connected in a tree topology and thereby linking every host to other directly, or indirectly. We built 3 different applications on this network. They are as follows:

1. SDN switch as a hub
2. SDN switch as a firewall
3. Selective routing using an SDN switch

We distribute these applications on different switches in our network. The block diagram with our custom topology is as shown below.

**APPLICATION LAYER (Applications such as selective routing, firewall and hub)**

**C1 – Controller (POX)**

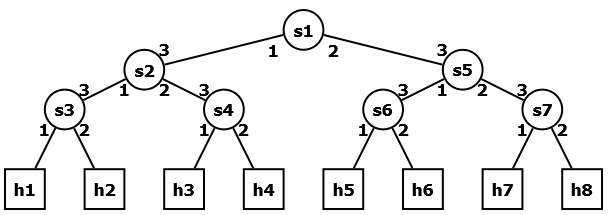
**s – SDN switch**

**h - Hosts**

**C1**

veth

Scapy



**Figure 3** Block diagram of the project

The controller sits on the control plane and interacts with the application layer with the northbound API. The applications that we developed reside in this layer. The southbound API of the controller interacts with the data plane where all the switches reside.

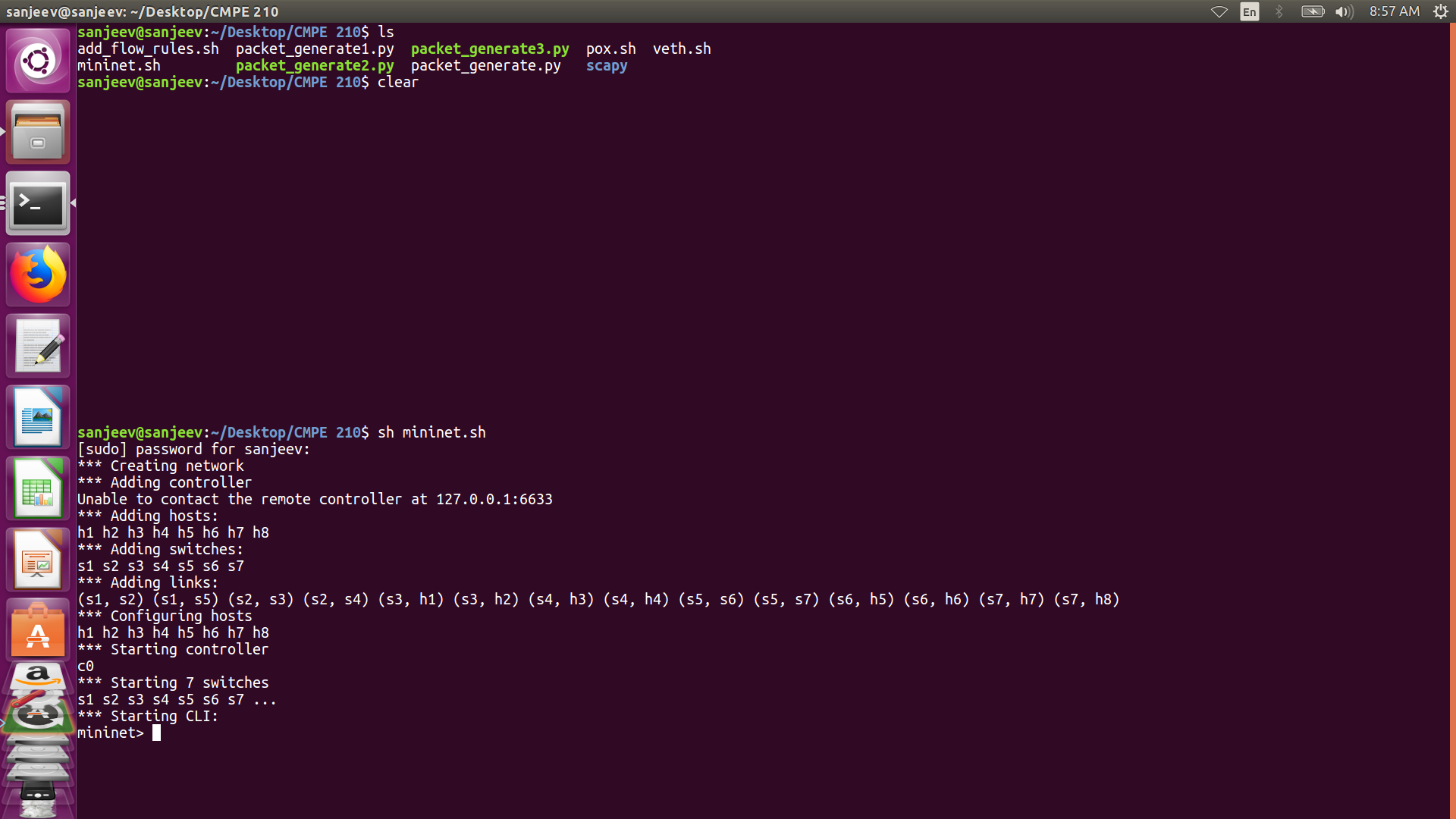
Scapy, a packet generator is chosen to generate a stream of packets for us. The following switches were running the applications we developed:

1. Switch s1🡪 Hub
2. Switches s2 and s5🡪 Firewall
3. Switches s3 and s7🡪 Selective Routing

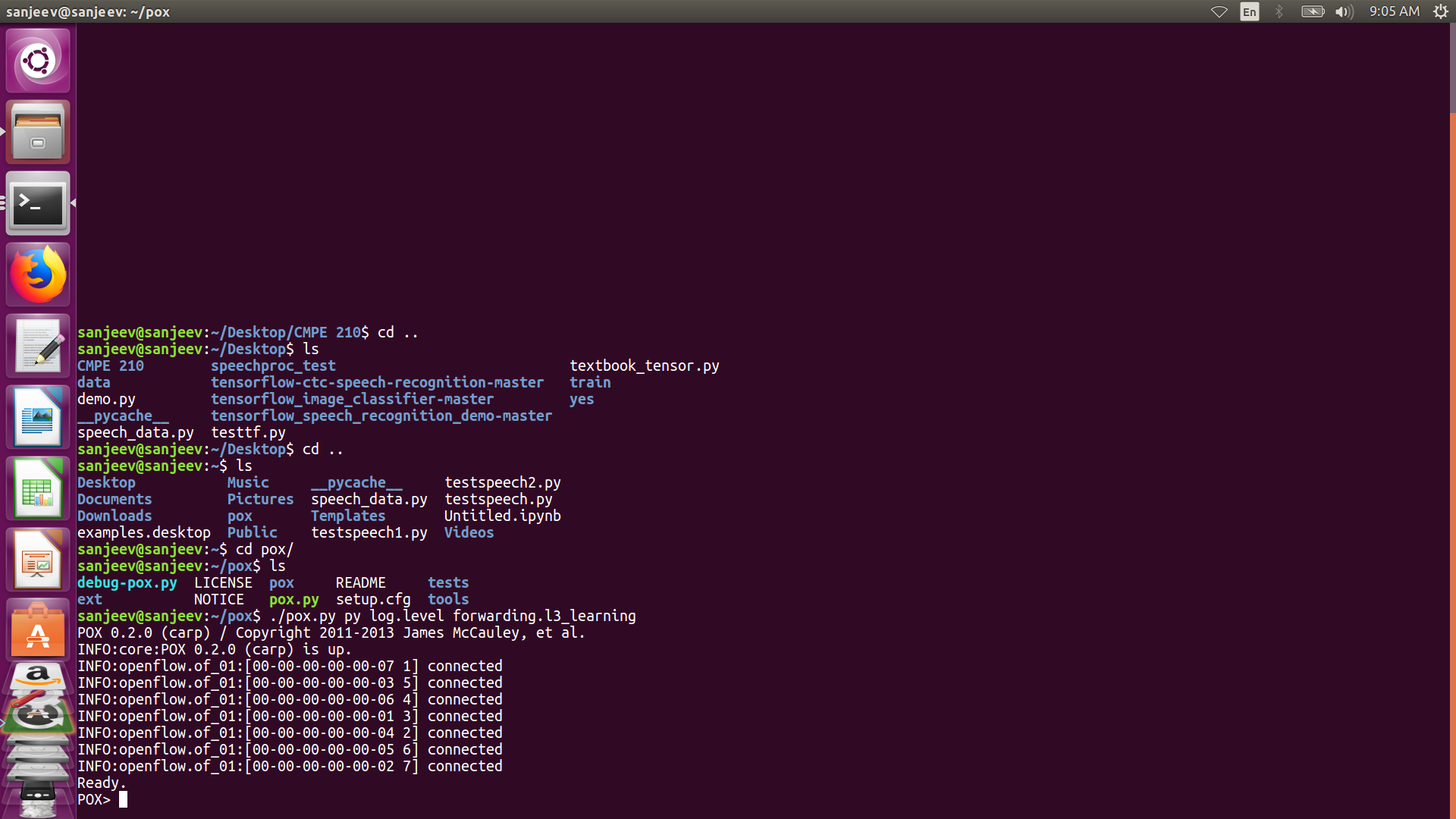
The respective switches are programmed accordingly by just selecting our applications. In simple terms, it is just like installing a software on our systems. Everything is ready to be installed. The whole project is automated and we have written scripts that automate all the applications. We have even automated the creation of the network in mininet and even that is a simple script that can run on any Linux machine.

The working of our model is explained in steps below.

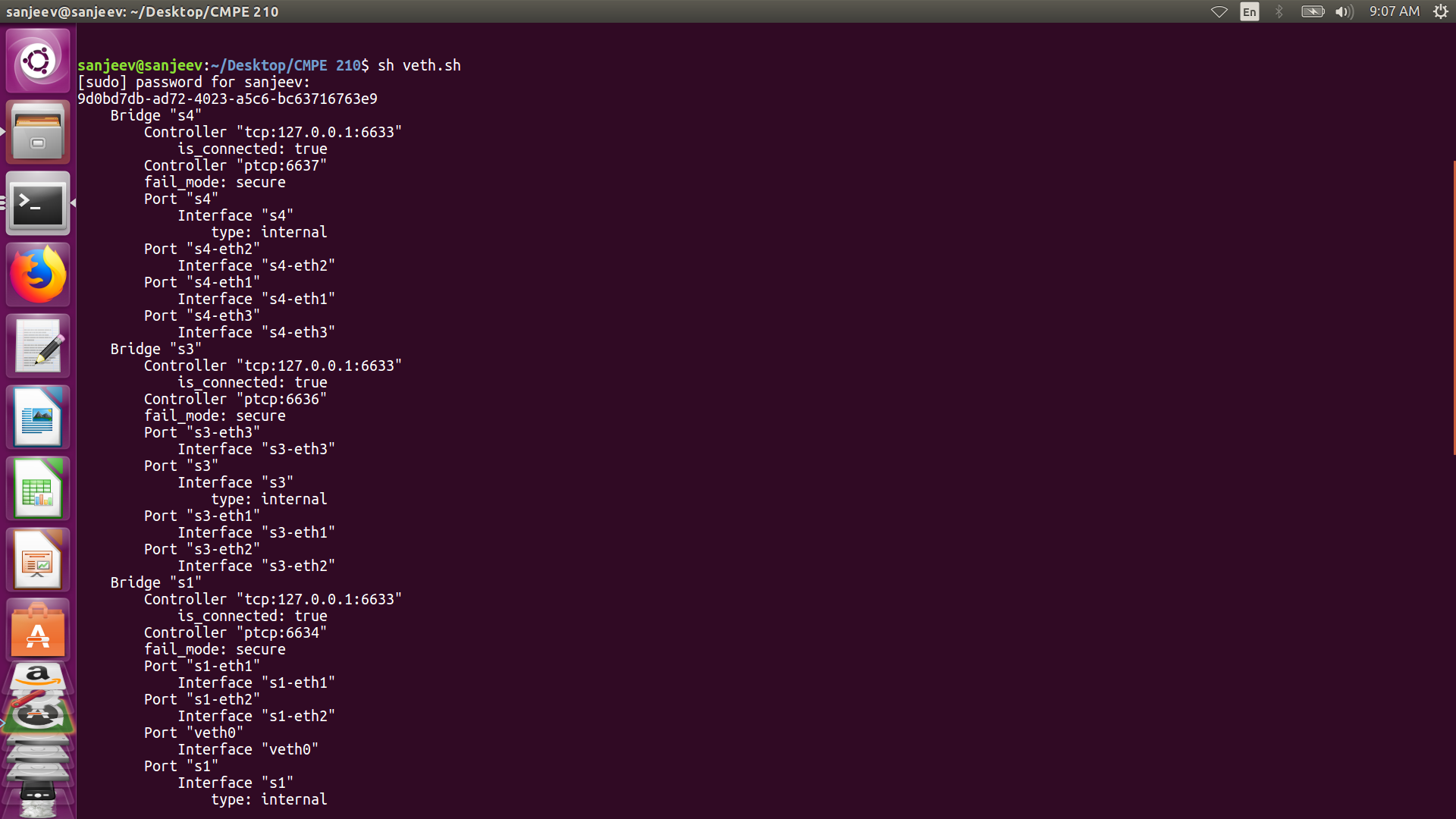
1. The mininet topology is created by running our customized automated script “mininet.sh” as shown in our screenshot below. This creates the mentioned topology that is shown in Figure.



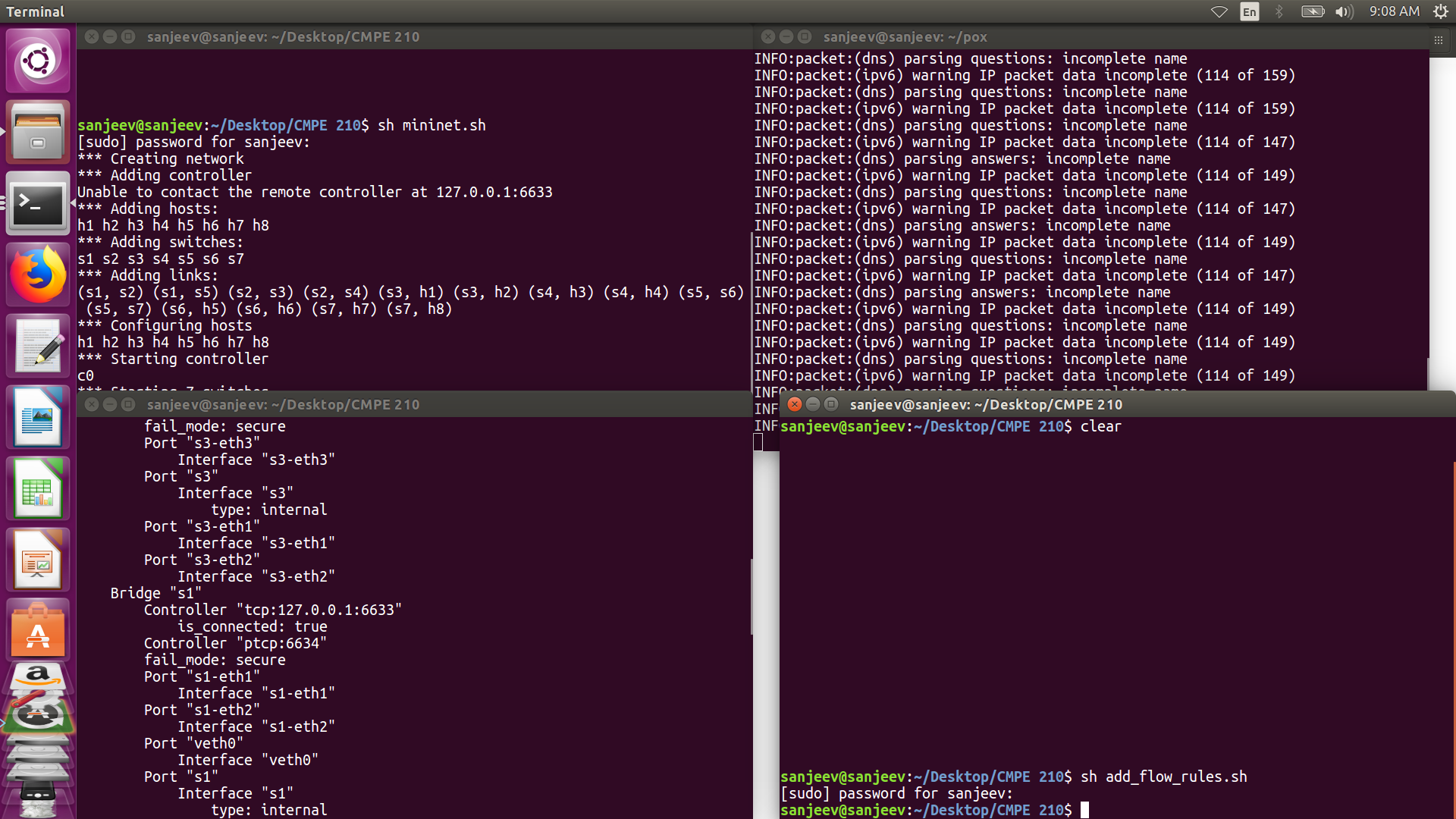
1. The POX controller is connected to all the switches in this topology. This step is executed as shown in the figure below. The POX controller gets invoked and gets connected to the mininet topology.



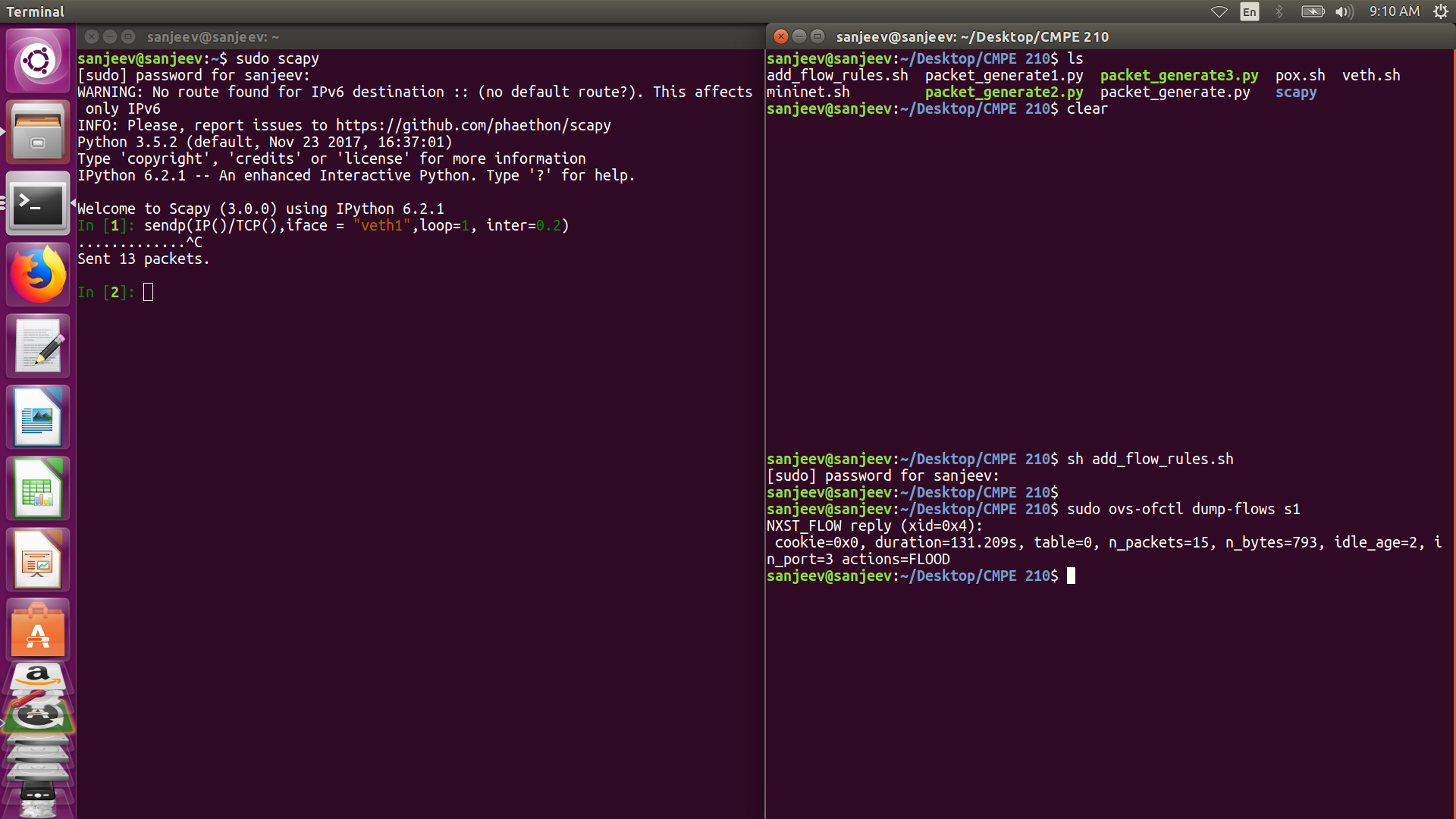
1. We send packets to switch s1 through Scapy. To connect to s1, we need to establish connection. This is achieved by creating a virtual ethernet cable and connecting the end of scapy to one port on s1. This veth pair is called veth1 and veth0. The creation is also an automated process and the script that we have created directly automates the connection.



1. Next, we add flows to the switch using the controller. We run the commands on the controller and this adds the necessary flows. The automation file for this step was built by us during this project and the execution is as shown below.



1. After adding the flows, we open scapy and create an IP packet using python. The reason we chose scapy was for its compatibility with our controller which is also python based. The generation of packets can be automated alongside the controller itself. The packet creation is as shown in the screenshot below.



We can see in the above screenshot that the flows that are sent from scapy are being received in s1. This is seen by dumping flows on the switch.

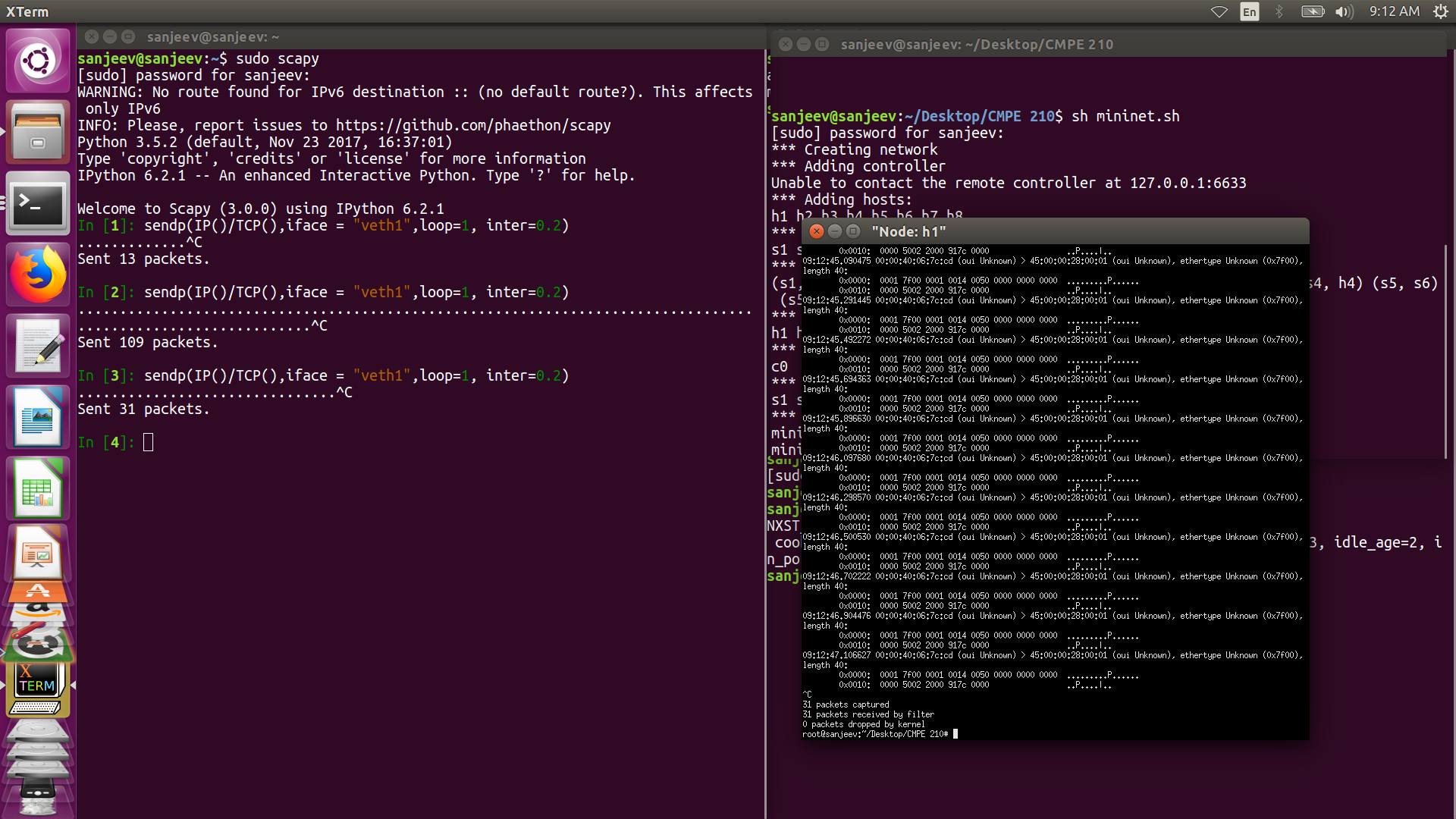
1. The flow rules for the switches are tabulated in the table below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Index** | **Switch** | **Flow Rule** | | **Application** |
| **input** | **output** |
| **1)** | S1 | All packets from port 3 | FLOOD to all output ports | HUB |
| **2)** | S2 | Any packet on port 3 | Send packets only to port 1, DROP packets to port 2 | FIREWALL |
| **3)** | S3 | Any packet on port 3 | Send packets only to port 1 | SELECTIVE ROUTING |
| **4)** | S5 | Any packet on port 3 | Send packets only to port 1, DROP packets to port 2 | FIREWALL |
| **5)** | S7 | Any packet on port 3 | Send packets only to port 2 | SELECTIVE ROUTING |

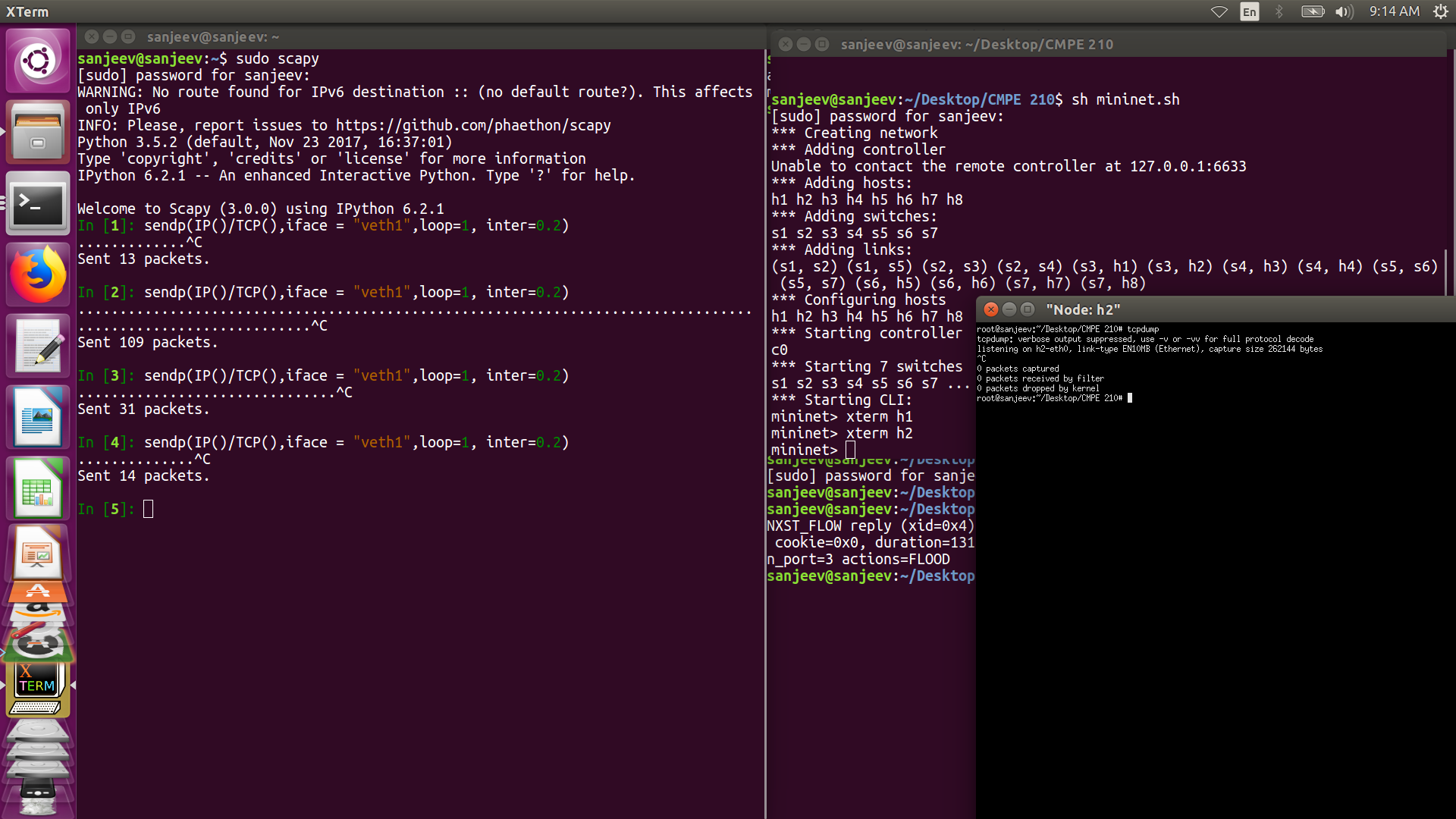
**Table 1** Flow Table for the switches

The “add\_flows.sh” file automates all these commands and executes the steps on the respective switches through the POX controller. We implement port-based routing in our application.

1. The working of the project is verified in this step. We would like to verify if our flow rules have been executed properly. According to our rules, switch S1 should flood all packets it receives. Switches S2 and S5 need to block any packets moving towards switches S4 and S6. Switches S3 and S7 need to selectively route packets to hosts h1 and h7 respectively. This means that only h1 and h7 will receive the packets being sent from scapy and none of the other hosts shall receive it. We verify this by initiating a packet flow on scapy.



In the above screenshot, we xterm into host h1 and tcpdump. The packets are being received and it is clearly as shown in the screenshot. The screenshot below shows the tcpdump for host h2.

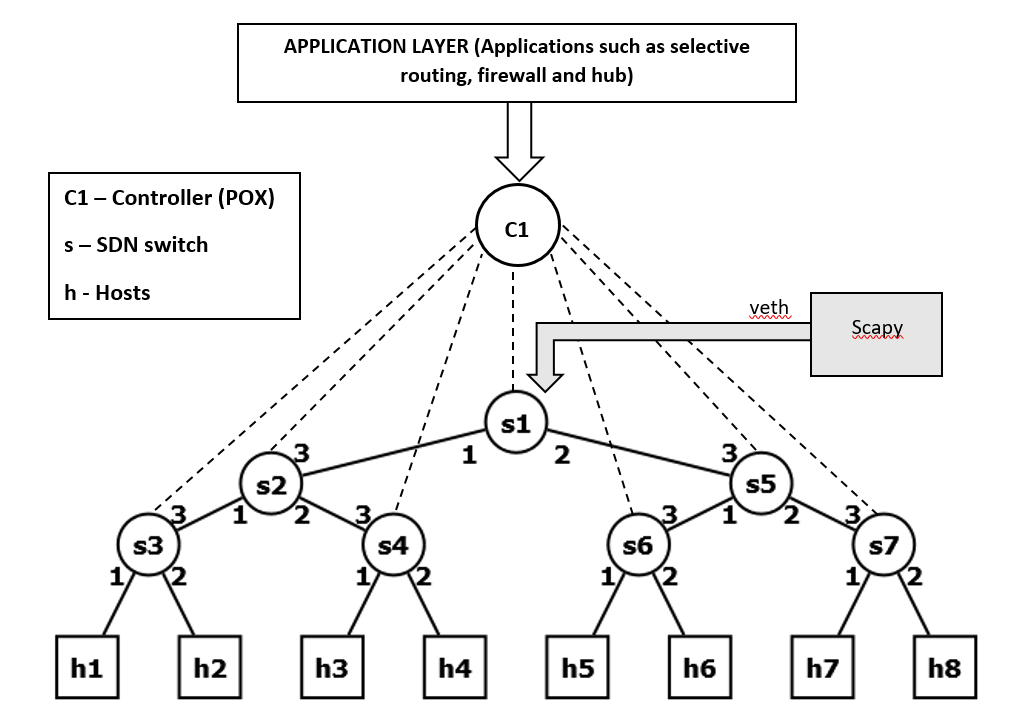


Host h2 does not receive any packets from s1 even though the packets are sent from scapy. This shows that our flow tables are working fine and the automation is also working properly.

The automated files are as given below.

** **

The block diagram shown below, which is an adaptation of Figure shows the working of our project based on the applications we developed on SDN. The packet flow is shown in green and the blocked switches are shown in red.

 **Figure 4** Working model of our project

**DISCUSSIONS AND CONCLUSIONS**

The applications that we have developed have some real-time test cases in the networking world. They are discussed as follows.

1. Highly scalable: Mininet is highly scalable and can emulate very large networks. The only drawback would be the CPU consumption, but with a dedicated RAM, that can be easily solved.
2. Selective Routing: This is an extremely useful application. Say host 1 is a premium customer and is paying more than others. The ISP wants to provide certain special services only to host 1 and not to any other customers. But the other host h2 is also in the same network. With SDN, host 1 can be given special access by the click of a button. And, if for any reason, the customer opts out from his premium account, he can be easily removed using the “selective routing” application that we demonstrated.
3. Firewall on switches: Parental control is an important firewall that many use. With SDN, enabling and disabling this feature for a specific group of users is just a click away!

Summarizing our project, we implemented multiple applications of SDN on an SDN network using mininet and POX controller. The switches were programmed to work as certain applications and we achieved the desired result of the applications that were developed.

We divided the responsibilities of this project and have briefly tabulated it as follows.

* Kishore Kumar Bheemappa Hanabar:
  + Mininet topology
  + Connecting POX controller to the topology remotely
  + Traffic generation
* Sanjeev Rao Palamand:
  + Adding flow rules to the switches
  + Application development
  + Automation of the developed applications

In conclusion, the project gave us the desired results for the applications that were developed. The whole project was automated and run by just executing scripts. This work can be extended to IP based routing and to real time networking scenarios outside the emulators as well.

**REFERENCES**

1. *SDN-DLB*, [online] Available: <http://clarkzinzow.me/projects-pages/folio-sdndlb.html>
2. *Dirk Loss*, [online] Available: <http://dirk-loss.de/>
3. S. Nanda, F. Zafari, C. DeCusatis, E. Wedaa, B. Yang, "Predicting Network Attack Patterns in SDN using Machine Learning Approach", *IEEE Conference on Network Function Virtualization and Software Defined Networks (IEEE NFV-SDN)*, 2016.
4. Patrick Engebretson, “The basics of Hacking and penetration Testing”, *2nd Edition Ethical Hacking Penetration Testing Made Easy, Syngress*, 2013
5. Sanjeev Rao Palamand, “Construction of a Simplified Software Defined Networking (SDN) Test-Bed”, Volume 10, Number 18, International Journal of Applied Engineering Research ISSN 0973-4562 (2015)