



MIT5200G Advanced Communication Networks

Network Denial of Service Detection Using SDN

Final Project Report - Individual

Group No: 02

Name: Samin Yasar

Student ID: 100796755

Submitted to:

Dr. Shahram S. Heydari

Associate Professor, Faculty of Business & IT

Ontario Tech University

Project Objective:

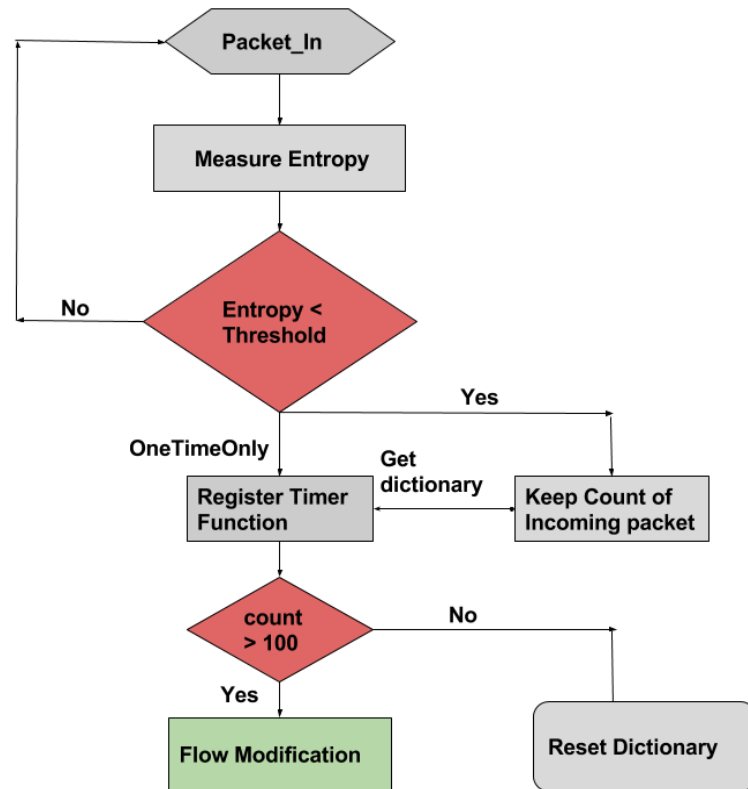
The main goal for this project was to learn about the Software-defined Architecture (SDN) and how it eases the administrators to manage the network by separating the control plane and the forwarding plane. This gave me an overview of the centralized controller, which is the control plane in the SDN facilitates the easy configuration, management, and security of the network [1]. Also, the forwarding plane or the data plane forwards the packets coming into the network matching the flow table is forwarded and if no flow is found how it communicates with the centralized controller using the OpenFlow protocol [2].

I learned about how the programmable centralized controller is vulnerable to a bunch of attacks and the focus of the project was Distributed Denial of Service (DDoS) attack detection using the SDN controller [3]. The DDoS is a type of attack that happens when an enormous amount the traffic is forwarded to the target host to exhaust the resources of the host and the attack is distribute from multiple sources. To simulate our environment, we used Mininet, and we used the built-in functionalities within the Mininet to create our environment we used pox as our controller [4], scapy a python-based library to generate packets to & from our SDN controller [5], python for the scripts, VMware to install the components and more will be discussed on these later in the report. In the attack we generated the all the source addresses were spoofed and as a result they will be forwarded out to the controller as no match will be found in the flow table of the data plane.

The aim of our project was to detect the DDoS attack happening in our system early. The theme to detect our attack early was based on entropy, which is the measure of randomness or disorderliness within the system [6]. The early detection mechanism is helpful as it will regain the control of the controller by detecting the first few hundred packets and the controller being flooded by malicious packets can be controlled considerably. The randomness of the incoming packets coming to the hosts were measured and if all the hosts have equal number packets coming to them the entropy will be maximum. So, when a particular host starts receiving a similar number of excessive packets the randomness decreases and as a result the entropy drops. As the DDoS attacks happen there will be large number of packets coming to a single host and the sudden dip in entropy will be observed.

So, we identified the test cases as seen from our presentation that we will first define a threshold of entropy for the normal traffic happening in our system as seen from the demonstration part of our presentation. Then generate attack to measure the entropy against the current threshold and if the entropy is below the threshold the number of incoming packets is counted for that threshold. We set the base entropy to 1, which is a 10% fault tolerance for our system.

The following flow diagram is given to better understand how it detects the DDoS attacks happening in our system.



Flow Chart Explanation:

Stage-1: For each packet_in message.

Stage-2: Measure the entropy value of the received packet.

Stage-3: If Entropy < Threshold

If yes then keep count of incoming packet & register it in the timer function

Else, not then go for normal execution.

Stage-4: Timer function gets update from the dictionary about the count of packet_in

Stage-5: If count > 100 then

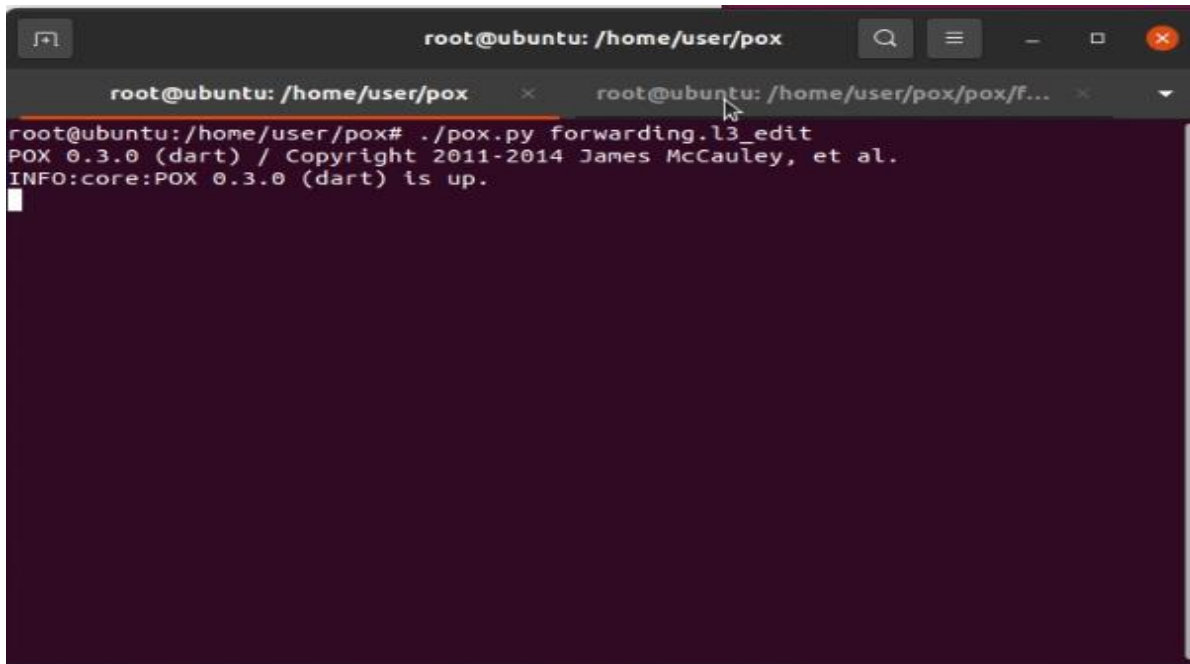
Flow modification message that is attack detected.

Else Reset dictionary

Stage-6: This ends the timer function and starts new timer.

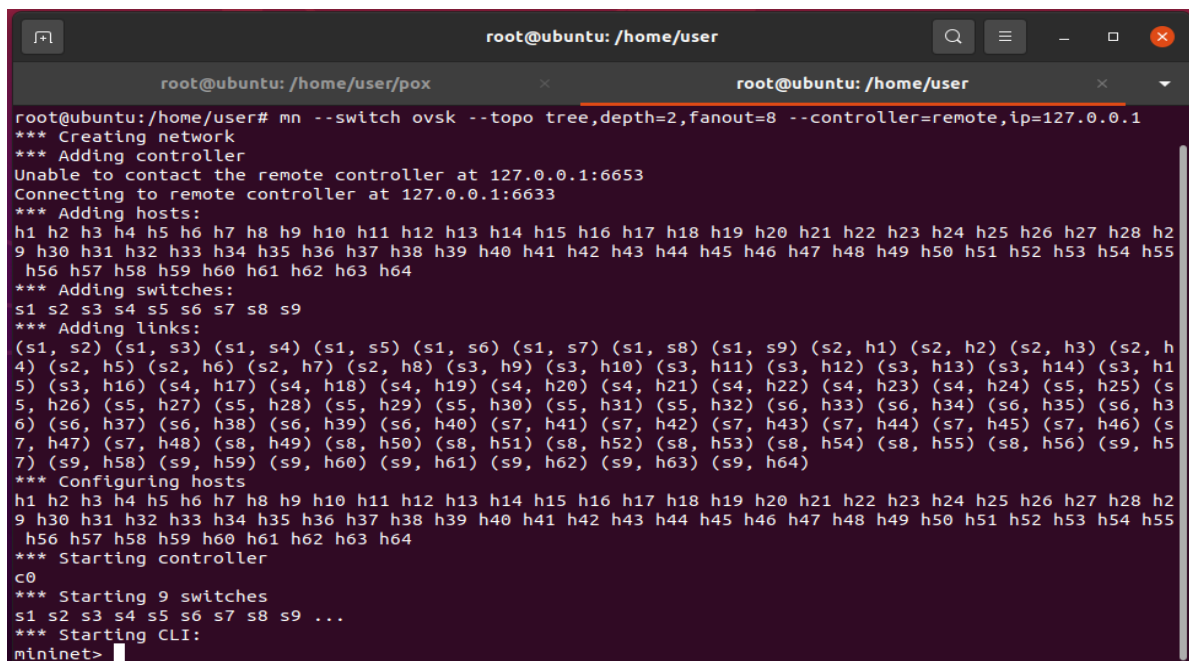
Project Results:

1) I ran the following command to run our pox controller, which comes built-in with the Mininet and it is very handy for experimentation.



```
root@ubuntu: /home/user/pox
root@ubuntu: /home/user/pox# ./pox.py forwarding.l3_edit
POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.3.0 (dart) is up.
```

2) I used the following command for running our topology in Mininet. The command creates a topology of 9 switches and 64 hosts.



```
root@ubuntu: /home/user
root@ubuntu: /home/user# mn --switch ovsk --topo tree,depth=2,fanout=8 --controller=remote,ip=127.0.0.1
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h2
9 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55
h56 h57 h58 h59 h60 h61 h62 h63 h64
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9
*** Adding links:
(s1, s2) (s1, s3) (s1, s4) (s1, s5) (s1, s6) (s1, s7) (s1, s8) (s1, s9) (s2, h1) (s2, h2) (s2, h3) (s2, h
4) (s2, h5) (s2, h6) (s2, h7) (s2, h8) (s3, h9) (s3, h10) (s3, h11) (s3, h12) (s3, h13) (s3, h14) (s3, h1
5) (s3, h16) (s4, h17) (s4, h18) (s4, h19) (s4, h20) (s4, h21) (s4, h22) (s4, h23) (s4, h24) (s5, h25) (s
5, h26) (s5, h27) (s5, h28) (s5, h29) (s5, h30) (s5, h31) (s5, h32) (s6, h33) (s6, h34) (s6, h35) (s6, h3
6) (s6, h37) (s6, h38) (s6, h39) (s6, h40) (s7, h41) (s7, h42) (s7, h43) (s7, h44) (s7, h45) (s7, h46) (s
7, h47) (s7, h48) (s8, h49) (s8, h50) (s8, h51) (s8, h52) (s8, h53) (s8, h54) (s8, h55) (s8, h56) (s9, h5
7) (s9, h58) (s9, h59) (s9, h60) (s9, h61) (s9, h62) (s9, h63) (s9, h64)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h2
9 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55
h56 h57 h58 h59 h60 h61 h62 h63 h64
*** Starting controller
c0
*** Starting 9 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 ...
*** Starting CLI:
mininet>
```

3) Now, I simulated a normal traffic in our environment to find out potential threshold of entropy in our system. As we can see from our results the entropy is set to 1, which is the threshold of entropy and it will detect attack when it goes below this threshold. We used Scapy to generate our packets coming from random ip and sending packets to random destinations.

```

root@ubuntu: /home/user/pox
root@ubuntu: /home/user/pox# ls
debug-pox.py  LICENSE  pox      README  tests
ext           NOTICE  pox.py  setup.cfg  tools
root@ubuntu: /home/user/pox# ./pox.py forwarding.l3_edit
POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.3.0 (dart) is up.
INFO:openflow.of_01:[00-00-00-00-00-07 8] connected
INFO:openflow.of_01:[00-00-00-00-00-06 2] connected
INFO:openflow.of_01:[00-00-00-00-00-05 10] connected
INFO:openflow.of_01:[00-00-00-00-00-03 4] connected
INFO:openflow.of_01:[00-00-00-00-00-04 9] connected
INFO:openflow.of_01:[00-00-00-00-00-08 6] connected
INFO:openflow.of_01:[00-00-00-00-00-01 3] connected
INFO:openflow.of_01:[00-00-00-00-00-02 5] connected
INFO:openflow.of_01:[00-00-00-00-00-09 7] connected
('Entropy : ', '1')
('Entropy : ', '1')
('Entropy : ', '1')
('Entropy : ', '1')
('Entropy : ', '1')
('Entropy : ', '1')
('Entropy : ', '1')
('Entropy : ', '1')
('Entropy : ', '1')
('Entropy : ', '1')
('Entropy : ', '1')
('Entropy : ', '1')
('Entropy : ', '1')
('Entropy : ', '1')
('Entropy : ', '1')

```

4) Finally, I generated the DDoS attack on our system we will see a sudden drop in the value of the entropy. As seen from the below output that the entropy is below 1 and it detects the DDoS happening in the system. So, it blocks the incoming port on the target host to stop the attack happening in the system early.

```

"Node: h1"
Sent 1 packets.
(Ether type=IPv4 I<IP frag=0 proto=udp src=114.247.148.39 dst=[10.0.0.64] I<UDP sport=80 dport=1 I>>>
Sent 1 packets.
(Ether type=IPv4 I<IP frag=0 proto=udp src=68.126.
Sent 1 packets.
(Ether type=IPv4 I<IP frag=0 proto=udp src=221.193.
Sent 1 packets.
(Ether type=IPv4 I<IP frag=0 proto=udp src=3.241.2
Sent 1 packets.
(Ether type=IPv4 I<IP frag=0 proto=udp src=199.184
Sent 1 packets.
(Ether type=IPv4 I<IP frag=0 proto=udp src=253.147
Sent 1 packets.
(Ether type=IPv4 I<IP frag=0 proto=udp src=93.159.
Sent 1 packets.
(Ether type=IPv4 I<IP frag=0 proto=udp src=217.110
Sent 1 packets.
(Ether type=IPv4 I<IP frag=0 proto=udp src=128.79.
Sent 1 packets.
(Ether type=IPv4 I<IP frag=0 proto=udp src=26.38.5
Sent 1 packets.
(Ether type=IPv4 I<IP frag=0 proto=udp src=66.92.5
Sent 1 packets.
(Ether type=IPv4 I<IP frag=0 proto=udp src=27.56.1
Sent 1 packets.
(Ether type=IPv4 I<IP frag=0 proto=udp src=37.222.
S CDROM is.
(Ether type=IPv4 I<IP frag=0 proto=udp src=218.127
Sent 1 packets.
(Ether type=IPv4 I<IP frag=0 proto=udp src=53.34.0
Sent 1 packets.
(Ether type=IPv4 I<IP frag=0 proto=udp src=43.174.
Sent 1 packets.
(Ether type=IPv4 I<IP frag=0 proto=udp src=79.185.23.204 dst=[10.0.0.64] I<UDP sport=80 dport=1 I>>>

root@ubuntu: /home/user/pox
root@ubuntu: /home/user/pox#
('dpid port and its packet count: ', '2', '[1: 9]', '9')
('Entropy : ', '0.200808982492')
('dpid port and its packet count: ', '2', '[1: 10]', '10')
('Entropy : ', '0.200808982492')
('dpid port and its packet count: ', '2', '[1: 11]', '11')
('Entropy : ', '0.200808982492')
('dpid port and its packet count: ', '2', '[1: 12]', '12')
('Entropy : ', '0.200808982492')
('dpid port and its packet count: ', '2', '[1: 13]', '13')
('Entropy : ', '0.200808982492')
('dpid port and its packet count: ', '2', '[1: 14]', '14')
('Entropy : ', '0.200808982492')
('dpid port and its packet count: ', '2', '[1: 15]', '15')
('Entropy : ', '0.200808982492')
('dpid port and its packet count: ', '2', '[1: 16]', '16')
('Entropy : ', '0.200808982492')
('dpid port and its packet count: ', '2', '[1: 17]', '17')
('Entropy : ', '0.200808982492')
('dpid port and its packet count: ', '2', '[1: 18]', '18')

DDOS DETECTED
('\n', '{2: {1: 18}}')
('\n', datetime.datetime(2021, 4, 19, 0, 50, 21, 599731), ': BLOCKED PORT NUMBER : ', '1', ' OF SWITCH I
'\n', '2')

root@ubuntu: /home/user/pox#

```

Project Role:

This was a group project and co-operation between our group members were essential to run the simulation and implementation of our project successfully. All the group members in our project equally contributed and I did the following for our project.

1) For the code part I found a suitable code for our project and along with my team we searched for codes that was the most suitable for our project. We did find a suitable code and we found out the functionalities of the scripts as not being that proficient in Python we could not write the code ourselves. I along with my teammates found that there was a compatibility issue with our code and the code was written for Python2. I along with my group members made it compatible with Python3. As there were four python scripts for our project, we divided the scripts between our four group members and each of us learned about how our part of the code works. We then had a discussion between our group members for the workings of the scripts. I worked on the attack.py script.

- i. traffic.py (packet generation code): In this code, generates random source ip addresses and sends packets to random destinations within the topology. We typically launch this normal traffic by using xterm to gain access to one of the hosts in the network and run the python script.
- ii. attack.py (attack generation code): This code is used to launch an attack to specific target in our network topology. Because of this attack the system entropy decreases. This code provides various time related functions and the logging module within this code helps implement system logging. This creates botnets and as the same way as described in the traffic.py creates a lot of random source ips and send out to the target host with the help of scapy. And we have explained in our previous flowchart related to entropy about how it works.
- iii. detection.py (DDoS detection code): In this code we made a count of packets in the window, we calculated the entropy and compared it with the threshold that we set in the traffic.py. When the entropy goes below this threshold then an attack will be detected. For this we made some changes in the l3_learning module of the pox controller so that it can detect the DDoS attack.
- iv. l3_edit.py: The l3_learning.py script which is by default comes with the pox controller and there was some editing done on it so that it can detect the DDoS attack. There were some classes present in the l3_edit.py. There was an instance class of the pox where it will extract the entropy values from the entropy dictionary and compares it with the threshold value. In the preventing class in handle_openflow_packetin, a dict of switch id and port along with the number of times it appeared. This dict was used to detect whether a DDoS attack occurred or not. In the timer function it counts the entropy value and after a certain number of packets it detects the DDoS has occurred and it blocks the port.

2) For presentation part each group members made their own slides, and they created their slides suitable to the project objectives and presented on it.

3) All of my group members along with myself did install all the necessary components needed to create our environment. I installed my components on VMWare. I did the following steps to install of the components.

- i. Downloaded the Ubuntu VM iso file.
- ii. Installed the VM appliance on my VMWare for creating the guest OS.
- iii. Now installed the Mininet on my Ubuntu VM in the following way:
 - a) `sudo git clone git://github.com/mininet/mininet`
 - b) `cd mininet`
 - c) `sudo git tag` # this command list all the available versions
 - d) `sudo git checkout -b mininet-2.3.0 2.3.0` # for installing the latest version
 - e) `cd ..` # This goes back to the previous directory
 - f) `sudo mininet/util/install.sh -a`

4) For creating the test environment for the project, I did so by following the below commands.

- i. Copy the python scripts of traffic.py and attack.py in the mininet/custom folder.
- ii. Copy the python scripts of detection.py and l3_edit.py to pox/pox/forwarding folder.
- iii. Pox controller this comes built in with the Mininet and to run the Pox controller we issued the below command.
`$ cd pox`
`$ sudo ./pox.py forwarding.l3_edit` # The l3_edit file is needed to run the pox.
- iv. As we used Mininet to create our custom topology and we issued the following command to run our topology. The command creates a topology with 9 OpenFlow switches and 64 hosts.
`$ sudo mn --switch ovsk --topo tree,depth=2,fanout=8 --controller=remote,ip=127.0.0.1`
- v. Now, use the xterm command, which opens a terminal to host h1.
`$ cd ~/mininet/custom`
`$ python3 traffic.py -s 2 -e 65` # This launches a regular traffic within the topology.
- vi. We observed the Pox controller and saw an Entropy value as seen from our results.
- vii. Now, we xterm to host h2 and launch the attack using the following command.
`$ cd ~/mininet/custom`
`$ python3 attack.py 10.0.0.64` # This command will launch an attack to host, h64.
- viii. After launching the attack, we will see changes to the value of the Entropy as seen from our results that when the value of is less than 1, then a DDoS attack was detected.

Conclusion & Future Work:

So, to conclude our report to work with Mininet as it was new to us and all our group members including me did a course on Udemy to better learn about the various functionalities of Mininet and that course was very helpful for us. I learned about SDN architecture how the control plane and forwarding plane communicate with each other using OpenFlow protocol. The SDN

decouples the two planes and the control plane, which is typically the brains of the SDN send instructions to the forwarding plane on how to forward packets according to the flow defined by the SDN controller. I also learned about the threat vectors related to the SDN architecture how it makes the SDN vulnerable to attacks. The aim of our project was the DDoS attack detection, which exhausts the resources of the target host. We simulated our environment using Mininet, used pox as our controller, scapy to generate our packets and some other components were also used as mentioned above.

In our above project simulation and implementation, we tried to detect our attack using entropy. We set an entropy threshold and generated attack on the target host and saw that there is sudden decrease in the value of the entropy. We detected the attack early and blocked incoming port on the detected host to stop the attack. There were some challenges in our simulation and all the codes we found on github were compatible to Python2 version and we made it compatible with the Python3 version to properly run our simulation. As our detection method was based on entropy are baseline threshold was set to 1 and if the attacker had known about this they can easily manipulate the packets so that it does not go below the threshold when simulating the attack. This is a challenge that we want to overcome as we continue to work on this. The number of false positives happening in our detection method is also a challenge and as our scheme blocks the incoming port that is under attack. So, this is hampering our network operations if a legitimate traffic packet is counted as below threshold and that is a challenge that we want to overcome also.

For future work of this project, we can do a Principal Component Analysis (PCA) to detect the DDoS attack happening in the system. We can do a comparison between the two analysis techniques and figure out which one will be more efficient for DDoS attack detection. Also, machine learning algorithms can also be an effective method for DDoS attack detection. Mitigation of the DDoS attack can be another area for future work. Adding more statistics collection to the controller and using more elaborate techniques we can pinpoint the malicious hosts, which is also a very interesting future work for our project.

References:

- [1] Velrajan, S. and Velrajan, S., 2021. SDN Architecture. [online] Tech.in | 5G, SDN/NFV & Edge Compute. Available at: <<https://www.thetech.in/2012/12/sdn-architecture.html>> [Accessed 26 March 2021].
- [2] En.wikipedia.org. 2021. OpenFlow - Wikipedia. [online] Available at: <<https://en.wikipedia.org/wiki/OpenFlow>> [Accessed 16 April 2021].
- [3] Hong, S., Xu, L., Wang, H. and Gu, G., 2021. Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures.
- [4] Open Networking Foundation. 2021. *MININET - Open Networking Foundation*. [online] Available at: <<https://opennetworking.org/mininet/>> [Accessed 16 April 2021].

- [5] Infosec Resources. 2021. *What Is Scapy? - Infosec Resources*. [online] Available at: <<https://resources.infosecinstitute.com/topic/what-is-scapy/>> [Accessed 16 April 2021].
- [6] Mousavi, S. and St-Hilaire, M., 2017. Early Detection of DDoS Attacks Against Software Defined Network Controllers. *Journal of Network and Systems Management*, 26(3), pp.573-591.
- [7] Kandoi, R. and Antikainen, M., 2015. Denial-of-service attacks in OpenFlow SDN networks. 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM),.
- [8] Dabbagh, M., Hamdaoui, B., Guizani, M. and Rayes, A., 2015. Software-defined networking security: pros and cons. *IEEE Communications Magazine*, 53(6), pp.73-79.
- [9] GitHub. 2021. Anandkumar26/DDOSAttack_SDN. [online] Available at: <https://github.com/Anandkumar26/DDOSAttack_SDN> [Accessed 20 April 2021].