**Early detection of DDoS and its prevention using centralized**

**SDN-Controller**

------ Jagan Mohan Reddy Peddabavi

----- Class ID 30

----- 011540464

**Abstract:**

Basically, as we know (SDN) software-defined networking architectural framework eases the work of the n/w administrators by separating the data plane from the control plane. This provides easy network configuration by supporting a programmable interface for applications development related to security, management etc. and the centralized logical controller provides more control over the total network, which has complete network visibility.

These SDN advantages exposes the network to vulnerabilities and the impact of the attacks is much severe when compared to traditional networks, where the network devices have protection from the attacks and limits the occurrence of attacks.

In this project, we want to explore various attacks that can be launched on SDN at different layers. we also want to evaluate some of the existing security methods in mitigating the attacks and explore a possible solution to prevent DDoS attacks using entropy.

Basically, Distributed Denial of Service (DDoS) attack is a DoS attack which utilizes multiple distributed attack sources. We know that every network in the system has an entropy and increase in the randomness causes entropy to decrease.

For preventing this DDoS threat, we want to use POX for attack detection and want to provide a solution that is effective in terms of the resources used.

More precisely, this project shows how DDoS attacks can consume controller resources and provide a solution to detect such attacks based on the entropy variation of the destination IP address. Now based on this entropy value, we shall block that specific port in the switch if it drops below certain threshold value, and then bring the port down.

**Introduction:**

Now-a-days cloud services are expanding, and large organizations are migrating towards SDN-based implementations of network. These virtual technologies provide manageability, predictability and quality of service. Security provision importance is relevant for centralized managed network and has become one of the concerns. Considering a centralized virtual server running as a controller, which installs and manages the flows in data plane through OpenFlow communication protocol.

The use of OpenFlow makes the controller a primary victim for the attacker because of the following reasons

1. In OpenFlow protocol there is no standard for security implementation and developers of products are implementing their own proprietary methods.
2. The programmable aspect of SDN also makes them much more vulnerable to numerous malicious attacks and code exploits.
3. The southbound interface can be targeted with denial of service and side channel attacks.
4. Errors related to configuration of SDN can be more serious than traditional network errors.
5. Establishment of trust is also crucial.

**We worked on the creating a DDos attack and detection of the attack on the entropy base. And preventing the DDos to occur.**

**What is a DDos attack?**

Basically, is a type of flood attack. Here many packets are sent to a network device for stopping the service or decreasing the performance of such a device.

If the source addresses of incoming are spoofed, then the switch would not find a match and the packet needs to be forwarded to the controller. The collection of DDoS spoofed packets and legitimate packets can bind the controller into continuous processing, which exhausts them. Due to this, the controller is unreachable for the new incoming legitimate packets. This will bring the controller down causing loss to the SDN architecture. For a backup controller, the same challenge is to be faced.

Such kind of attacks can be detected in the early stage by monitoring few hundred of packets considering changes in entropy. The early detection of DDoS attacks stops the controller from going down. The term ‘early’ is related to tolerance level and traffic being handled by the controller. Due to this, the impact of malicious packets flooding can be controlled. Such a mechanism needs to be lightweight and high response time. The high response time saves the controller during the attack period for regaining the control by terminating the DDoS attack.

**Why entropy?**

The main reason for considering entropy is its ability for measuring randomness in a network. The higher the randomness the higher is the entropy and vice versa. So, whenever the entropy is less than a threshold value we can say that a DDos attack is occurred.

**Project Implementation**

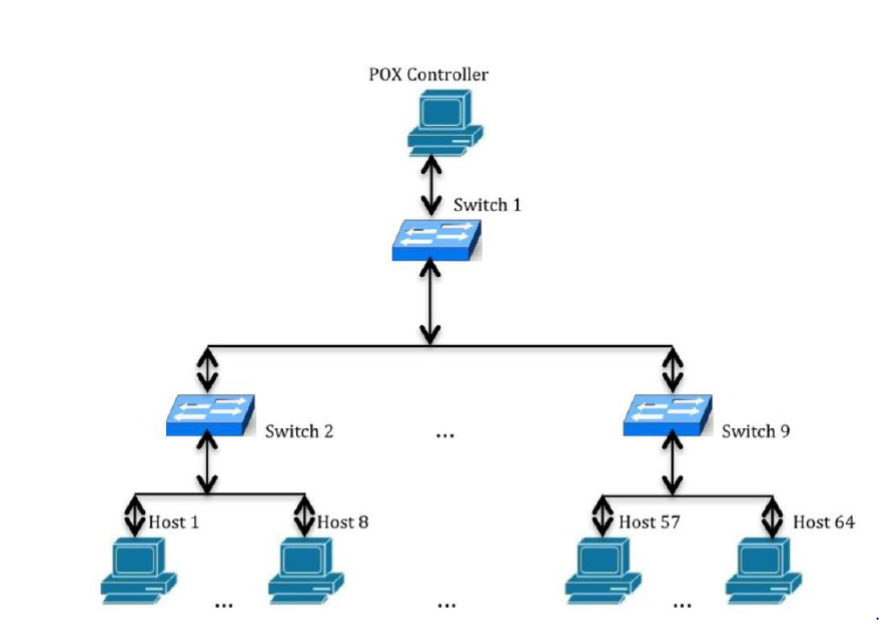
Basically, an Openflow controller is connected to a network. We then observe the entropy of the traffic related to the controller under normal and attack conditions.

We used the **POX** controller for this project, because it runs on Python.

The network emulator used is **Mininet** for creation of network topology.

Packet generation is done with the help of **Scapy**. Where Scapy is used for generation of packets, sniffing, scanning, forging of packet and attacking. Scapy is used for generation of UDP packets and spoofing the source IP address of the packets.

Here first we create a Mininet topology of 9 switches and 64 hosts as in fig below.



We then started pox controller from another terminal and then we can see 9 openflow links are connected for 9 open switches.

A screenshot of a computer screen

Description generated with very high confidence

**Now the project is divided into 2 parts ---1 packet generation & detection**

**---2 attack & detection**

1. packet generation: we then run the packet generation program from one of the host which generates random source ip and send the packets to random destinations. we then able to see the entropy in controller terminal.
2. Attack: Here we run the attack from few selected hosts to a selected target. Now the entropy value in the controller decreases.
3. After that for every 5 sets of process i.e. 250 packets entropy is below the threshold value then DDos is detected and those ip are blocked.

**Launch traffic code:(packet Generation)**

**Action**: In the launch traffic code, we basically, generates random source ip and send the packets to random destinations btw the host. We generally give start and end value from the command prompt of one of the node for example h1

**Modules imported:**

we import **sys** module of Python for accessing system specific parameters and functions.

The **getopt** module helps scripts to parse the command line arguments.

We use **os** module to import **Popen** to fetches shell commands into python program.

**Scapy** module is used to import functions such as sendp, IP, UDP, Ethernet and TCP.

The **random** module is used to import **randrange** function.

Here we used the **if\_name\_** is used to avoid the running of the main function when called from other programs.

**Code** **explaination**:

Here first initialized the start and end values of the destination hosts between which we send the packets from random source ips as 2 and 30.

Then we written a parser function, to fetch the start and end values from the command line interface to the python program.

Then gendest function is used to generate destination IP addresses of the hosts based on entered start and end values.

The sourceIPgen function is used for generating random source IP addresses and we don’t use the values [1,2,10,127,169,172,192] for the first octet of IP address as they are not valid to be used regular purpose (reserved). The sourceIPgen and gendest are linked with module pox.lib.addresses.

Then we need to open ethernet interface eth0 to send packets.

So, we used popen command that reads the IP address using ifconfig and awk is a text parser command for reading the contents.

Then packets are created and sent out from eth0 with the help of sendp along with the destination ip and the source ip that we created using randrange function and we assigned the UDP source and destination port addresses as 2 and 80 respectively.

These packets are linked to module pox.lib. Packet.ipv4.

Here we used sendp function send IP packets out with an interval of 0.2 seconds. The sendp function is linked to modules pox.lib.packet.arp for generating ARP request to send the generated IP packet to the appropriate host having the corresponding MAC address, pox.lib.recoco module is used for providing time interval between two sent packets and pox.openflow.libopenflow\_01 module is used for discovery of hosts and switches in the network.

**Launch attack code:**

**Action:** Here we run the attack from few selected hosts to a selected target. Now the entropy value in the controller decreases. Here we give the target ip address from the command prompt of the hosts which are acting like a botnet.

**Modules used:**

The Launch Attack code is like Launch Traffic code. The time module provides various time related functions. The logging module helps implement logging system for applications. The logging.getlogger function will suppress all messages that have a lower level of seriousness than error messages, before importing scapy.

**Code explanation:**

Here we get the IP address of target from the command prompt of the bonet armies. As in the same way of traffic launch we create the packet with the random src ip and the send it out through the eth0 with help of sendp function of scapy.

**Attack detection code:**

**Action:** here first we make a count of 50 packets in the window and then we calculate the entropy and compare it with threshold that we set and make a count of consecutive entropy value lower than threshold. If this count reaches 5 then we can say that ddos had occurred otherwise not.

For this we written a code for detection and did some changes in the l3\_learning module of the pox controller so that it can detect the ddos. These changes are explained below.

**Entropy formula**

Here we detect the entropy with the help of 2 factors

1. destination ip and
2. no. of the times it repeated.

Here we used the window size to be 50 and

Probability of an destination ip occurred in the window is given as pi

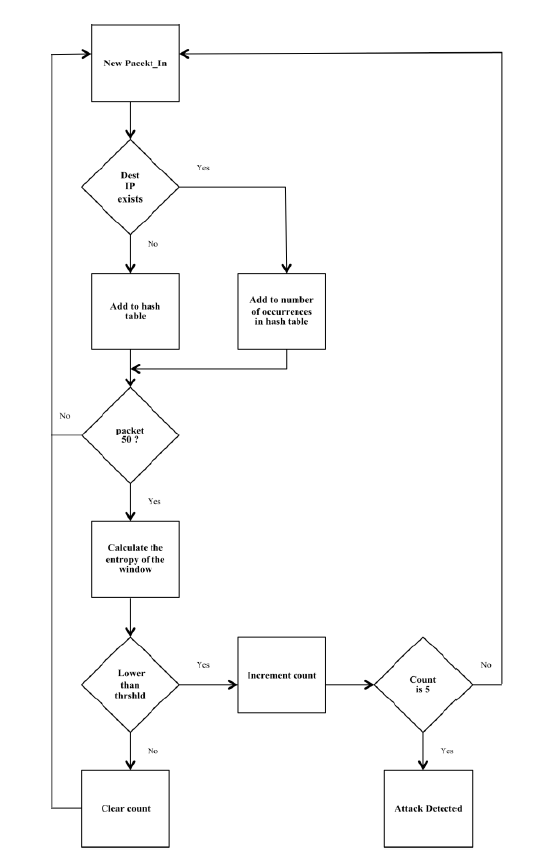
pi=(xi)/n where x is no. of event in the set and n to be the window size.

Now

entropy H= - sum of all (pi)log(pi)

Where i is from o to n

The sequence of steps we use in the detection code is given as a flow below.



**Modules imported in detection:**

In the Detection Code, **math** module is imported for performing mathematical functions such as finding log values. The **pox.core** module comprises of POX’s core API and functionalities. The core.getlogger function returns a logger, which is the root logger of the hierarchy.

**Code explaination:**

Here in the class Entropy, we defined the entropy dictionary, IP list and destination entropy are as null. The count and value are assigned values as 0 and 1 respectively.

In the statcolect function, we basically collect statistics related to the detection of attack i.e entropy.

Here every packet in message is collected into iplist and count is incremented for every packet in. When the count value reaches 50, the 50 Packet\_In messages would be parsed for their destination IP addresses in the hash table. If it is present in the table its value increments otherwise it is listed with 1. Entropy is calculated using this hash table values.

In the entropy function,

We calculate the probability of destination ip occurred in the window and then we calculate the entropy by the formula

entropy H= - sum of all (pi)log(pi)

Then we store this value in entropy dictionary which is used in l3\_learning module of pox to compare and say whether attack had occurred or not.

**L3\_learning module changes:**

Here we import the entropy function from detection file and then make set\_timer and defendddos as false as pox has a predefined method for defending ddos we are making it off.

In isinstance class of pox we will extract the entropy values of the windows from entropy dictionary and compare it with a threshold value (we used 0.5 here) and whenever entropy is less than threshold we implement preventing class otherwise we will set timer as false.

In the preventing class in handle\_openflow\_packetin , we create a diction of switch id and port along with number of times it appeared. It looks like {switch id, (port, count)} and we create timer is true here.

Now this diction is used to detect whether a ddos is occurred or not.

In the timer function we check the count value and if it is greater than equal to 5 then we can say that a ddos had occurred and we block the port of the switch by sending the

msg = of.ofp\_packet\_out(in\_port=i)

core.openflow.sendToDPID(dpid,msg)

**My contribution for the project:**

Primarily, I worked on attack launching and my project partner worked on packet generation.

Then we worked together on detection and then later I focused stimulation of the project.

**Manual**

**Installation of Scapy:**

sudo apt-get install python-scapy

**Test environment:**

1. Creation of a new python script for normal traffic generation in the folder mininet/custom

$ cd mininet/custom

$ vim launchTraffic.py

2. creating an attack traffic file and using the following command:

$ vim launchAttack.py

3. Running the following commands in Mininet:

$ cd pox/pox/forwarding

$ vim detection.py

**Steps for performing the project task:**

**Finding the threshold of the usual traffic:**

a. Creating a Mininet topology by entering the following command:

$ sudo mn --switch ovsk --topo tree,depth=2,fanout=8 --controller=remote,ip=127.0.0.1,port=6633

A screenshot of a computer screen

Description generated with very high confidence

b. In the Mininet terminal of virtual box enter the following command for running the pox controller:

$ cd pox

$ python ./pox.py forwarding.l3\_edited

A screen shot of a computer

Description generated with very high confidence

c. To determine the IP address of the POX controller, entering the following command in Mininet:

$ ifconfig

The loopback address is the IP address in the above command.

d. Now opening xterm for a host by typing the following command:

mininet>xterm h1

A screen shot of a computer

Description generated with very high confidence

e. In the xterm window of host h1 running the following command:

# python launchTraffic.py –s 2 –e 65

A screenshot of a cell phone

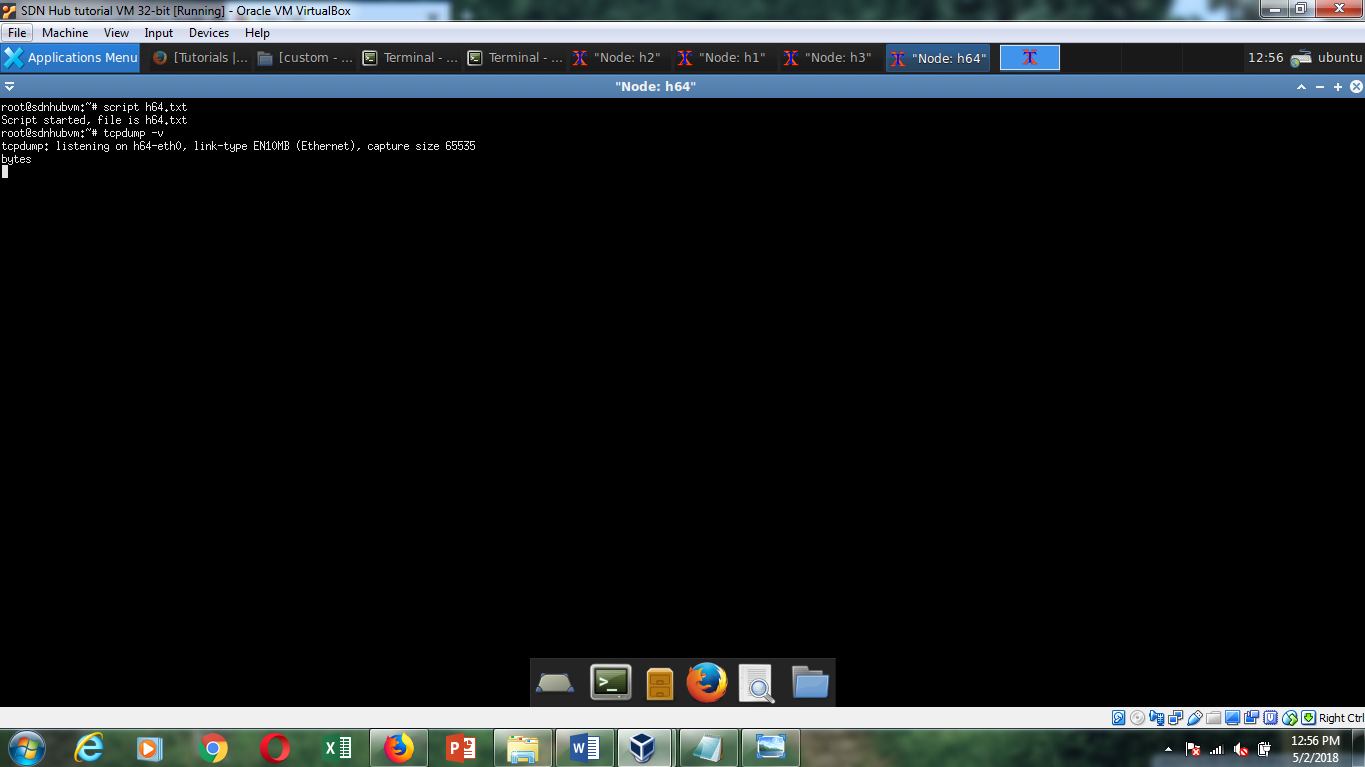
Description generated with very high confidence

**Detection of DDoS threat using the value of Entropy:**

On xterm window of h64 entering the following commands:

# script h64.txt

# tcpdump –v



**Entropy value before the DDoS attack:**

A screenshot of a computer

Description generated with high confidence

Now repeating step e) on h1 and parallelly entering the following commands to run the attack traffic from h4 and h6 xterm windows to attack on 56

# python launchAttack.py 10.0.0.56

A screen shot of a computer

Description generated with very high confidence

A screenshot of a cell phone

Description generated with high confidence

A screenshot of a cell phone

Description generated with very high confidence

A screenshot of a cell phone

Description generated with very high confidence

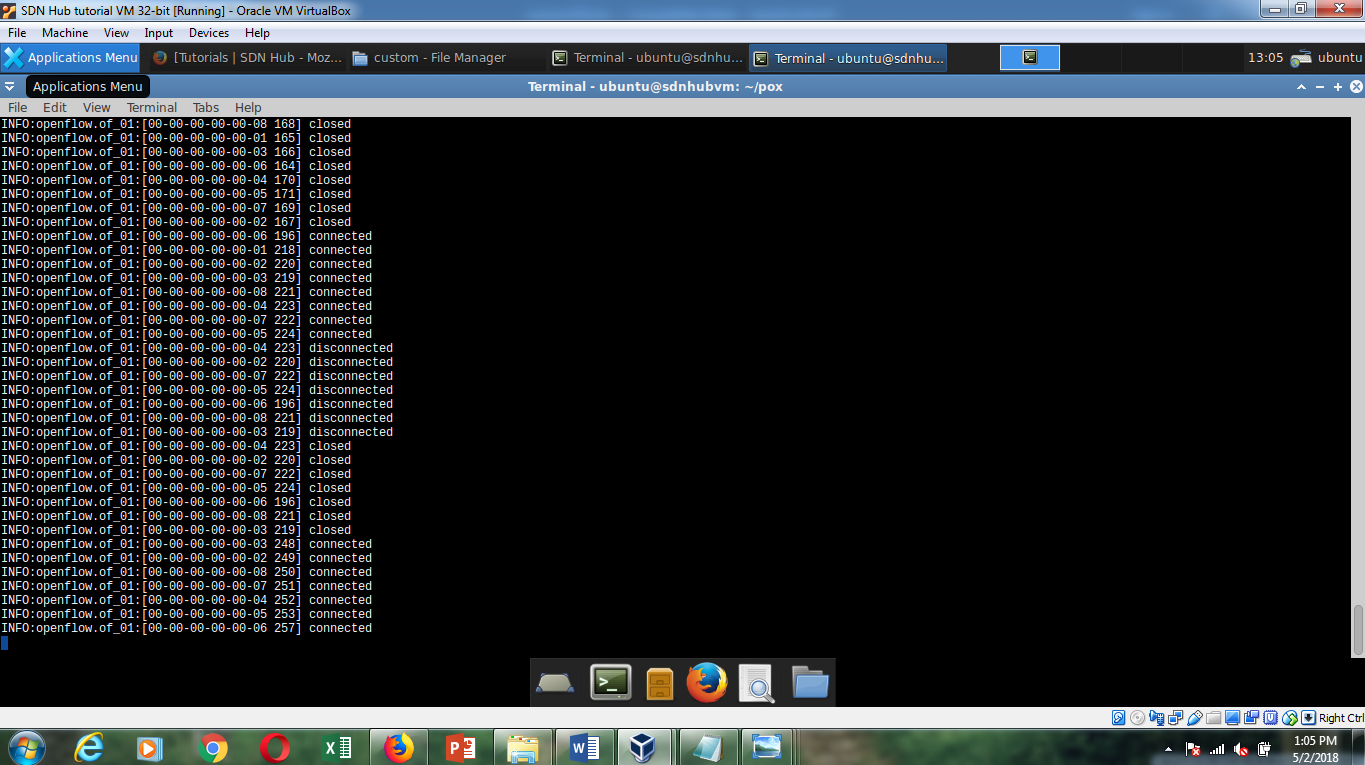
**Entropy value after DDoS attack:**

A screenshot of a cell phone

Description generated with very high confidence

Observing the entropy values in the POX controller. The value decreases below the threshold value (which is equal to 0.5 here) for normal traffic. Thus, we can detect the attack within the first 250 packets of malicious type of traffic attacking a host in the SDN network.

After the hosts stop sending attack packets, the switches are started again by the POX controller after shutting them down during the attack which is shown as follows:



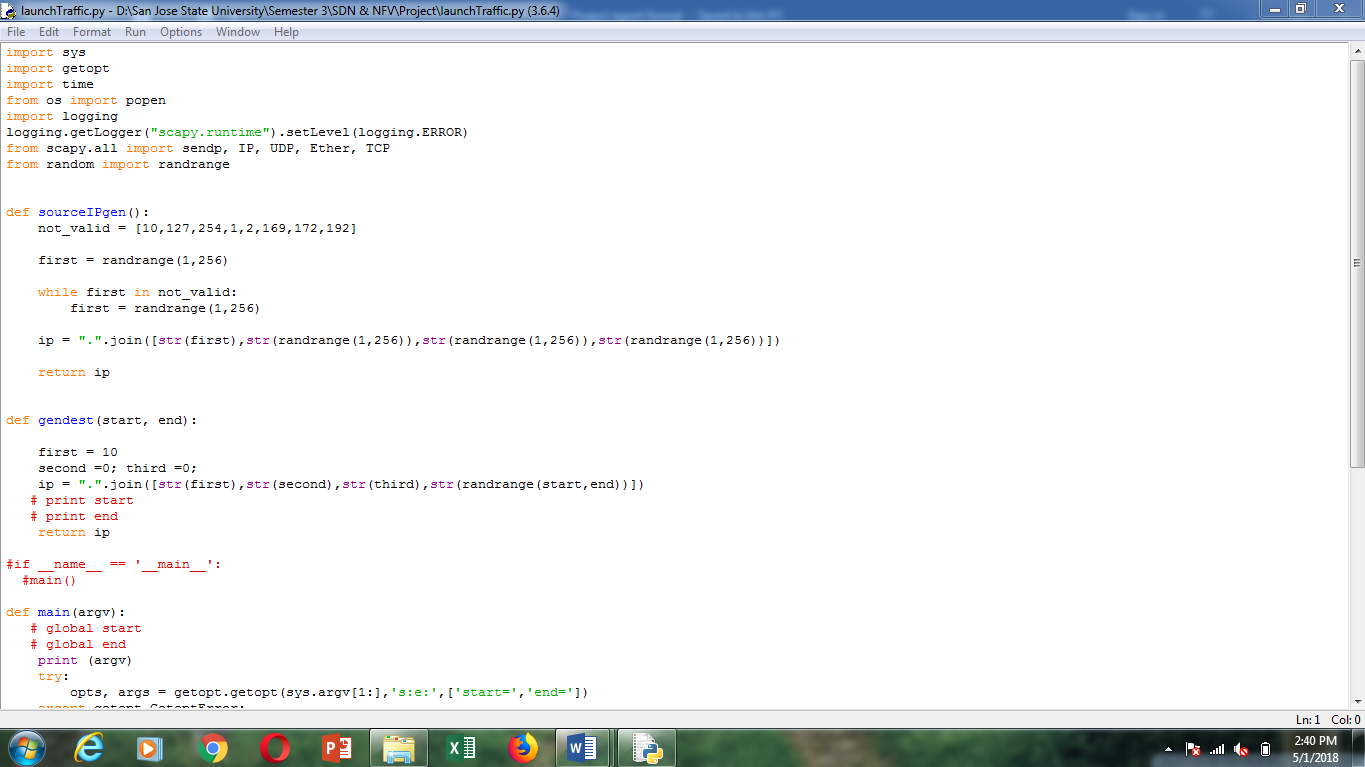
On successful completion of the above steps, terminating tcpdump on h64 by entering ‘control/command + c’

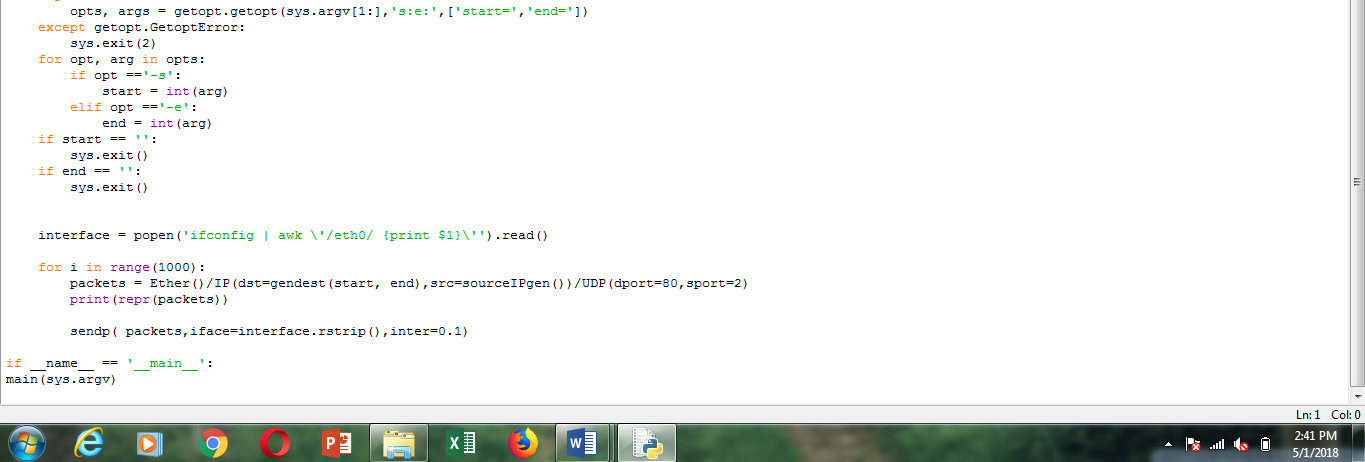
Stop running Mininet topology by entering the following command:

mininet>exit

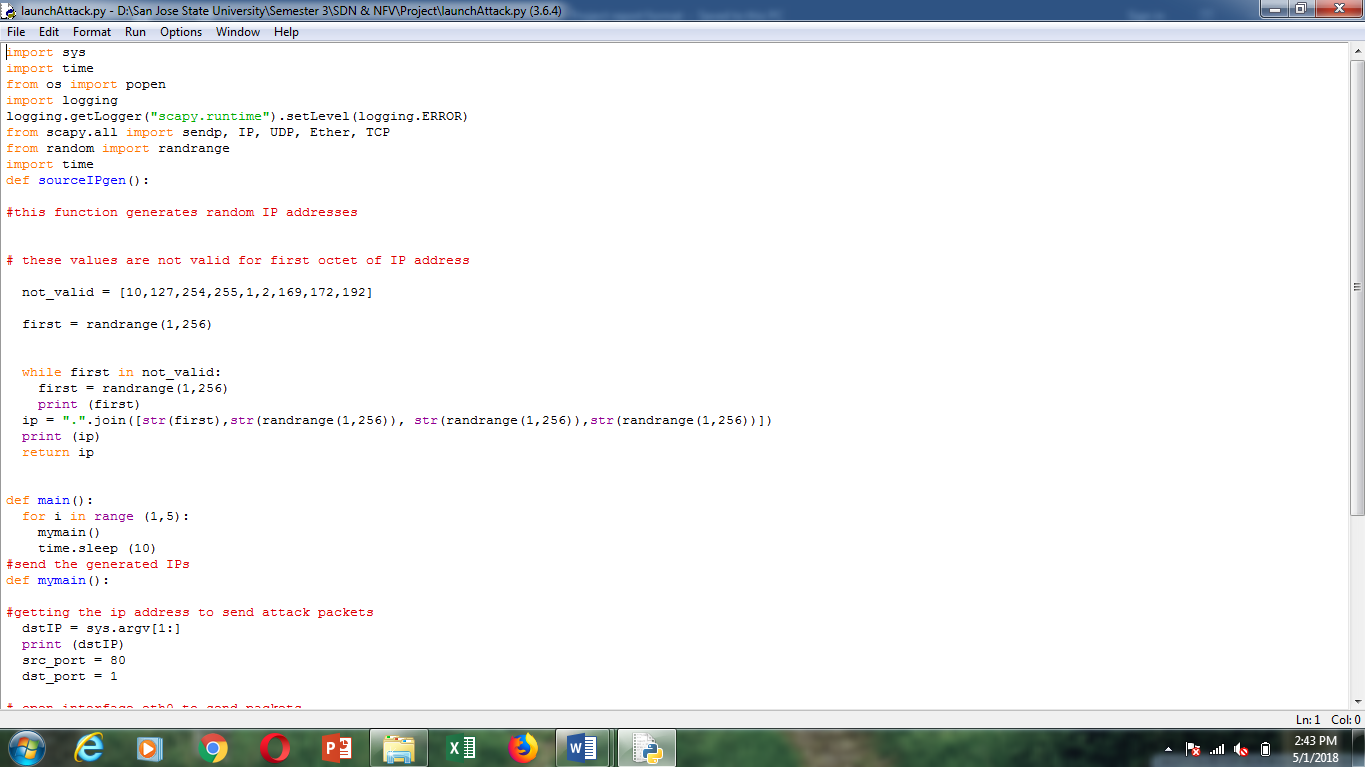
Appendix

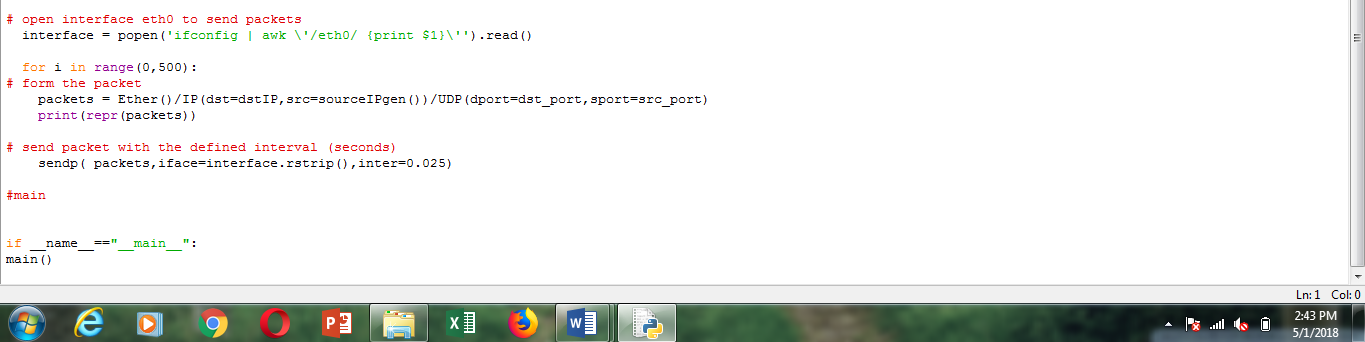
Launch Traffic





Launch attack





Detection code

