

Configuration Manual

MSc Research Project
CyberSecurity

Vishnu Poovathoor Arunkumar
Student ID: 22224785

School of Computing
National College of Ireland

Supervisor: Raza Ul Mustafa

National College of Ireland
MSc Project Submission Sheet
School of Computing



Student Name: Vishnu Poovathoor Arunkumar
Student ID: 22224785
Programme: Masters in CyberSecurity **Year:** 2023-2024
Module: MSc Research Project
Supervisor: Raza Ul Mustafa
Submission Due Date: 12 August 2024
Project Title: Implementing Information theory techniques for detecting multi-vector DDoS attacks in SDN

Word Count: 2513 **Page Count:** 12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Vishnu Arun

Date: 11 August 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Vishnu Poovathoor Arunkumar
Student ID: 22224785

1 Introduction

The research conducted for this study involves utilizing a network simulation environment, that is hosted on a virtual machine. The configuration manual is intended to provide information on steps to reproduce the research and the results recorded in the report. The manual will also include commands needed for each part of the study, that can be executed in order to obtain the results. The manual is divided into the sections:

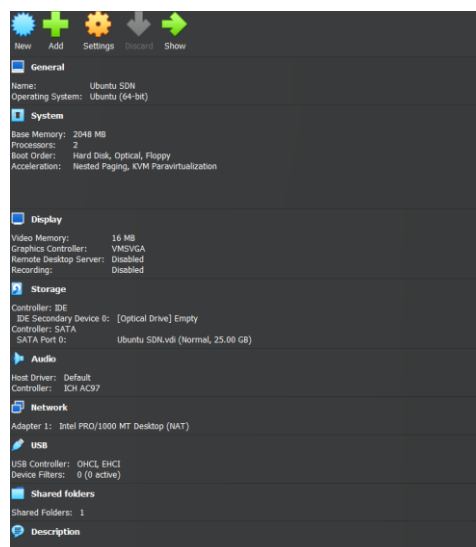
- Hardware specification
- Lab setup
- Simulation
- Trials and Logging

2 Hardware Specifications

The host machine specifications are as follows:

- Windows 11 13th Gen Intel(R)
- Core(TM) i5-1335U, 1300 Mhz, 10 Core(s), 12 Logical Processor(s)
- 16.0 GB Physical Memory (RAM)
- 475 GB SSD Memory

Oracle VirtualBox VM is used to deploy an Ubuntu 20.04.6 LTS VM on the host machine. VirtualBox is used for installing virtual machines that can run separate OS, and is installed on the host machine. (ORACLE,2019). Ubuntu 20.04.6 LTS VM can be installed by downloading the image from the website (releases.ubuntu.com., n.d.). The key resources specified for the VM machine is as follows, which are selected during the installation process of the Ubuntu image:



3 Lab setup

Use sudo to install software on the VM at root privileges, this is done by using “sudo su” and entering the password of your VM on the terminal of the VM.

3.1) Prerequisites

Install the following:

- python3: apt install python3
- pip3: apt install python3-pip
- xterm: apt-get install xterm
- git: apt-get install git
- Scapy: pip3 install scapy

3.2) Mininet

The network simulation is carried out using Mininet. Steps for installing Mininet are explained as follows:

Clone into the Mininet git repository, and run install.sh with the “a” flag to install additional software for the lab. The commands are:

- git clone <https://github.com/mininet/mininet>
- cd mininet
- mininet/util/install.sh -a

Mininet comes with MiniEdit, a GUI based network topology editor, which is used to create the topology script attached in the artefacts section. The script can be copied to the Desktop of your VM. The script can be launched by using the interpreter itself from the Desktop terminal:

- python3 topology.py

The topology will search for a remote controller which needs to be started within the VM. It is good practice to clear any residual data after closing the topology itself, as it ensures a fresh loading of the network virtual devices. This is done with:

- mn -c

3.3) POX SDN controller

Installing additional components includes the POX controller, the SDN controller used for the project. However, I have installed POX by cloning into the git repository of POX.

Install POX by cloning into the git repository:

- git clone <http://github.com/noxrepo/pox>

After downloading the directory:

- cd pox
- git branch

The default branch is set to `gar` and will display this, I have changed it to use the “`halosaur`” branch, which has Python3 based functionalities added to it:

- git checkout halosaur
- git branch

The POX directory will be copied onto the Desktop after following these steps. The directory includes the different components that exist, out of which the `l3_learning` component was customized entirely for the purpose of this project. The modified controller is shared with the artefacts. Copy this controller program into the “ext” filepath within the POX directory. This is where new or custom components are usually stored. The controller can be launched from a Desktop terminal with:

- `python3 pox/pox.py ext.DDoS_Controller`

This starts the POX controller, which listens for connections at the default address of 127.0.0.1 at port 6633, which is used by OpenFlow.

3.4) Starting the lab

After connection is established, your VM screen should look like the following:

[illegible]

The left terminal shows the controller, version details, and that it has successfully connected to the two OVSKernel Switches that is used in the network. The right terminal shows the successful initialization of the topology, and the network properties assigned to it from the specifications given in the TCLink parameters in the script. The network details can be verified by looking at the code artefact. For reference, the parameters given to the network links are as follows:

The main parameters customized are bandwidth, delay, packet loss and use hierarchical token bucket.

between switch s1 to first 6 hosts:

- `net.addLink(h, s1, bw=100, delay='5ms', loss=0, use_htb=True)`

between switch s2 to the remaining 6 hosts:

- `net.addLink(h, s2, bw=100, delay='5ms', loss=0, use_htb=True)`

between the switches s1 and s2:

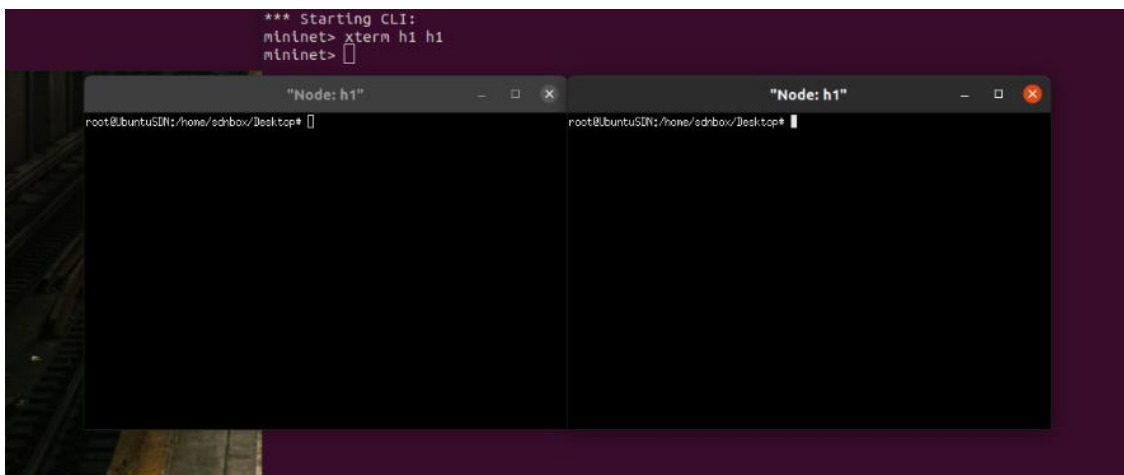
- `net.addLink(s1, s2, bw=1000, delay='1ms', loss=0, use_htb=True)`

3.5) xterm windows

The network traffic is started for both regular and attack purposes using the virtual hosts in the lab by using xterm windows. It can be opened for them by using the following command inside the terminal that is running the Mininet simulation (Mininet Project Contributors, 2021):

- `mininet> xterm hostname`

You can open multiple xterm windows for the same hosts as well. Opening the windows for hosts h1 twice should look like this:



The simulation, attack pattern testing will be carried out using these windows, while the Entropy values and detection will be seen on the terminal running the POX controller.

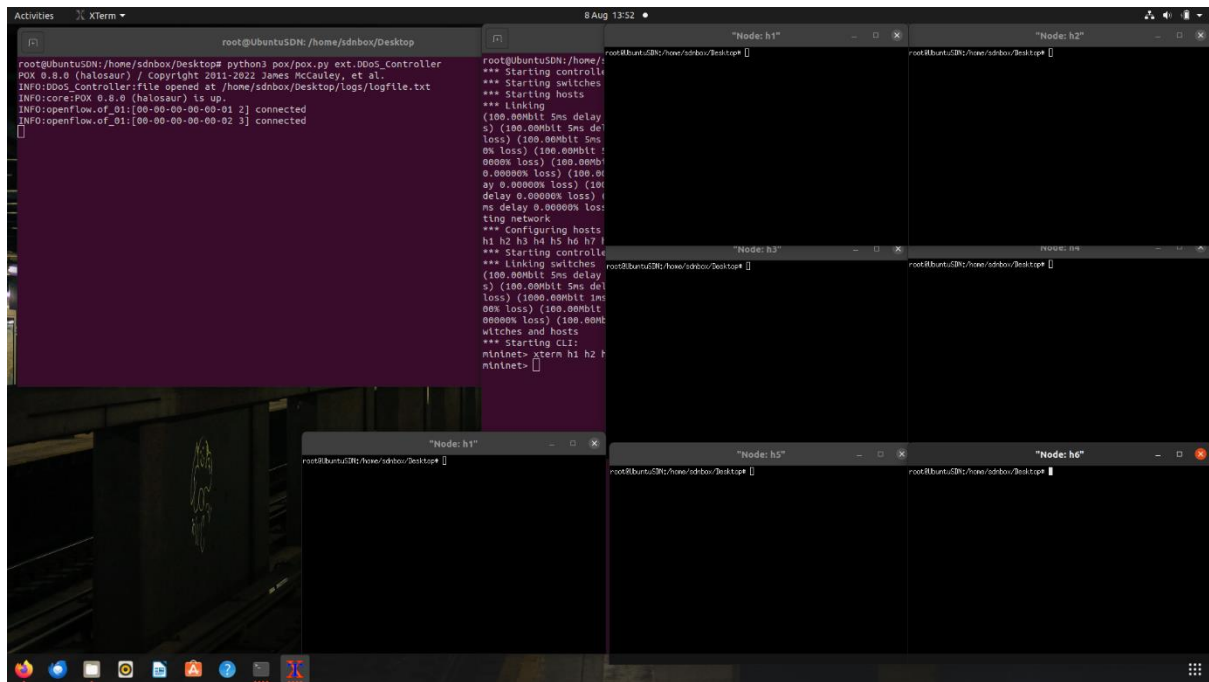
3.6) Log directory

The POX controller component also includes a feature to log the entropy metrics into a text file. To use this, make a directory on the Desktop by using a VM terminal, and give all the necessary privileges:

- `mkdir home/yourVMname/Desktop/logs`
- `chmod 777 logs`
- `ls -ld logs`

The logging system works per simulation, it means that after a simulation is over, it overwrites a new logfile within the directory, so make sure it has been copied to a different location for later use after each trial.

After setting up the lab tools, the final lab should look similar to this:



Most of the attack patterns are simulated within the host range from h1 to h6 for the study, and the regular traffic is simulated for the entire network via an extra h1 xterm window.

A few useful commands to test in the xterm windows to make sure everything is running fine. Test these commands in any xterm window (Mininet Project Contributors, 2021):

- ping 10.0.0.host-number

To test network performance, assuming h1 to be the server:

- iperf -s

Run this command on any other host xterm after the previous step to check UDP stats:

- iperf -c 10.0.0.1 -u

3.7) Traffic scripts

Three traffic scripts are created for the study and shared in the code artefacts section. The scripts are titled

- realtraffic.py
- udpflood.py
- bursttraffic.py

These files are to be copied to the Desktop of your VM as well.

Finally, make sure all the programs are given required privileges for execution, you can set it to root privileges by using chmod and check the privileges after:

- chmod 777 filename
- ls -l filename OR ls -ld directoryname

4 Simulation

The lab is ready for starting the testing after setting it up from the previous steps defined.

4.1) Normal traffic baseline

The normal traffic generated by the script “realtraffic.py” acts as a baseline for this project. The traffic simulated is used to populate the network with random activity, similar to traffic in a network without any attacks. The script is to be executed in host h1’s xterm window at all times during different attack scenarios. Execute the script as follows:

- `python3 realtraffic.py -s 1 -e 12 -d 10000`

The “s” flag is used to specify the first IP address in the network, that is host h1. The “e” flag is used to specify the last host; h12. The “d” flag specifies the duration of the script to run in seconds, and I have set it at 10000 seconds.

Changing the variables in this part of the simulation will yield in different base entropy values, since the entropy is calculated for the entire network and populating only part of the network will lead to lower number of total probability distributions; which is not the basis of the further testing scenarios that are carried out. It can be noticed that the averages calculated will stay close to 1, as the randomness of the traffic is still implemented, and the ratios are still uniform.

After execution, the POX controller terminal and the xterm for h1 will look like the following:

The screenshot displays a Kali Linux desktop environment. The top panel shows the date and time as 8 Aug 14:53. The main workspace contains several windows:

- Terminal Window:** The title bar reads "root@UbuntuSDN:/home/sdnbox/Desktop". The terminal output shows the execution of a Python script named `pox.py` with the `-u` flag. The script connects to a POX 0.8.0 instance on `10.0.0.0`. It then displays Shannon's entropy and Renyi's Entropy for traffic at different alpha values (1.5 and 0.5). The output shows that Shannon's entropy is constant at 1.000000, while Renyi's Entropy varies slightly between 1.000000 and 1.433640 depending on the alpha value.
- Packet Capture Windows:** Several windows titled "Node: h*", "Node: h1", "Node: h2", and "Node: h3" are open, showing network traffic details. These windows display packet numbers, source/destination IP addresses, ports, and protocols. For example, the "Node: h1" window shows a sequence of UDP packets from `10.0.0.1` to `10.0.0.1` on port `51401`.

The bottom of the screen shows the standard Linux taskbar with various application icons and system status indicators.

The entropy values are immediately starting to get calculated after the script gets executed, until then the POX controller is idle. The h1 xterm window will display the packet details. The averages of the ratios of the entropy values are not displayed until the first three time, as the deque is not filled yet.

4.2) Attack scripts – UDP Flooding

The commands for executing each of the scripts are as follows

For executing UDP flood, execute the following command in the attacker host xterm window:

- `python3 udpflood.py targetIPAddress packetrate`

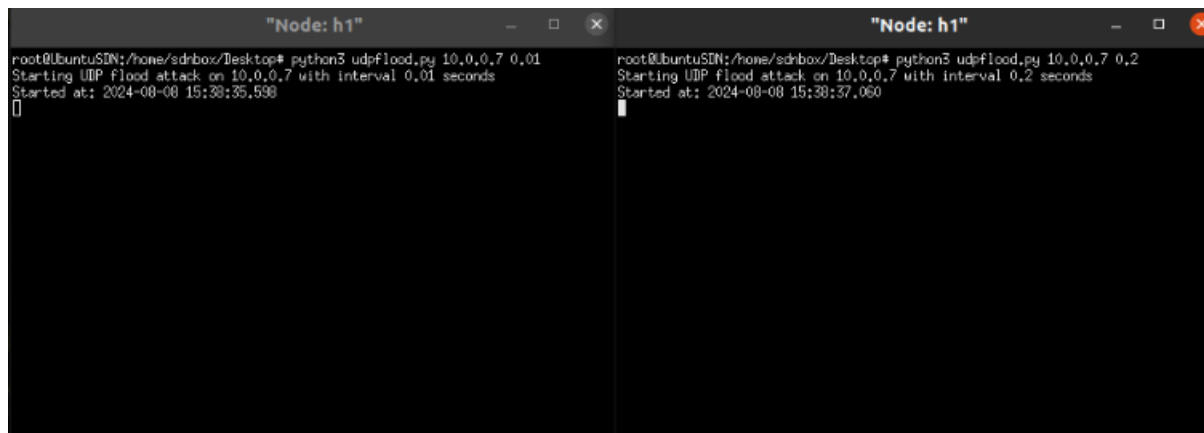
For example, attacking host h7 from host h1 will be done in h1's xterm as follows:

- `python3 udpflood.py 10.0.0.7 0.01`

This will send packets with the interval between them set to 0.01 second, which means at a high-rate of 100 packets per second. The `udpflood.py` script is designed for the purpose of executing both high and low-rate attacks. To start a low-rate attack of 5 packets per second from host h1 targeting h7:

- `python3 udpflood.py 10.0.0.7 0.2`

The attacks should give out the following prompts on the respective xterm windows:



```
"Node: h1"
root@UbuntuSDN:/home/sdrbox/Desktop# python3 udpflood.py 10.0.0.7 0.01
Starting UDP flood attack on 10.0.0.7 with interval 0.01 seconds
Started at: 2024-08-08 15:38:35.598
[]

"Node: h1"
root@UbuntuSDN:/home/sdrbox/Desktop# python3 udpflood.py 10.0.0.7 0.2
Starting UDP flood attack on 10.0.0.7 with interval 0.2 seconds
Started at: 2024-08-08 15:38:37.060
```

Note that the attack scripts don't display packet details, as displaying outputs of high rates can lead to high consumption of system resources, especially when there are multiple xterms.

4.3) Attack scripts – Burst flooding

Another attack script used for the trials is used to execute burst-rate attacks, which sends packet collections at fixed intervals in order to escape existing flooding detection mechanisms. The burst script is executed as follows:

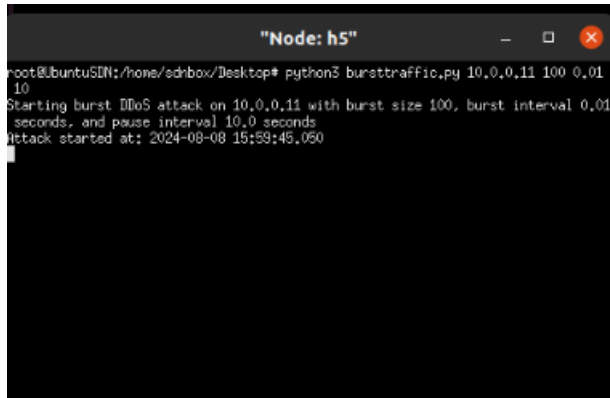
- `python3 bursttraffic.py targetIPAddress packetcollectionsize intervalbetweenpacket burstinterval`

For example, executing this attack from h5 to h11 is done by:

- `python3 bursttraffic.py 10.0.0.11 100 0.01 10`

This will send 100 packets per second at a burst interval of 10 seconds.

Executing this attack should look like this:



```
root@UbuntuSDN:/home/sdnbox/Desktop# python3 bursttraffic.py 10.0.0.11 100 0.01 10
Starting burst DDoS attack on 10.0.0.11 with burst size 100, burst interval 0.01 seconds, and pause interval 10.0 seconds
Attack started at: 2024-08-08 15:59:45.050
```

This attack changes both the entropy and average ratio values, but it is noticed by the difference values calculated of the averages between each entropy calculation window.

It is noticed that the attack scripts display the time of execution once it is started. This time value is used for calculating the detection time of each attack, as the controller logs the point of time when a detection is triggered.

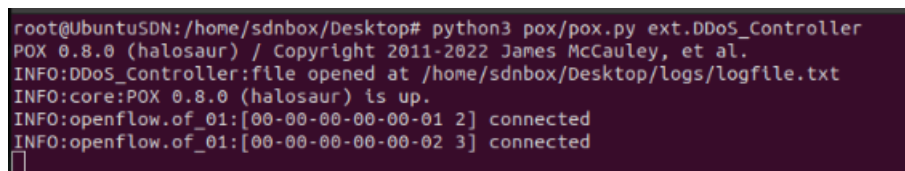
The next section will explain the combinations of attacks executed for studying the detection mechanism.

5 Trials and Logging

The testing scenarios are listed below, and the instructions of each pattern of attack is described. The realtraffic.py script is started before each of the scenarios by using the command:

- `python3 realtraffic.py -s 1 -e 12 -d 10000`

It can be noticed in the POX controller terminal that a log file has been created.



```
root@UbuntuSDN:/home/sdnbox/Desktop# python3 pox/pox.py ext.DDoS_Controller
POX 0.8.0 (halosaur) / Copyright 2011-2022 James McCauley, et al.
INFO:DDoS_Controller:file opened at /home/sdnbox/Desktop/logs/logfile.txt
INFO:core:POX 0.8.0 (halosaur) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:openflow.of_01:[00-00-00-00-00-02 3] connected
```

Make sure that the file located in this path is copied to another location before each trial carried out. This can be used for viewing the calculations.

5.1) Test scenario 1: High-rate attacks to one victim

The attack was executed from hosts h1 h2 h3 h4 to h7. The commands are executed in immediate succession:

- `python3 udpflood.py 10.0.0.7 0.01(from xterm h1)`
- `python3 udpflood.py 10.0.0.7 0.01(from xterm h2)`
- `python3 udpflood.py 10.0.0.7 0.01(from xterm h3)`
- `python3 udpflood.py 10.0.0.7 0.01(from xterm h4)`

5.2) Test scenario 2: High-rate attacks to 4 victims

This attack was executed from hosts h1 h2 h3 h4 to h7 h8 h9 h10 respectively. The commands are executed in immediate succession:

- python3 udpflood.py 10.0.0.7 0.01(from xterm h1)
- python3 udpflood.py 10.0.0.8 0.01(from xterm h2)
- python3 udpflood.py 10.0.0.9 0.01(from xterm h3)
- python3 udpflood.py 10.0.0.10 0.01(from xterm h4)

5.3) Test scenario 3: Low-rate attacks to 1 victim

The attack was executed from hosts h1 h2 h3 h4 to h7. The commands are executed in immediate succession:

- python3 udpflood.py 10.0.0.7 0.2(from xterm h1)
- python3 udpflood.py 10.0.0.7 0.2(from xterm h2)
- python3 udpflood.py 10.0.0.7 0.2(from xterm h3)
- python3 udpflood.py 10.0.0.7 0.2(from xterm h4)

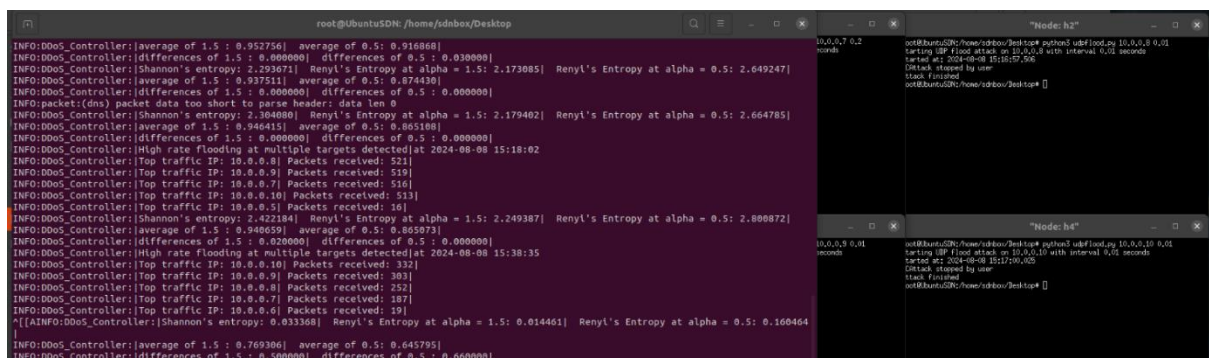
5.4) Test scenario 4: Low-rate attacks to 4 victims

This attack was executed from hosts h1 h2 h3 h4 to h7 h8 h9 h10 respectively. The commands are executed in immediate succession:

- python3 udpflood.py 10.0.0.7 0.2(from xterm h1)
- python3 udpflood.py 10.0.0.8 0.2(from xterm h2)
- python3 udpflood.py 10.0.0.9 0.2(from xterm h3)
- python3 udpflood.py 10.0.0.10 0.2(from xterm h4)

The trigger for each of these attacks are displayed on the POX terminal.

For example, the POX terminal should display the following during the high-rate flood at multiple victims:



```
root@UbuntuSDN: /home/sdnbox/Desktop
INFO:DDoS_Controller:[average of 1.5 : 0.952756] average of 0.5: 0.916868]
INFO:DDoS_Controller:[differences of 1.5 : 0.000000] differences of 0.5 : 0.030000]
INFO:DDoS_Controller:[Shannon's entropy: 2.293071] Renyi's Entropy at alpha = 1.5: 2.173085] Renyi's Entropy at alpha = 0.5: 2.649247]
INFO:DDoS_Controller:[average of 1.5 : 0.937511] average of 0.5: 0.974430]
INFO:DDoS_Controller:[differences of 1.5 : 0.000000] differences of 0.5 : 0.000000]
INFO:packet:(dns) packet data too short to parse header: data len 0
INFO:DDoS_Controller:[Shannon's entropy: 2.304000] Renyi's Entropy at alpha = 1.5: 2.179402] Renyi's Entropy at alpha = 0.5: 2.664785]
INFO:DDoS_Controller:[average of 1.5 : 0.946415] average of 0.5: 0.865100]
INFO:DDoS_Controller:[differences of 1.5 : 0.000000] differences of 0.5 : 0.000000]
INFO:DDoS_Controller:[High rate flooding at multiple targets detected]at 2024-08-08 15:18:02
INFO:DDoS_Controller:[Top traffic IP: 10.0.0.8] Packets received: 521]
INFO:DDoS_Controller:[Top traffic IP: 10.0.0.9] Packets received: 519]
INFO:DDoS_Controller:[Top traffic IP: 10.0.0.7] Packets received: 516]
INFO:DDoS_Controller:[Top traffic IP: 10.0.0.10] Packets received: 513]
INFO:DDoS_Controller:[Top traffic IP: 10.0.0.5] Packets received: 16]
INFO:DDoS_Controller:[Shannon's entropy: 2.422184] Renyi's Entropy at alpha = 1.5: 2.249387] Renyi's Entropy at alpha = 0.5: 2.806872]
INFO:DDoS_Controller:[average of 1.5 : 0.940659] average of 0.5: 0.865073]
INFO:DDoS_Controller:[differences of 1.5 : 0.020000] differences of 0.5 : 0.000000]
INFO:DDoS_Controller:[High rate flooding at multiple targets detected]at 2024-08-08 15:38:35
INFO:DDoS_Controller:[Top traffic IP: 10.0.0.10] Packets received: 332]
INFO:DDoS_Controller:[Top traffic IP: 10.0.0.9] Packets received: 303]
INFO:DDoS_Controller:[Top traffic IP: 10.0.0.8] Packets received: 252]
INFO:DDoS_Controller:[Top traffic IP: 10.0.0.7] Packets received: 187]
INFO:DDoS_Controller:[Top traffic IP: 10.0.0.6] Packets received: 19]
[[[INFO:DDoS_Controller:[Shannon's entropy: 0.933358] Renyi's Entropy at alpha = 1.5: 0.014461] Renyi's Entropy at alpha = 0.5: 0.160464]
INFO:DDoS_Controller:[average of 1.5 : 0.769306] average of 0.5: 0.645795]
INFO:DDoS_Controller:[differences of 1.5 : 0.500000] differences of 0.5 : 0.660000]
```

The thresholds are set in the controller component code shared in the artefacts section. The upcoming tests are conducted after estimating thresholds for them based on the previous thresholds obtained.

5.5) Test scenario 5: Incremental multi-vector attack

The attack is carried out in gradual stages, targeting 2 systems at a time. The attack starts from h1 and h2, targeting h7 and h8. Host h1 executes a high-rate attack, while h2 executes low-rate. After initial detection, the second level is executed from h3 and h4 targeting h9 and h10 in the same pattern, followed by h5 and h6 targeting h11 and h12. The time between attacks was not decided, and was unequal.

The steps for execution are as follows:

- `python3 udpflood.py 10.0.0.7 0.01`(from xterm h1)
- `python3 udpflood.py 10.0.0.8 0.2`(from xterm h2)

Second level is executed with:

- `python3 udpflood.py 10.0.0.9 0.01`(from xterm h3)
- `python3 udpflood.py 10.0.0.10 0.2`(from xterm h4)

Third level:

- `python3 udpflood.py 10.0.0.11 0.01`(from xterm h5)
- `python3 udpflood.py 10.0.0.12 0.2`(from xterm h6)

5.6) Test scenario 5: Multi-vector attack with burst fire

Hosts h1 h2 and h3 h4 target h7 h8 and h9 h10 respectively with high and low vector attacks, and host h5 is used to attack h11 with the burst traffic script. The first two multi-vector attacks are shortly separated in time, and the final burst attack is conducted after the entropy values are stabilized during the flood attack.

The steps for execution are as follows:

- `python3 udpflood.py 10.0.0.7 0.01`(from xterm h1)
- `python3 udpflood.py 10.0.0.8 0.2`(from xterm h2)

Continued by:

- `python3 udpflood.py 10.0.0.9 0.01`(from xterm h3)
- `python3 udpflood.py 10.0.0.10 0.2`(from xterm h4)

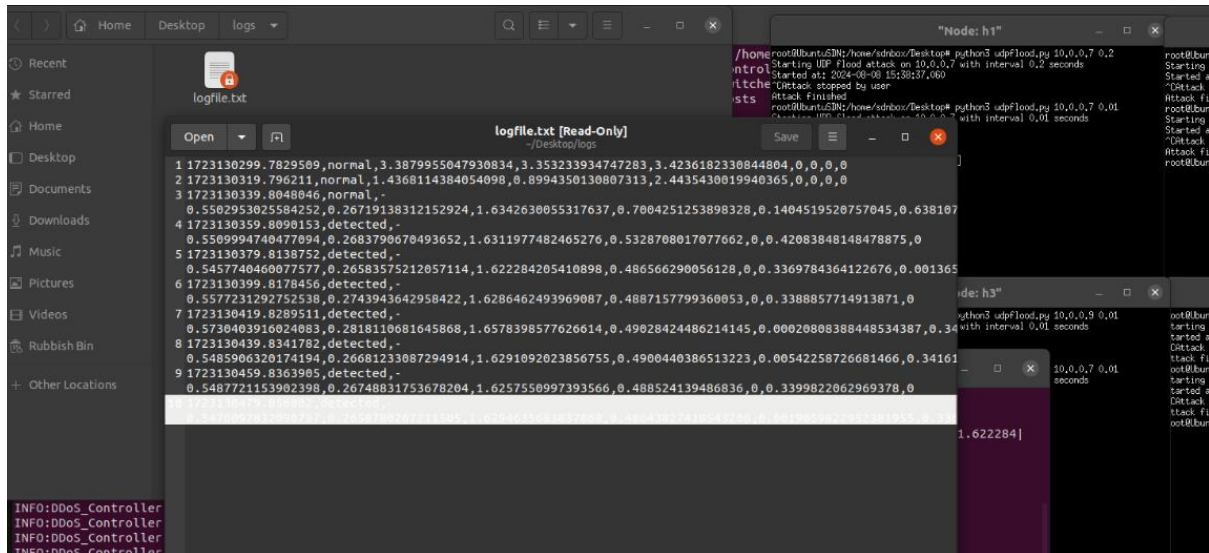
with the final attack:

- `python3 bursttraffic.py 10.0.0.11 100 0.01 10` (from xterm h5)

The difference values between the calculated ratio averages in each time window are used for the purpose of detecting differences flooding and bursting in this test scenario.

5.7) Log files

During each test scenario, a log file is saved in txt format, which can be used to view the values generated at each calculation window. The log files are used for designing the graphs to explain the values in the main report. The log files analysed for the report are shared with the artefacts as well. Log files are supposed to look like this:



The screenshot displays a file manager window with a sidebar on the left showing 'Recent', 'Starred', 'Home', 'Desktop', 'Documents', 'Downloads', 'Music', 'Pictures', 'Videos', 'Rubbish Bin', and 'Other Locations'. The main pane shows a file named 'logfile.txt' with a 'Read-Only' status. The file content is a list of numerical data points, some followed by status labels like 'normal', 'detected', or 'normal,-'. Below the file manager, there are two terminal windows. The top terminal window, titled 'Node: h1', shows a command prompt where a user has run a Python script 'udpflood.py' with arguments '10.0.0.7 0.2'. The output indicates the script started at 2024-08-08 15:38:37.060 and was stopped by the user. The bottom terminal window, titled 'Node: h3', shows a similar command prompt where the user has run 'python3 udpflood.py 10.0.0.9 0.01' with an interval of 0.01 seconds. The output shows the script started at 2024-08-08 15:38:37.060 and was stopped by the user. The bottom terminal window also shows a command prompt where the user has run 'python3 udpflood.py 10.0.0.7 0.01' with an interval of 0.01 seconds. The output shows the script started at 2024-08-08 15:38:37.060 and was stopped by the user.

Details on the calculation and logic implemented for the POX controller are explained in depth in the final report.

ORACLE (2019). *Downloads – Oracle VM VirtualBox*. [online] Virtualbox.org. Available at: <https://www.virtualbox.org/wiki/Downloads>

releases.ubuntu.com. (n.d.). *Ubuntu 20.04.6 LTS (Focal Fossa)*. [online] Available at: <https://releases.ubuntu.com/focal>

mininet.org. (n.d.). *Download/Get Started With Mininet - Mininet*. [online] Available at: <https://mininet.org/download>

Mininet Project Contributors (2021). *Mininet Walkthrough - Mininet*. [online] Mininet.org. Available at: <https://mininet.org/walkthrough/>

noxrepo.github.io. (n.d.). *Installing POX — POX Manual Current documentation*. [online] Available at: <https://noxrepo.github.io/pox-doc/html/>

scapy.readthedocs.io. (n.d.). *Welcome to Scapy's documentation! — Scapy 2.4.3. documentation*. [online] Available at: <https://scapy.readthedocs.io/en/latest/>