CHAPTER FIVE

RESULTS AND DISCUSSIONS

5.1 INTRODUCTION

This chapter presents the performance evaluation of the proposed scheme (elapsed-time based scheme for detecting and mitigating DDoS attacks in the SDN environment). Three extensive experiments have been carried out to evaluate the proposed scheme's performance, in execution time, under different experimental scenarios (different DDoS types, different scales of attack traffic generation in short and long terms). Thereafter, an explanation about the results obtained from each experiment is carried out respectively. Performance evaluation is provided after each experiment. The results of the mixture traffic scenario in each experiment are then compared with the results of all solutions given in chapter two table 2.4 for validation. The comparison is conducted based on the performance metrics namely overhead and accuracy. The false alarms produced are also compared to all previous works in the same comparison tables. The evaluation aims to show that the proposed scheme is able to provide better performance (high accuracy with very low false alarms and low measured overhead) under all traffic states.

According to (Behal & Kumar, 2016), it has been observed that there is no appropriate dataset available for validating the DDoS research among the various publically available datasets because they suffer from a number of limitations. Most

of the available datasets are obsolete, lack the required characteristics of network traffic or not made available to the research community for security reasons.

Another observation from the various solutions that have been proposed to detect DDoS attacks is that datasets are mostly used in the Machine Learning based solutions.

According to (Yavanoglu & Aydos, 2017), datasets are used in Machine Learning and Artificial Intelligent since they are the primary tools for analyzing network traffic and detecting abnormalities.

5.2 SIMULATION EXPERIMENTS

In this section, three experiments are conducted to evaluate the performance of the proposed scheme each of which has a different scenario. The scenarios applied in the three experiments are UDP flood scenario, low rate SYN scenario and a mixture of UDP and SYN DDoS attacks scenario. The first and second experiments have three different test cases which are normal traffic generation, attack traffic generation and a mixture of normal and attack traffic generation. The third experiment has one test case which is a mixture of normal traffic and UDP and SYN attack traffic. The three experiments are discussed in the following subsections.

5.2.1 Experiment One (UDP Flood Scenario)

In a UDP Flood, DDoS attackers send highly-spoofed UDP (user datagram protocol) packets at a very high packet rate using a large source IP range. The victim's network is overwhelmed by the large number of incoming UDP packets. This attack normally consumes network resources and available bandwidth, exhausting the network until it goes offline.

5.2.1.1 Test Case One (Normal Traffic Generation)

Normal traffic means noticing no variations happening suddenly in the network traffic as long as we know the maximum limit of the traffic normally used in the network.

In this test case, normal traffic will be generated from multiple hosts using Python scripts. Python scripts are running on these hosts manually and separately. As appeared in Figure 5.1, no indications of any sudden increases in the network traffic.

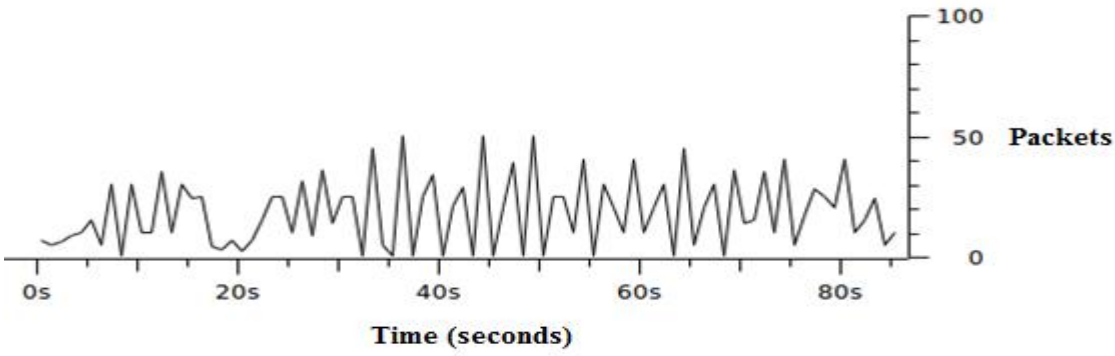
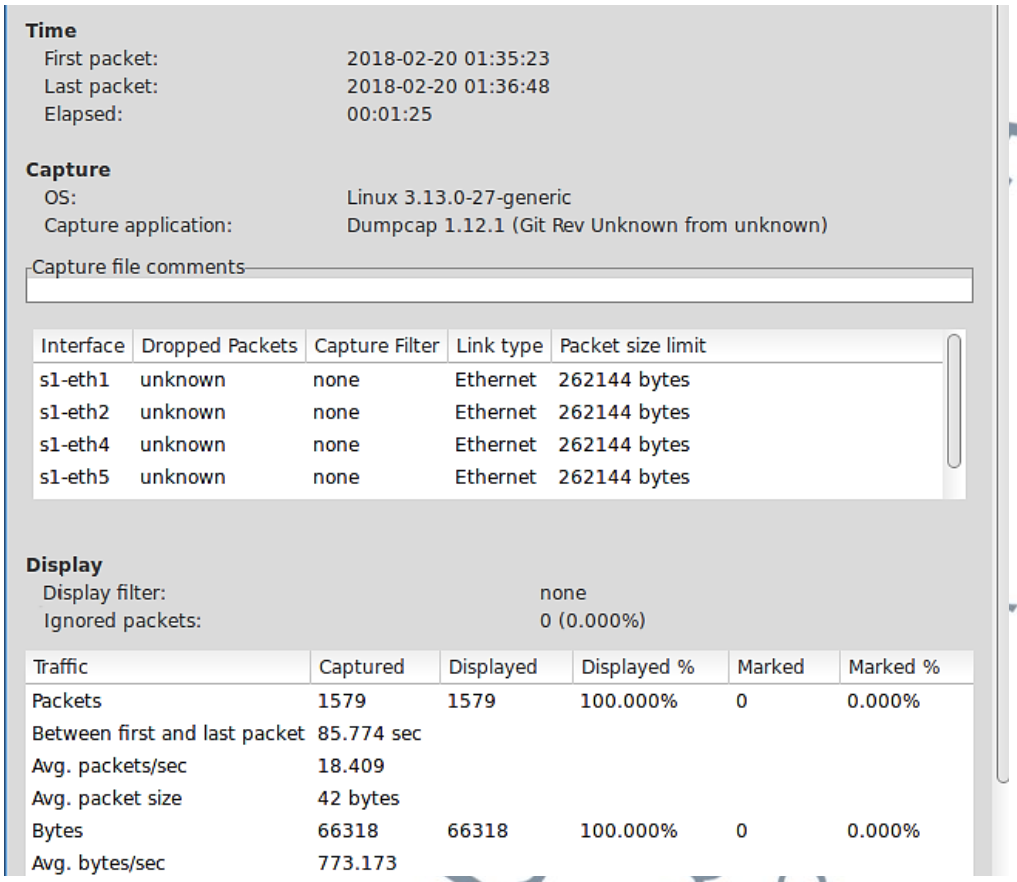


Figure 5.1: The performance under normal UDP traffic

The traffic is generated from five hosts and the time estimated for generation is one minute and twenty five seconds as shown in Figure 5.2. During this time we monitor the reaction of the scheme against these packets that generated from these hosts across the network. Based on the information in Figure 5.2, the traffic generated from these hosts created more than a thousand and five hundred packets in this one minute.



Time

First packet:	2018-02-20 01:35:23
Last packet:	2018-02-20 01:36:48
Elapsed:	00:01:25

Capture

OS:	Linux 3.13.0-27-generic
Capture application:	Dumpcap 1.12.1 (Git Rev Unknown from unknown)

Capture file comments

Interface	Dropped Packets	Capture Filter	Link type	Packet size limit
s1-eth1	unknown	none	Ethernet	262144 bytes
s1-eth2	unknown	none	Ethernet	262144 bytes
s1-eth4	unknown	none	Ethernet	262144 bytes
s1-eth5	unknown	none	Ethernet	262144 bytes

Display

Display filter:	none
Ignored packets:	0 (0.000%)

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	1579	1579	100.000%	0	0.000%
Between first and last packet	85.774 sec				
Avg. packets/sec	18.409				
Avg. packet size	42 bytes				
Bytes	66318	66318	100.000%	0	0.000%
Avg. bytes/sec	773.173				

Figure 5.2: Summary of normal traffic

In Figure 5.3, POX controller terminal shows that the scheme takes no actions against these generated packets because the scheme has realized that this traffic is normal and carries no suspicious packets. Although the number of the packets exceeds one of the scheme thresholds, the scheme takes no actions against these packets. This is proving that the scheme does not produce any false positives.

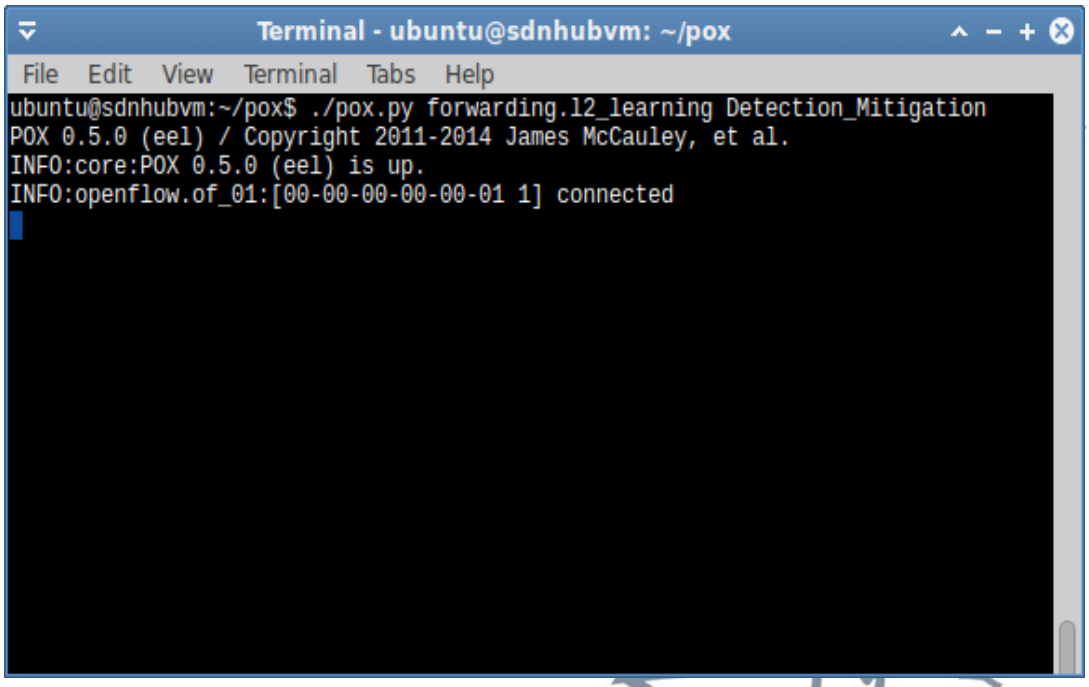


Figure 5.3: Results under normal UDP traffic

5.2.1.2 Test Case Two (Attack Traffic Generation)

In this test case, the attack traffic will be generated on two scales, a scale of low amount of packets and another of huge amount of packets. While the low amount of packets scale is used to generate a few attack packets to look as normal packets, the scale of large amount of packets is used to generate a huge amount of traffic to represent the flooding DDoS attack. The low amount of packets scale and the large amount of packets scale are used to simulate the change in volume of traffic that the attack does when move from high volume to low volume. Python scripts were run from the hosts manually. A tool called Scapy is used to generate UDP attack traffic.

5.2.1.2.1 Low packets scale

When number of packets that travel to a certain destination is low and these packets are coming to this destination continuously then, these packets are considered suspicious packets. This small number of packets will create a large number of flows

to contain them and that will eventually deplete the storage capacity of OpenFlow switch.

Two DDoS scripts are running in this case on two hosts h1 and h2 to generate a low amount of attack packets. After the DDoS scripts begin running, the scheme begins showing the suspicious flows that detected directly as manifested in Figure 5.4.1. These flows will be deleted from the switch once they exceed the threshold (ten flows).

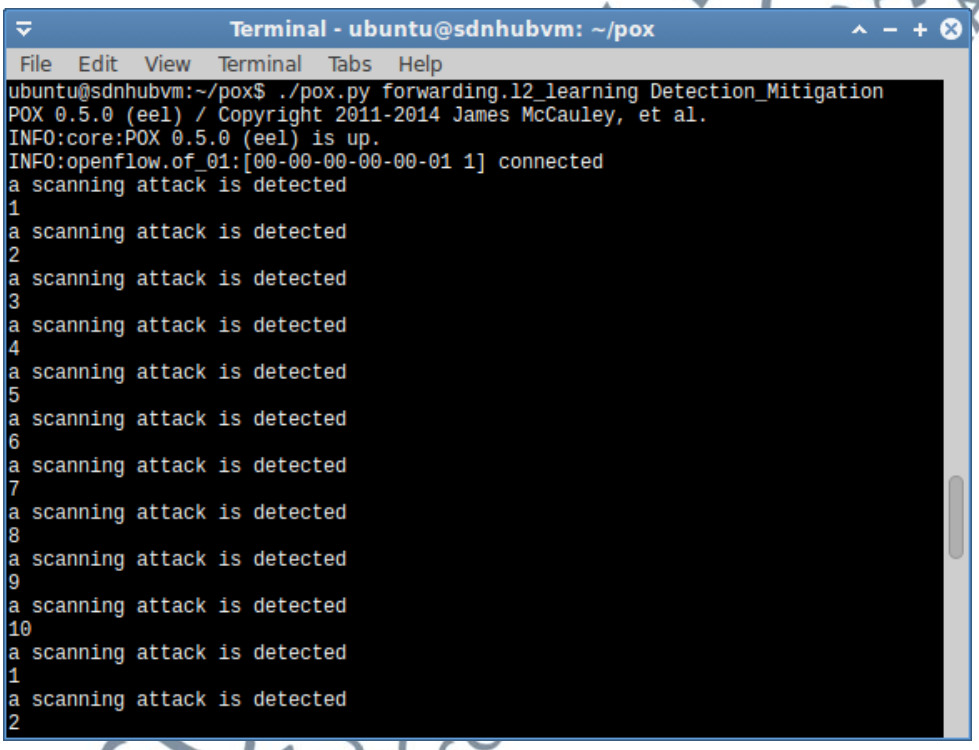


Figure 5.4.1: The first results of the low scale of packets

While the Figure 5.4.2 shows the continuous detecting of the suspicious flows and removing them when they reach ten flows by the scheme, the Figure 5.4.3 shows that the scheme stopped taking any actions when scripts stopped generating traffic. This proves that the scheme does not produce any false positives or false negatives.

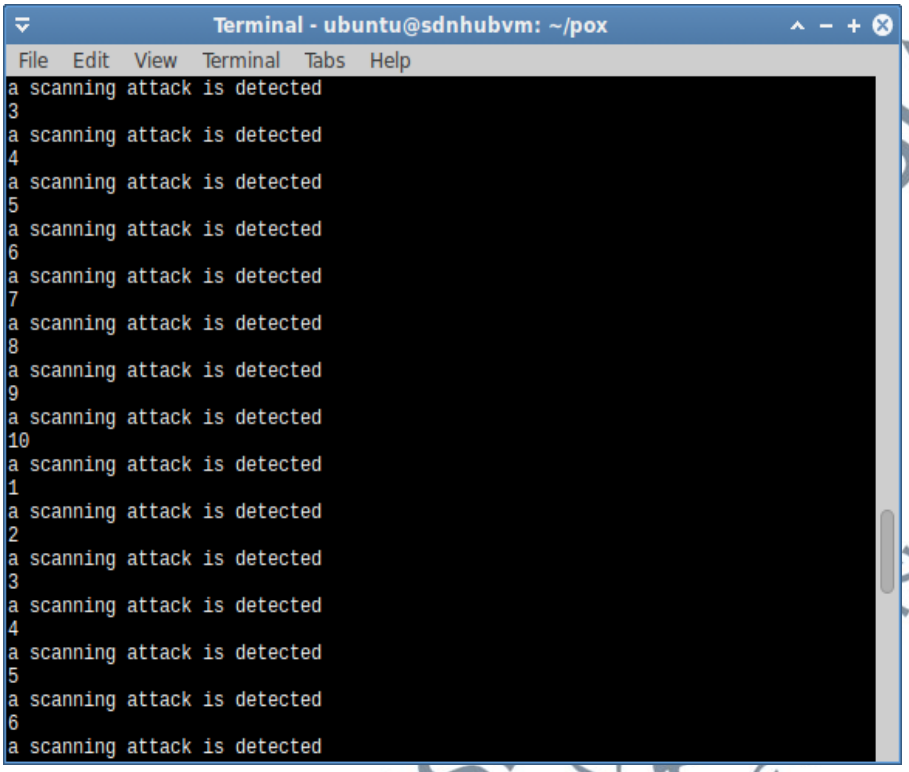


Figure 5.4.2: Results after seventy seconds

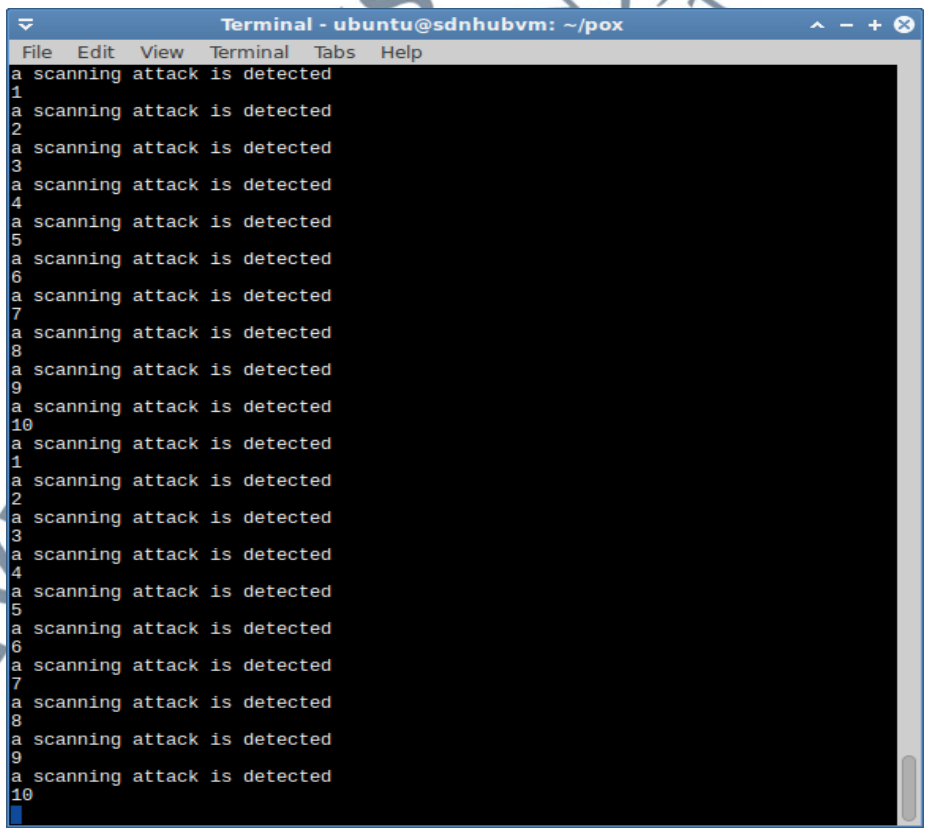


Figure 5.4.3: The last results

To avoid being detected, the attack packets have been sent at a rate of lower than ten packets per second during the time of attack traffic generation as appeared in Figure 5.4.4 but, the scheme has accurately detected the flows that created by the low amount of packets and removing these flows at the time of the attack as shown in Figures 5.4.1, 5.4.2 and 5.4.3. The average of sent packets per second was (7.268 packets/sec) within about two minutes as shown in the Figure 5.4.5. The total number of generated packets during this time was eight hundred and thirty-three packets.

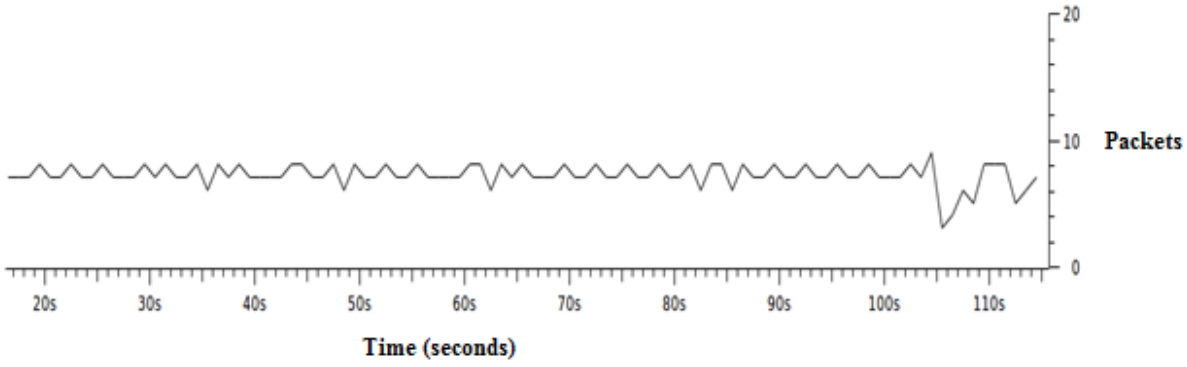
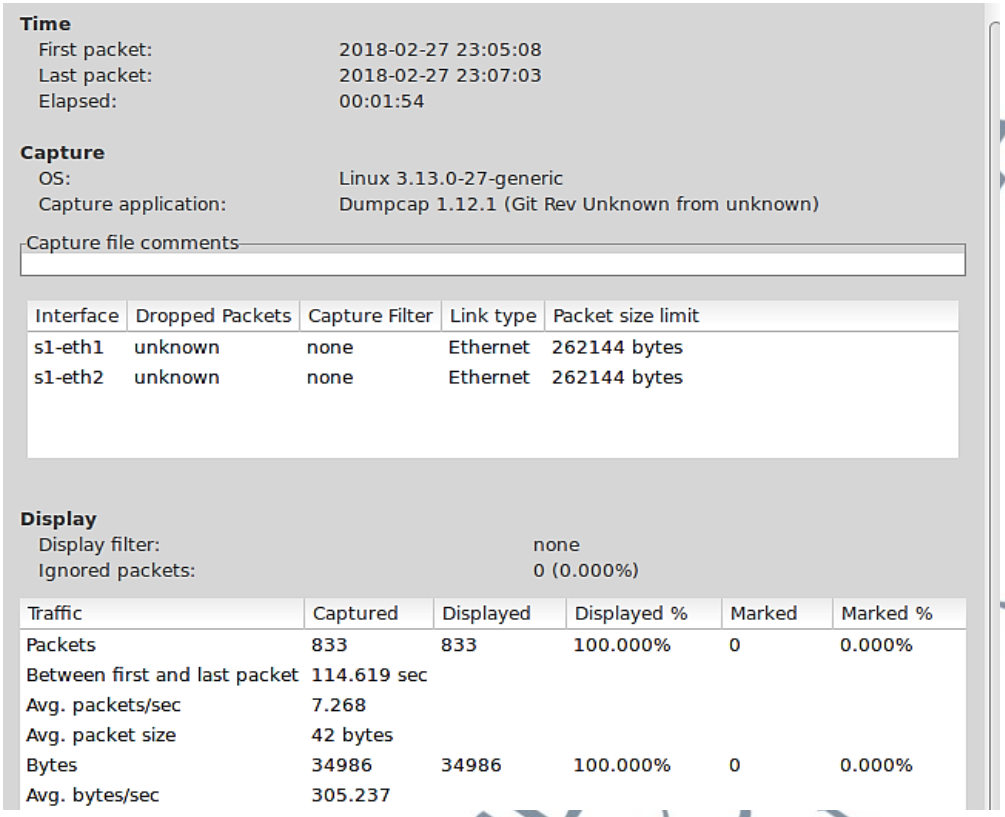


Figure 5.4.4: The Performance under the low scale of packets



Time

First packet:	2018-02-27 23:05:08
Last packet:	2018-02-27 23:07:03
Elapsed:	00:01:54

Capture

OS:	Linux 3.13.0-27-generic
Capture application:	Dumpcap 1.12.1 (Git Rev Unknown from unknown)

Capture file comments

Interface	Dropped Packets	Capture Filter	Link type	Packet size limit
s1-eth1	unknown	none	Ethernet	262144 bytes
s1-eth2	unknown	none	Ethernet	262144 bytes

Display

Display filter:	none
Ignored packets:	0 (0.000%)

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	833	833	100.000%	0	0.000%
Between first and last packet	114.619 sec				
Avg. packets/sec	7.268				
Avg. packet size	42 bytes				
Bytes	34986	34986	100.000%	0	0.000%
Avg. bytes/sec	305.237				

Figure 5.4.5: Summary of the low scale of packets

5.2.1.2.2 Large packets scale

Five hosts are used to emulate the attackers to send UDP flood attack traffic and the sixth host is used to emulate the victim. The idea behind this scenario is to attack the controller with multiple attack machines from the nearest point to the controller to put the scheme under a critical situation.

After running the scheme as it appears in Figure 5.3, we run the DDoS attack scripts from the hosts h1, h2, h3, h5 and h6. As we mentioned before, the host that plays the role of victim is the host h4. Once the attack machines begin generating the attack traffic towards the target machine, the scheme begins gathering the statistical information of the number of flows and packets. As appeared in Figure 5.5.1, the scheme raises a warning of an attack when it notice that the number of

flows is getting increased and the number of packets does not exceed the threshold (1500 packets). The maximum number that these suspicious flows can reach before removing is ten flows. It is important to mention here that the log message from the packet subsystem "(UDP parse) UDP packet data shorter than UDP len" shown in the Figure 5.5.1 is neither an error nor a necessarily indicative of any problem as stated in POX Wiki.

When the attack moves from the low volume to the high volume of attack packets, the scheme stops looking into the flows and begins checking the statistical information gathered about the packets. As we can see in Figure 5.5.1, the scheme is directly noticing that these packets are suspicious as soon as the number of the packets begins to increase rapidly and shows a warning message that says there is an attack detected from the source IP address 10.0.0.6 which is an attack machine on the destination IP address 10.0.0.4 the victim through the protocol seventeen which is the UDP protocol. This means that the detection method in the scheme was activated and it works in a successful way.

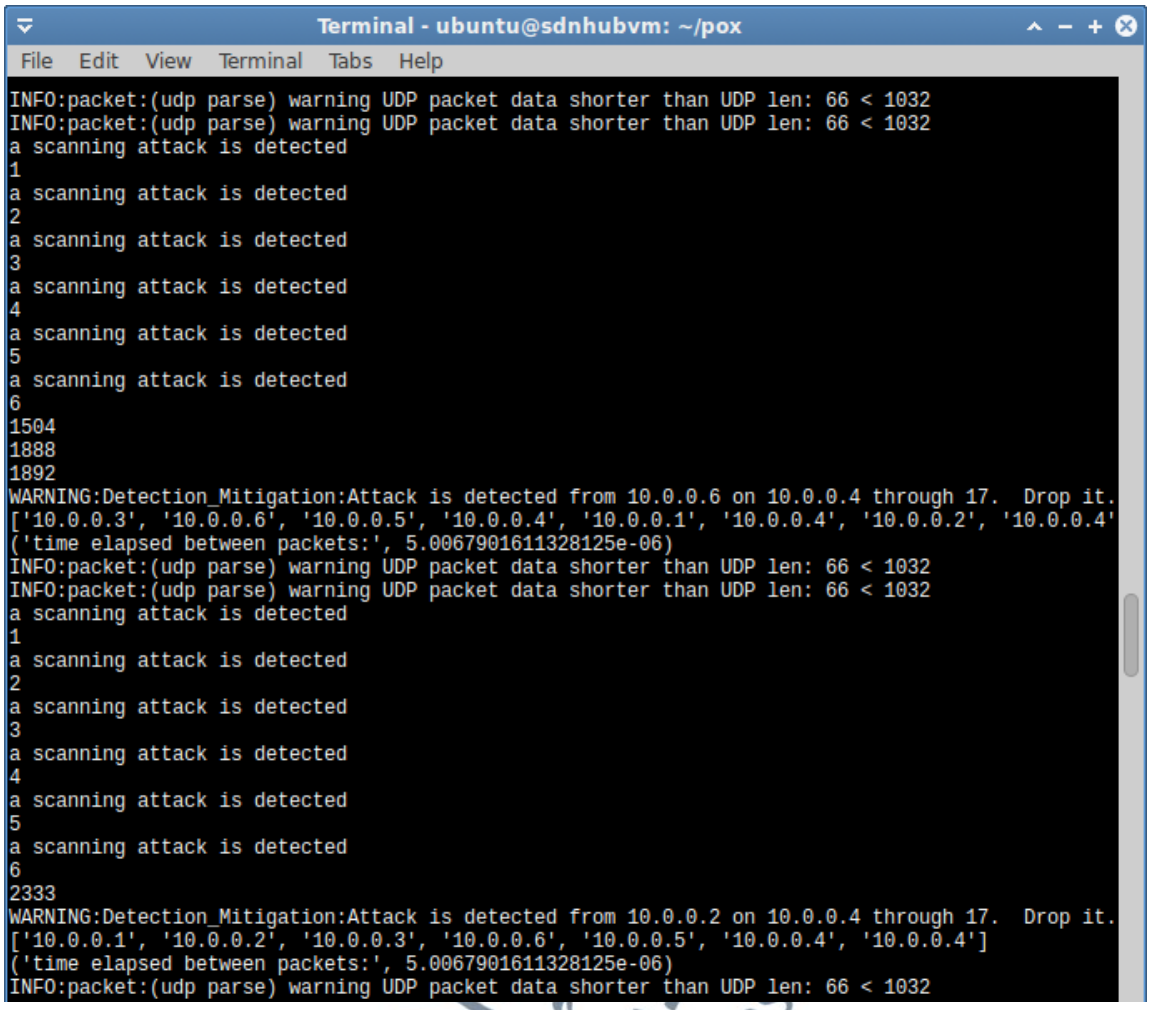


Figure 5.5.1: The first results of the large scale of packets

After the detection work is finished the scheme goes forward to drop these packets using its mitigation method. We can see clearly that the packets have been dropped by noticing that the scheme begins looking into the flows rather than the packets. As we can see, the number of packets has been decremented from one thousand eight hundred and ninety-two packets to lower than the threshold which indicates that the mitigation method is activated and works properly. This is an evidence of the effectiveness of the mitigation function in dropping and terminating the attack packets and flows once they detected. It is important to notice that the number of flows begins from number one which means that the previous flows have been removed. Also, we can notice that the number of the packets goes above the

threshold again and the detection function detects attack packets coming from different source IP address which is 10.0.0.2. The detected packets are directly dropped by the mitigation function.

Figure 5.5.2 shows that the attack traffic is coming from another source IP address to targeting the same victim and again the scheme is accurately detecting and dropping the DDoS attack packets. A list of IP addresses is provided to show where the traffic comes from and goes to.

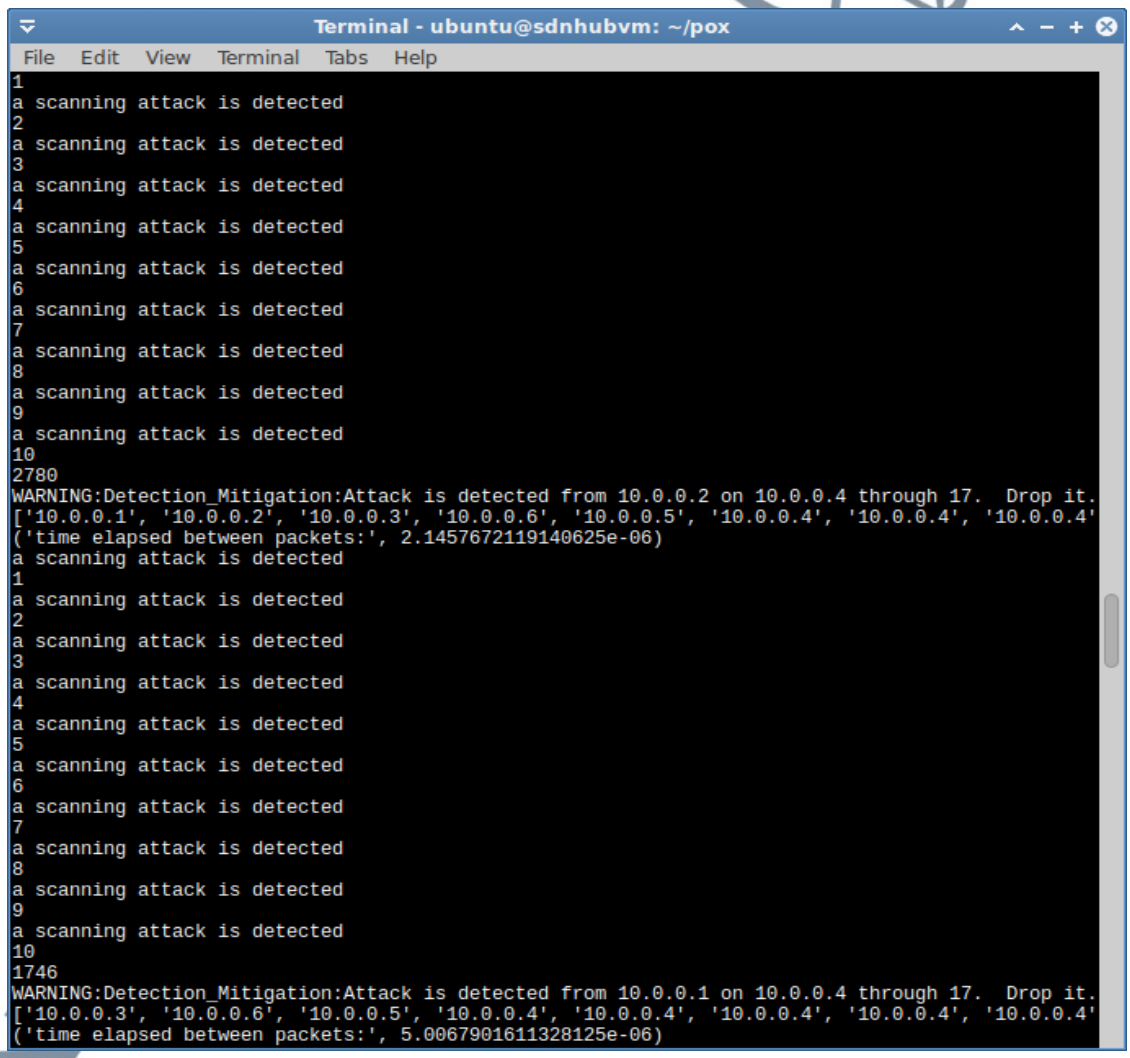


Figure 5.5.2: Results after five minutes

As appeared in Figure 5.5.3, dropping attack packets and removing the suspicious flows is continued by the scheme.

```
Terminal - ubuntu@sdnhubvm: ~/pox
File  Edit  View  Terminal  Tabs  Help
1
a scanning attack is detected
2
a scanning attack is detected
3
a scanning attack is detected
4
1586
2719
WARNING:Detection_Mitigation:Attack is detected from 10.0.0.6 on 10.0.0.4 through 17.  Drop it.
['10.0.0.4', '10.0.0.6', '10.0.0.4', '10.0.0.3', '10.0.0.4', '10.0.0.4']
('time elapsed between packets:', 4.0531158447265625e-06)
1897
3332
WARNING:Detection_Mitigation:Attack is detected from 10.0.0.6 on 10.0.0.4 through 17.  Drop it.
['10.0.0.4', '10.0.0.4']
('time elapsed between packets:', 5.0067901611328125e-06)
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
1701
3182
WARNING:Detection_Mitigation:Attack is detected from 10.0.0.5 on 10.0.0.4 through 17.  Drop it.
['10.0.0.4', '10.0.0.4']
('time elapsed between packets:', 1.0013580322265625e-05)
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 66 < 1032
a scanning attack is detected
1
a scanning attack is detected
2
a scanning attack is detected
3
a scanning attack is detected
4
a scanning attack is detected
5
1835
WARNING:Detection_Mitigation:Attack is detected from 10.0.0.2 on 10.0.0.4 through 17.  Drop it.
['10.0.0.6', '10.0.0.5', '10.0.0.1', '10.0.0.2', '10.0.0.4', '10.0.0.4']
('time elapsed between packets:', 5.0067901611328125e-06)
```

Figure 5.5.3: Results after fifteen minutes

The Figure 5.5.3 shows that the number of packets has been decremented from two thousand seven hundred and nineteen packets to one thousand eight hundred ninety-seven. This means about one thousand packets has been dropped approximately. We can also notice that the number of the packets has sharply increased again because we are under a flooding DDoS attack. The attack packets reach the duple of the threshold in five seconds (time given to collect the statistical

information) and the scheme's detection and mitigation functions take a place to detect and drop these packets.

As clearly demonstrated in Figure 5.5.4, the scheme stays notified of the number of flows that being created when the number of packets is still under the threshold. We also still notice that the traffic is coming from different source IP addresses and the time elapsed between packets is different in each detection process.

Due to sending only partial of packets by the switch to the controller, we see the subsystem log message that says that the UDP packet data shorter than UDP length is displayed from time to time.

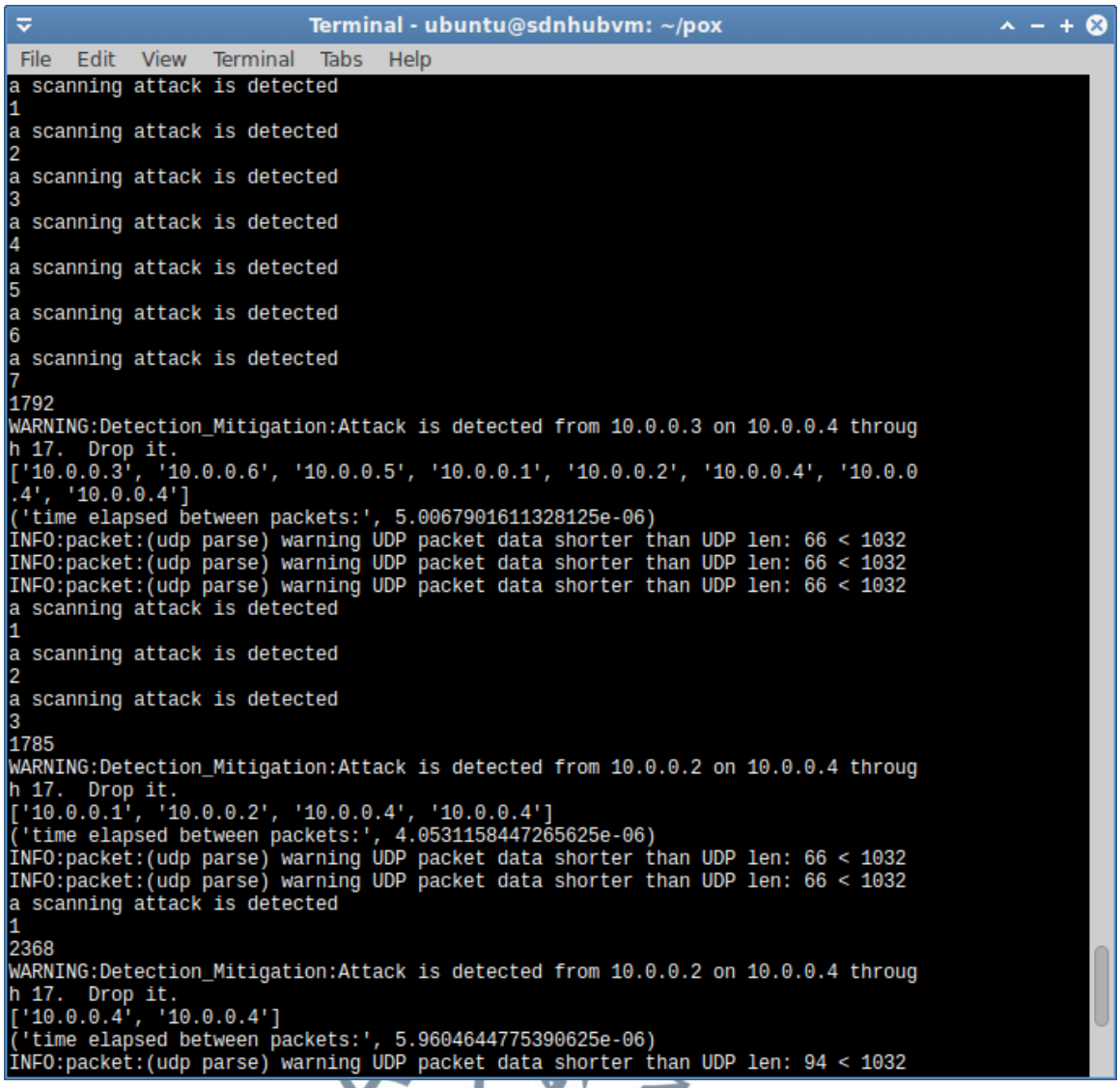


Figure 5.5.4: Results after twenty five minutes

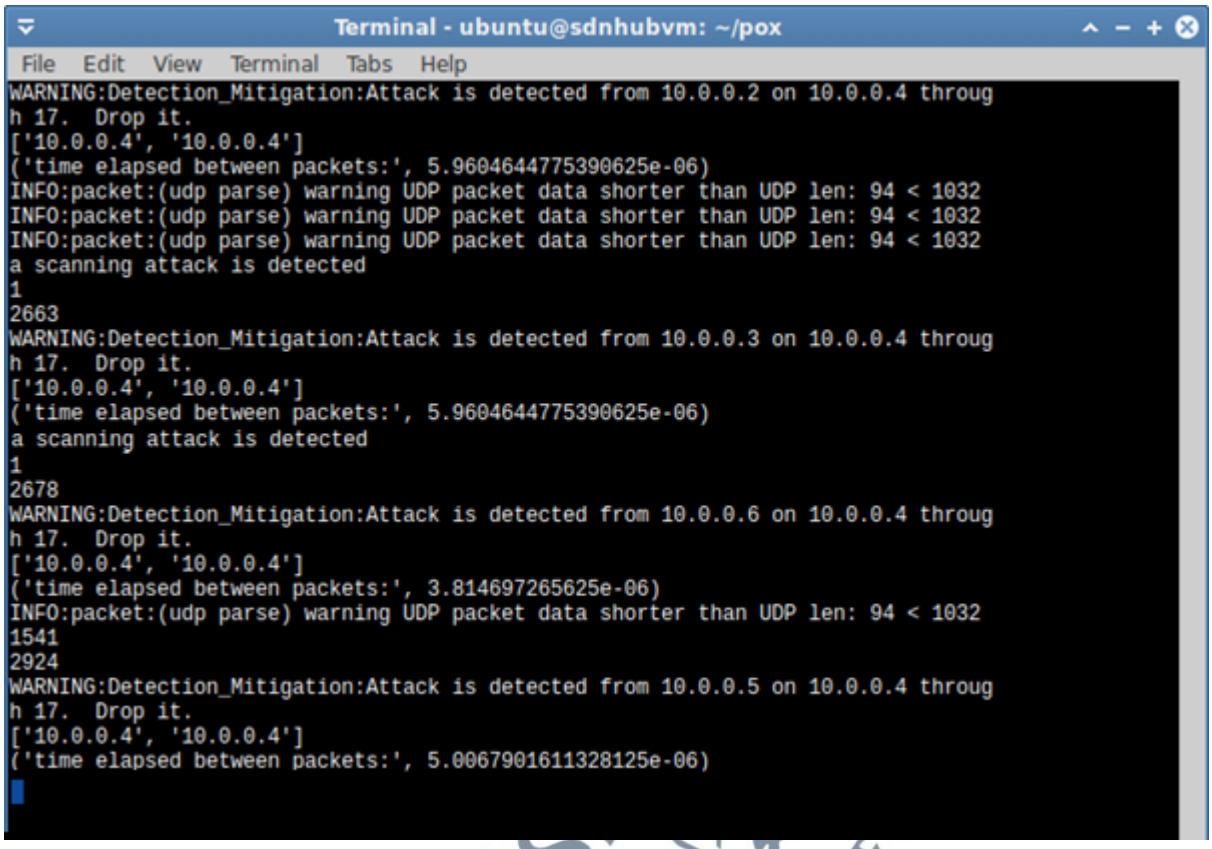


Figure 5.5.5: Results after thirty minutes

Figure 5.5.5 shows the last results have been obtained by the scheme. By monitoring all previous POX terminals, we can see that the functionality of the scheme has neither affected by the increase in the number of attack packets nor by the increase in the number of flows in terms of detecting and mitigating attack packets that change in volume of packets.

The graphs shown in Figure 5.5.6, illustrates the performance of SDN components under a large scale of UDP flood DDoS attack. The X-axis represents the number of packets while the Y-axis represents the time in seconds. The graphs show that the attack is changing from the high volume of the attack packets to the low volume and vice versa during the time of the attack. The graphs also show that no performance degradation has occurred in the performance of SDN components due to this change. Therefore, we can say the scheme has successfully protected the

SDN components from being unavailable to legitimate users while these components under a changeable DDoS attack.

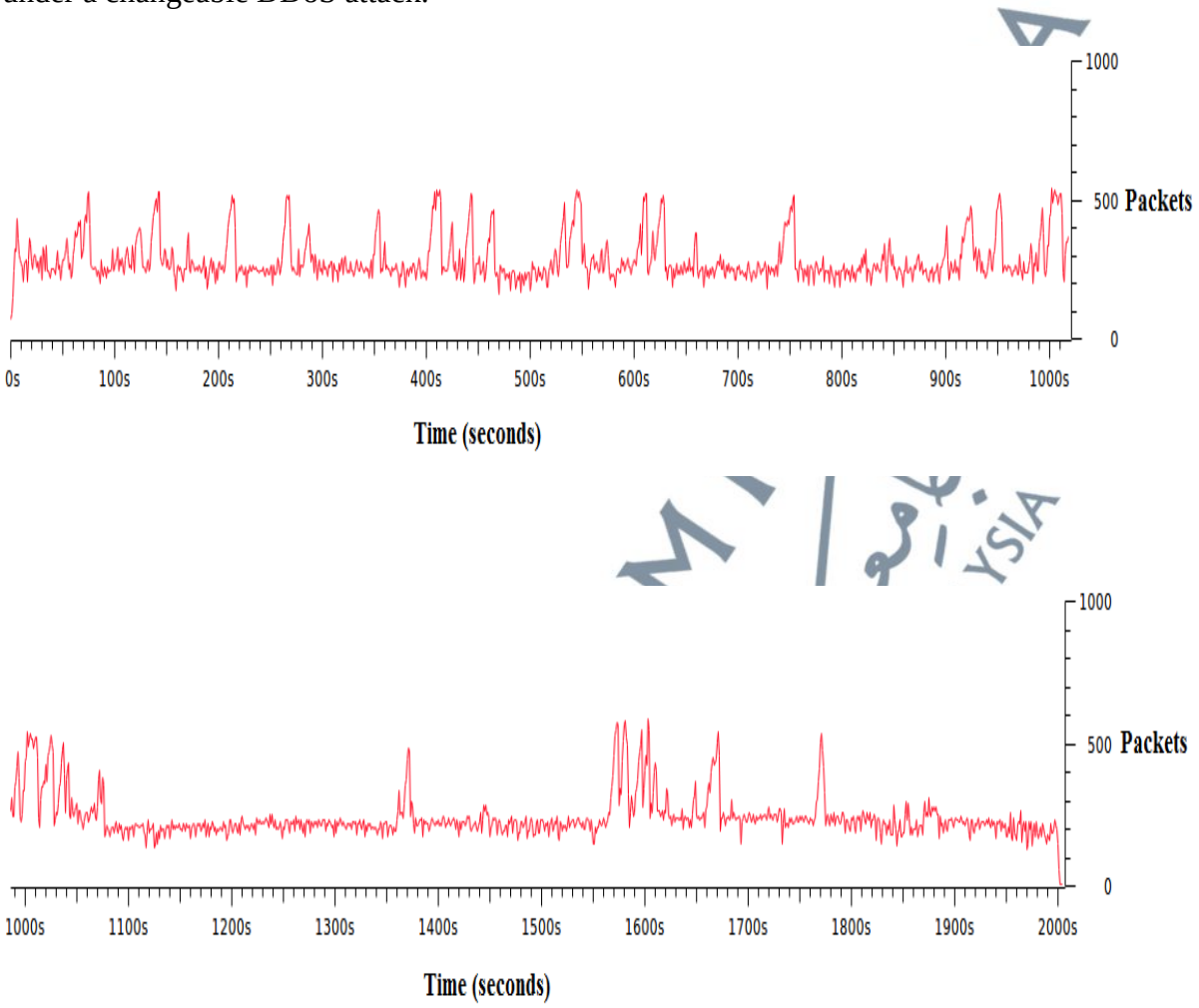
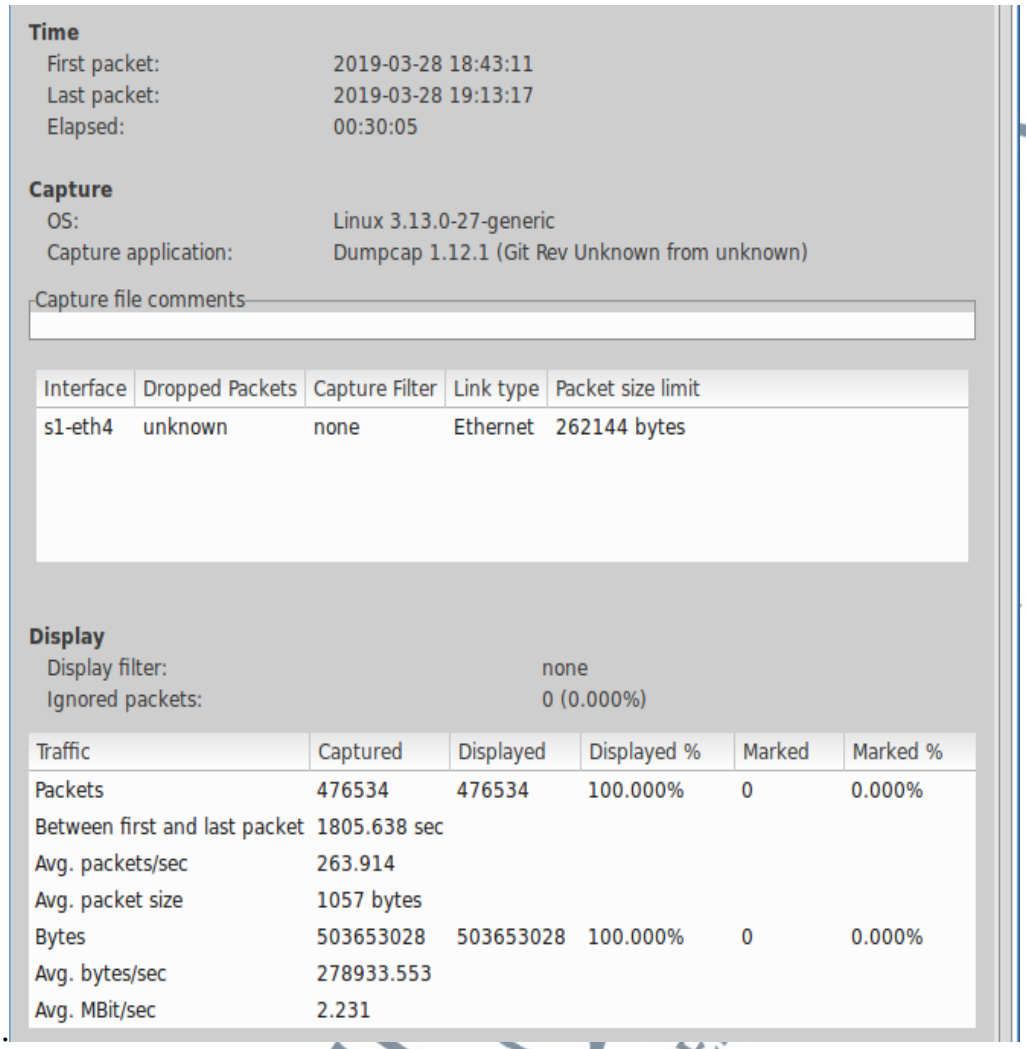


Figure 5.5.6: The performance under the large scale of packets (0-1000s, 1000s-2000s)

The information displayed in Figure 5.5.7 is a summary of the time taken to generate the attack traffic from the first packet to the last packet, interface of the victim machine that has been received the attack traffic, the total number of packets and the average of sent packets per second.



Time

First packet:	2019-03-28 18:43:11
Last packet:	2019-03-28 19:13:17
Elapsed:	00:30:05

Capture

OS:	Linux 3.13.0-27-generic
Capture application:	Dumpcap 1.12.1 (Git Rev Unknown from unknown)

Capture file comments

Interface	Dropped Packets	Capture Filter	Link type	Packet size limit
s1-eth4	unknown	none	Ethernet	262144 bytes

Display

Display filter:	none
Ignored packets:	0 (0.000%)

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	476534	476534	100.000%	0	0.000%
Between first and last packet	1805.638 sec				
Avg. packets/sec	263.914				
Avg. packet size	1057 bytes				
Bytes	503653028	503653028	100.000%	0	0.000%
Avg. bytes/sec	278933.553				
Avg. MBit/sec	2.231				

Figure 5.5.7: Summary of the large scale of packets

5.2.1.3 Test Case Three (Mixture of UDP Flood Attack and Normal Traffic)

In this test case, the attack traffic and normal traffic are mixed together in the same traffic generation. The hosts h3, h4 and h6 are the hosts designated to generate the attack traffic while hosts h2 and h5 are used to generate normal traffic. The host h1 is the victim.

On the POX controller terminal where the scheme is running, nothing displayed although the scripts that generate the normal traffic are currently running.

Figure 5.6 shows the controller terminal within sixty seconds of generating normal traffic.

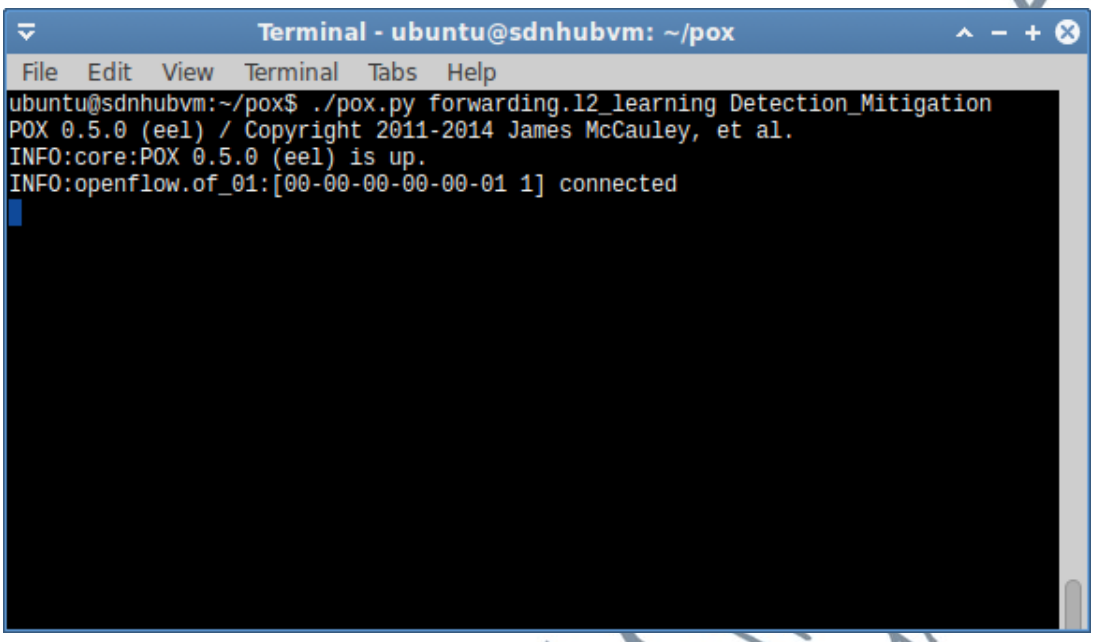


Figure 5.6: POX terminal within the first sixty seconds

After seventy seconds of generating normal traffic, the DDoS UDP flood attack is launched from the attack hosts along with the normal traffic that is still running. As appeared in Figure 5.7, the scheme detects and mitigates the attack flows and packets. Also, we can see in Figure 5.7, the IP addresses of hosts that used to generate the normal traffic do not appear during the time of attack detection. The results appear in Figures 5.7, 5.8, 5.9, 5.10, 5.11 and 5.12 are the results have been appeared after three minutes, five minutes, ten minutes, twelve minutes, fifteen minutes and twenty minutes respectively during the scheme running.

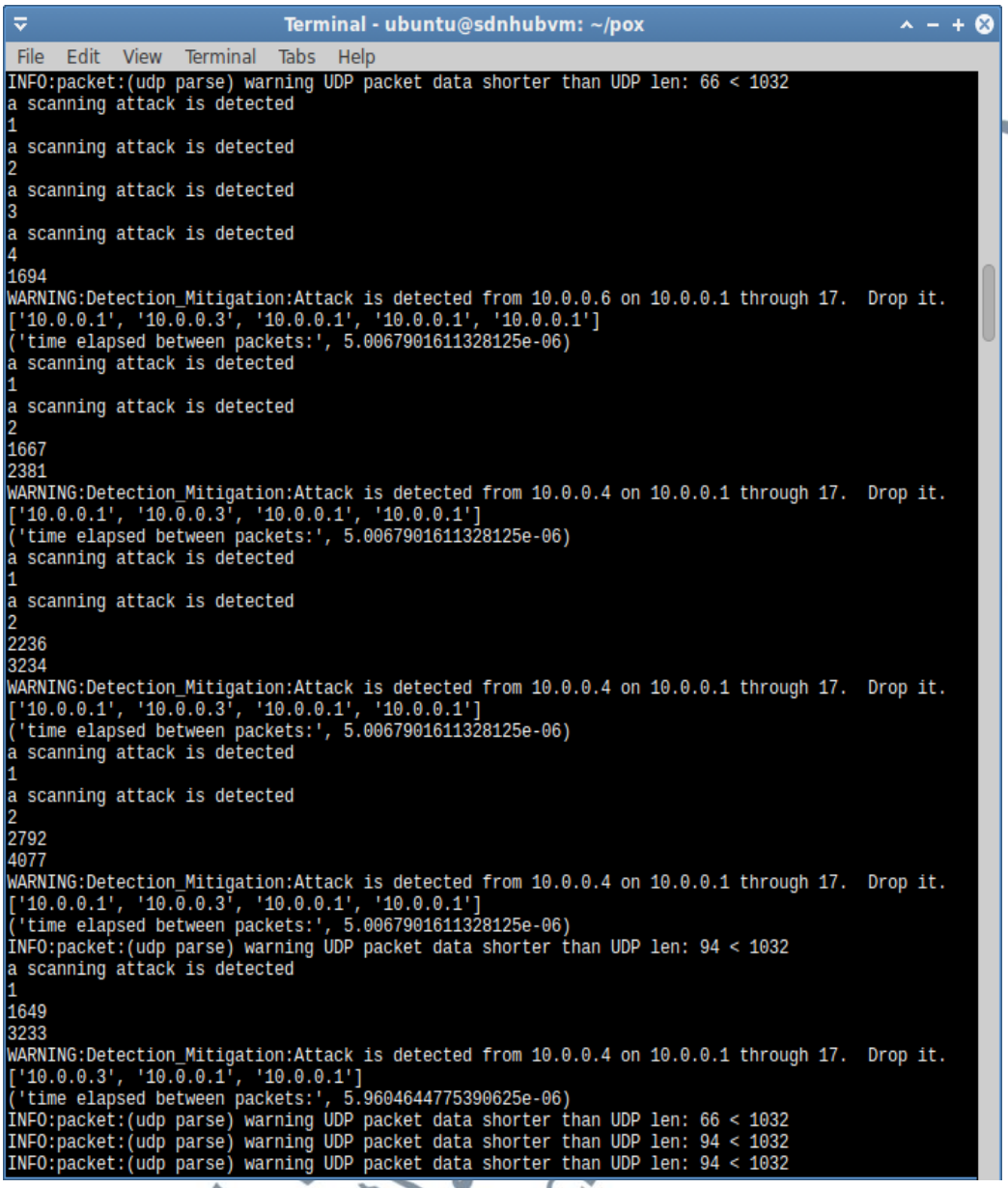


Figure 5.7: Results after three minutes

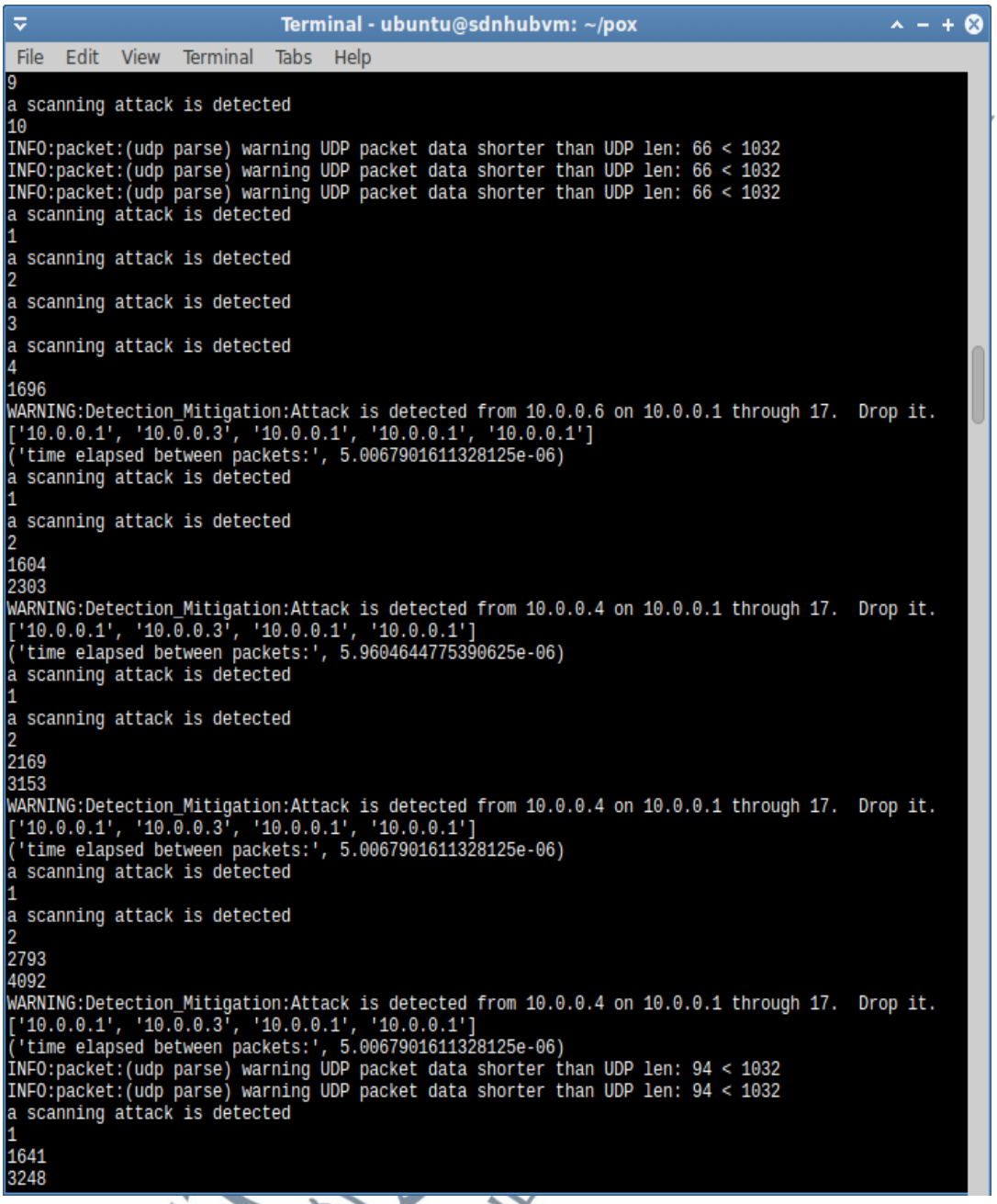


Figure 5.8: Results after five minutes

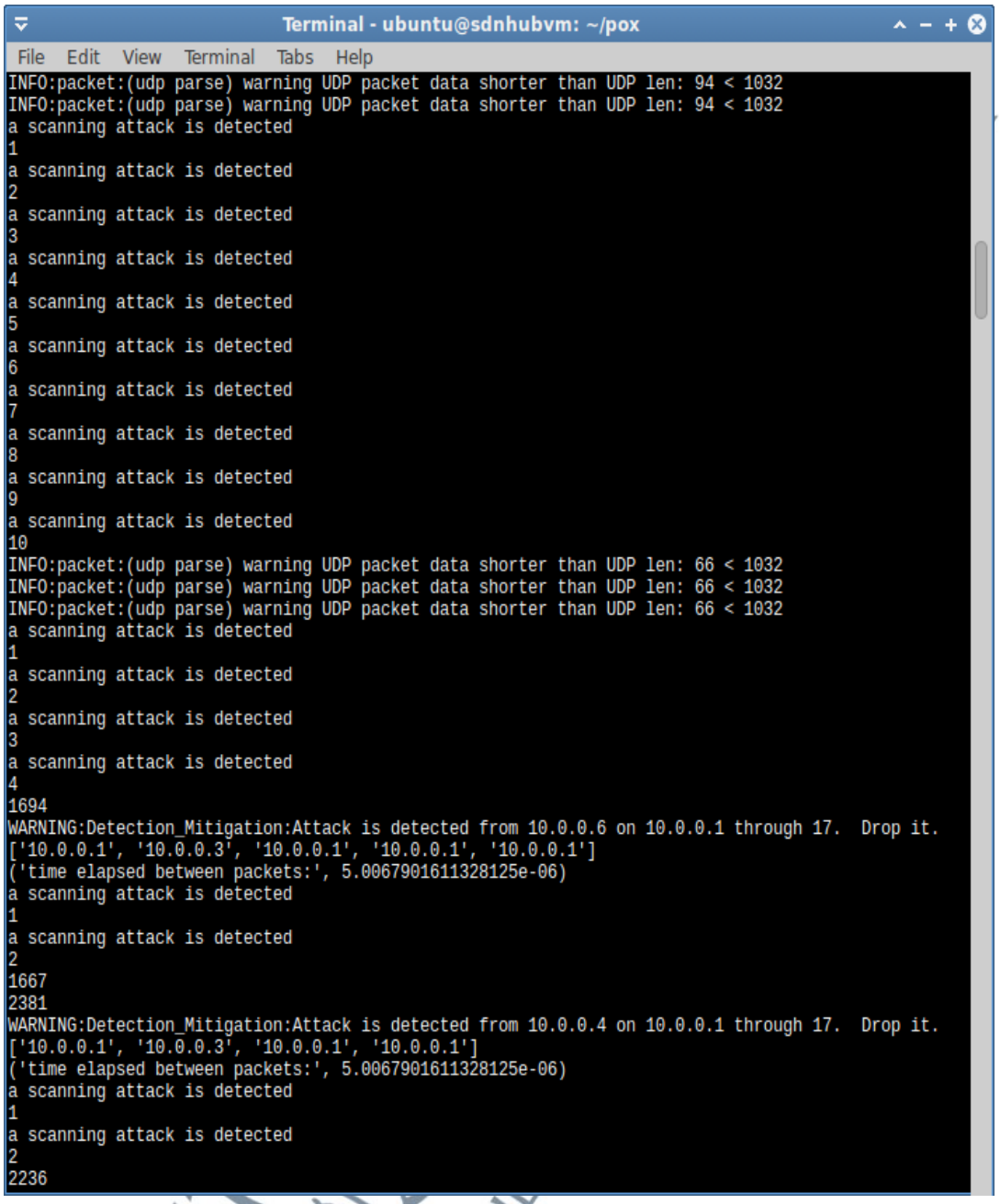


Figure 5.9: Results after ten minutes

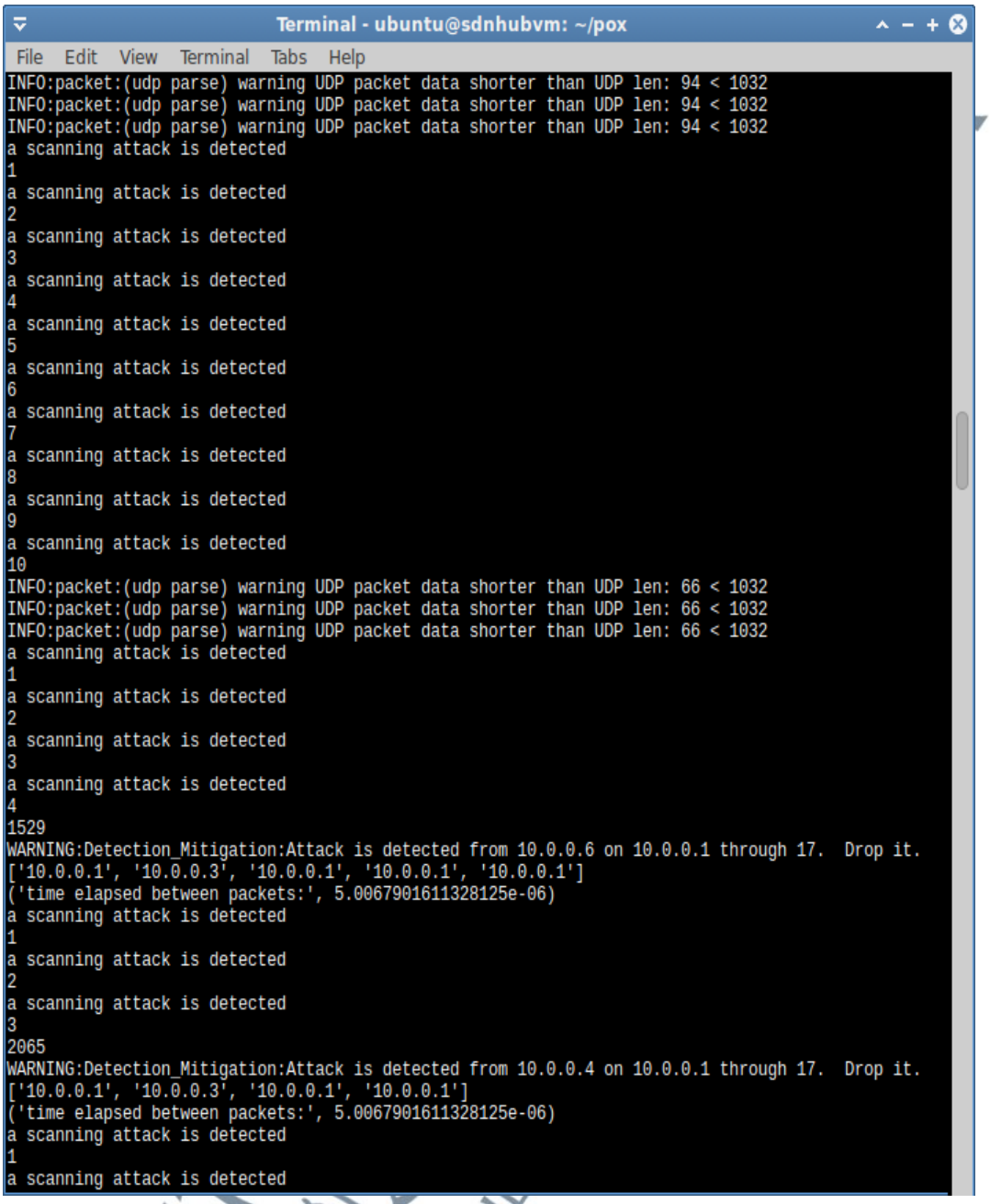
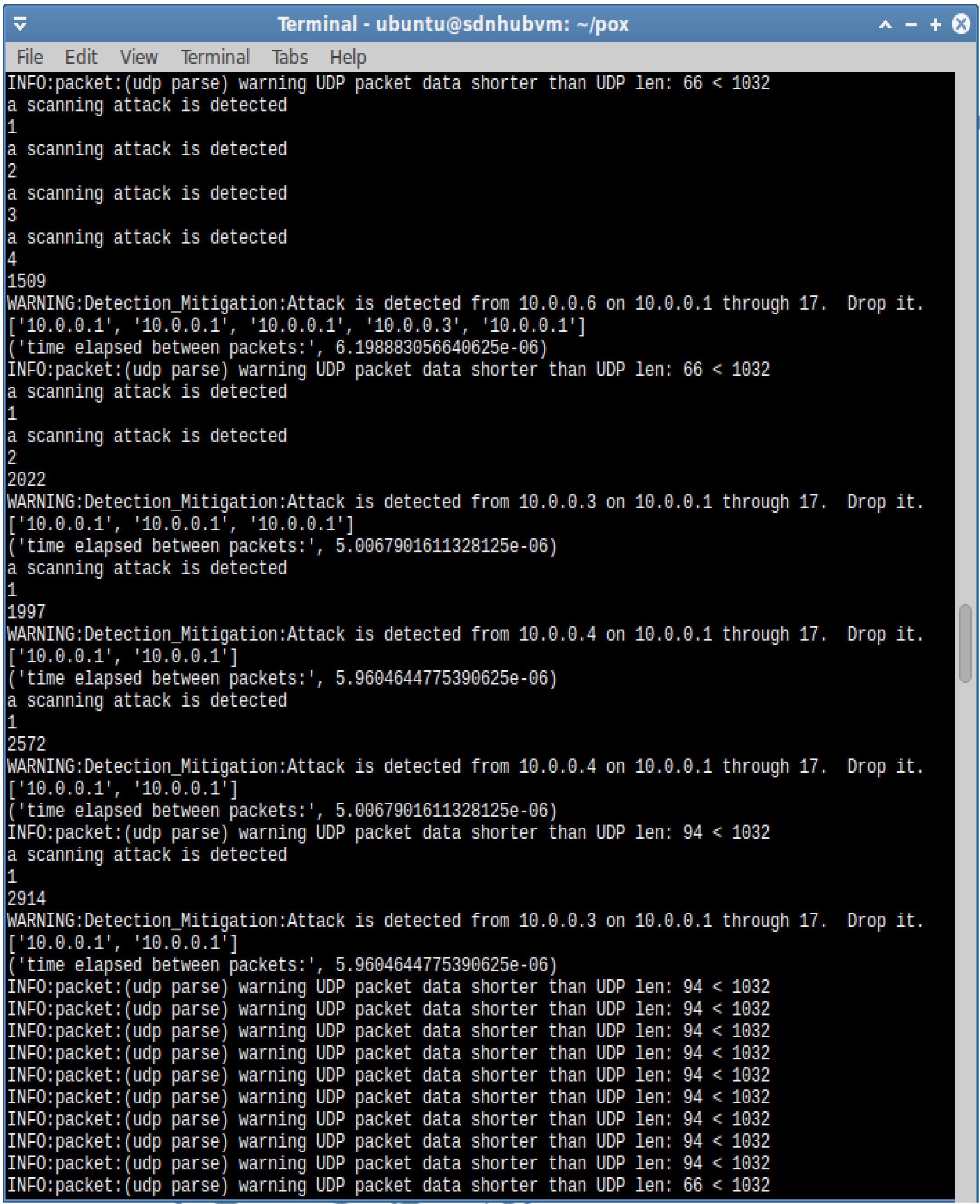


Figure 5.10: Results after twelve minutes



```
Terminal - ubuntu@sdnhubvm: ~/pox
File  Edit  View  Terminal  Tabs  Help
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 66 < 1032
a scanning attack is detected
1
a scanning attack is detected
2
a scanning attack is detected
3
a scanning attack is detected
4
1509
WARNING:Detection_Mitigation:Attack is detected from 10.0.0.6 on 10.0.0.1 through 17.  Drop it.
['10.0.0.1', '10.0.0.1', '10.0.0.1', '10.0.0.3', '10.0.0.1']
('time elapsed between packets:', 6.198883056640625e-06)
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 66 < 1032
a scanning attack is detected
1
a scanning attack is detected
2
2022
WARNING:Detection_Mitigation:Attack is detected from 10.0.0.3 on 10.0.0.1 through 17.  Drop it.
['10.0.0.1', '10.0.0.1', '10.0.0.1']
('time elapsed between packets:', 5.0067901611328125e-06)
a scanning attack is detected
1
1997
WARNING:Detection_Mitigation:Attack is detected from 10.0.0.4 on 10.0.0.1 through 17.  Drop it.
['10.0.0.1', '10.0.0.1']
('time elapsed between packets:', 5.9604644775390625e-06)
a scanning attack is detected
1
2572
WARNING:Detection_Mitigation:Attack is detected from 10.0.0.4 on 10.0.0.1 through 17.  Drop it.
['10.0.0.1', '10.0.0.1']
('time elapsed between packets:', 5.0067901611328125e-06)
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
a scanning attack is detected
1
2914
WARNING:Detection_Mitigation:Attack is detected from 10.0.0.3 on 10.0.0.1 through 17.  Drop it.
['10.0.0.1', '10.0.0.1']
('time elapsed between packets:', 5.9604644775390625e-06)
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 66 < 1032
```

Figure 5.11: Results after fifteen minutes

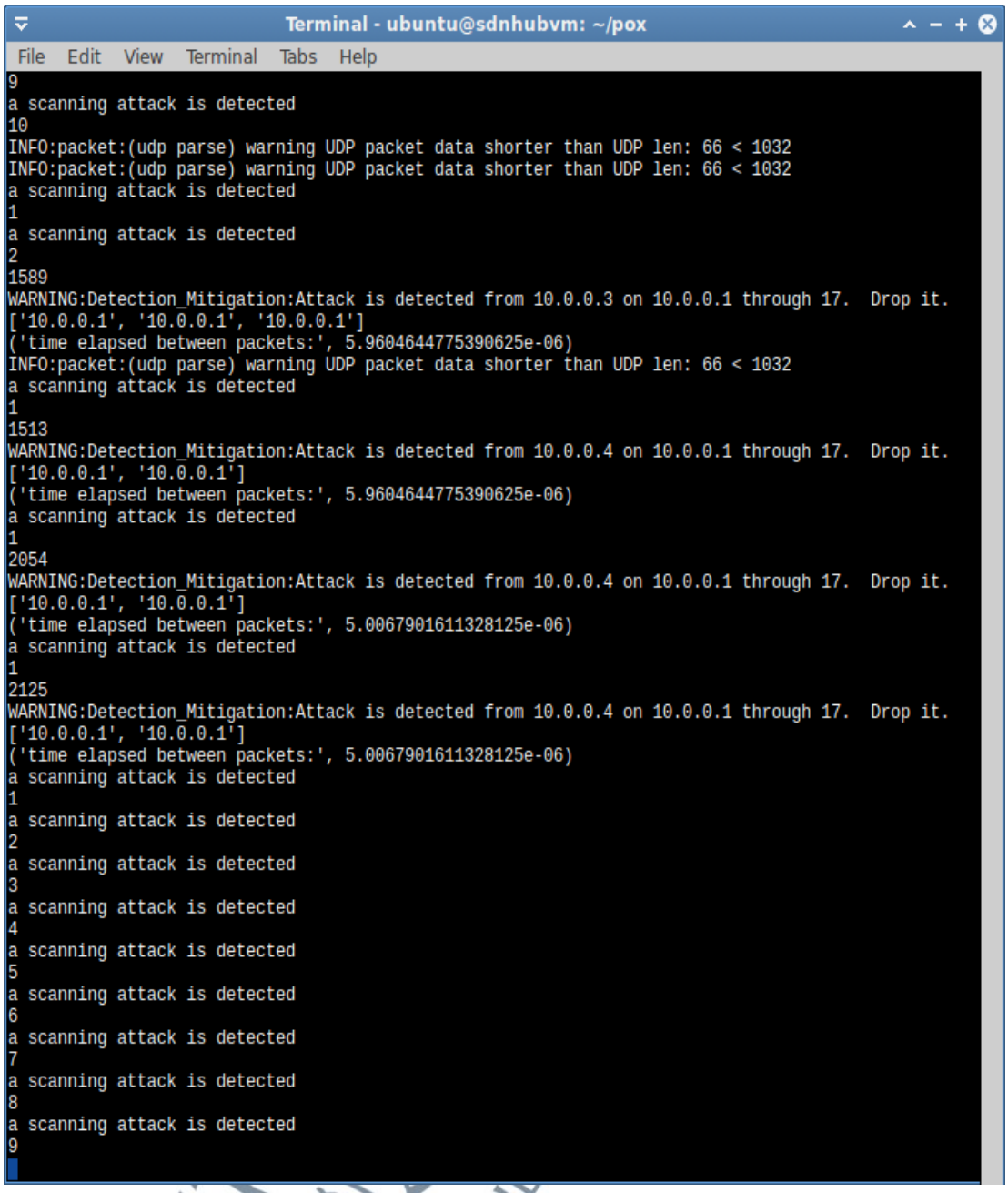


Figure 5.12: Results after twenty minutes

During traffic generation time, the IP addresses of hosts that generate the normal traffic do not appear. This proves that our scheme has not produced any false positives or false negatives.

In all graphs appeared in Figure 5.13, the red line represents the normal traffic and the black line represents the UDP attack traffic. All graphs show that the scheme is successfully detecting the change in attack packets made by the DDoS

attack at the time of traffic generation. This practically proves that the scheme accurately detected the attack packets until the last second of traffic generation. Figure 5.13 shows that the scheme has protected the SDN components during the time of the attack from performance degradation caused by a changeable attack.

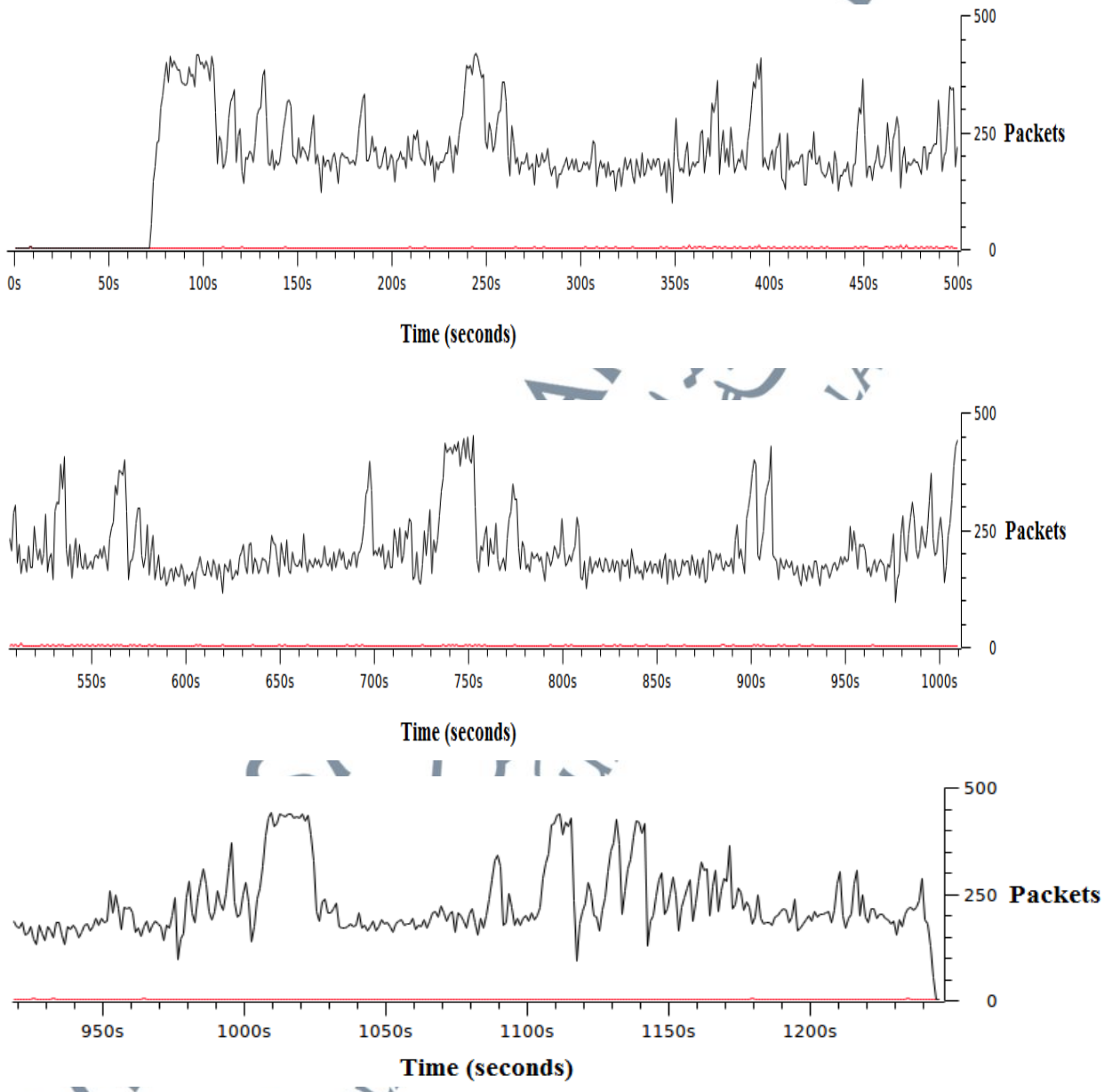


Figure 5.13: The performance under the mixture of traffic (0-500s, 500s-1000s, 1000s-1240)

The lists of source and destination IP addresses that used to send and receive packets generated in the mixture traffic shown in Figures 5.14 and 5.15. The source IP list shows the exact number of attack packets generated in the network as well as the number of legitimate packets.

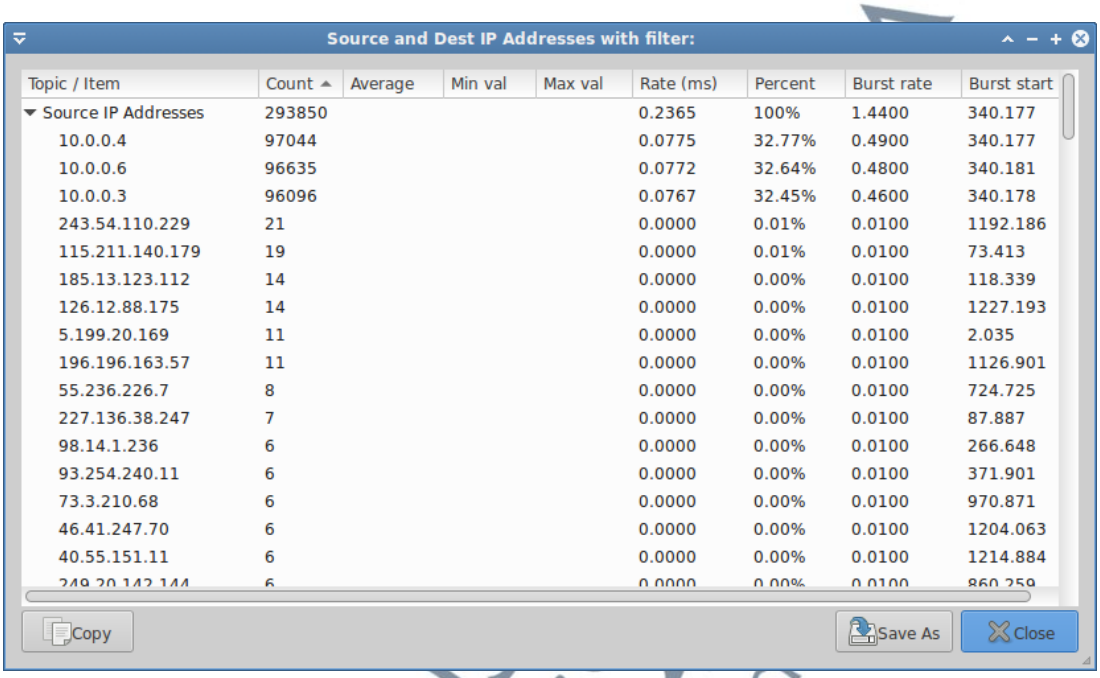


Figure 5.14: The list of source IP addresses

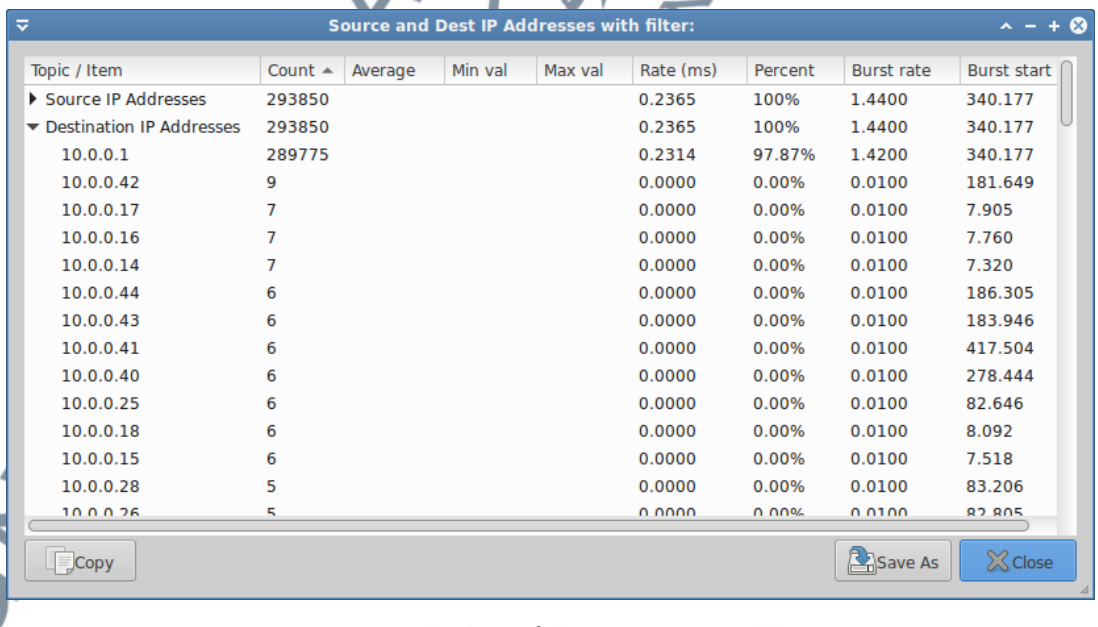
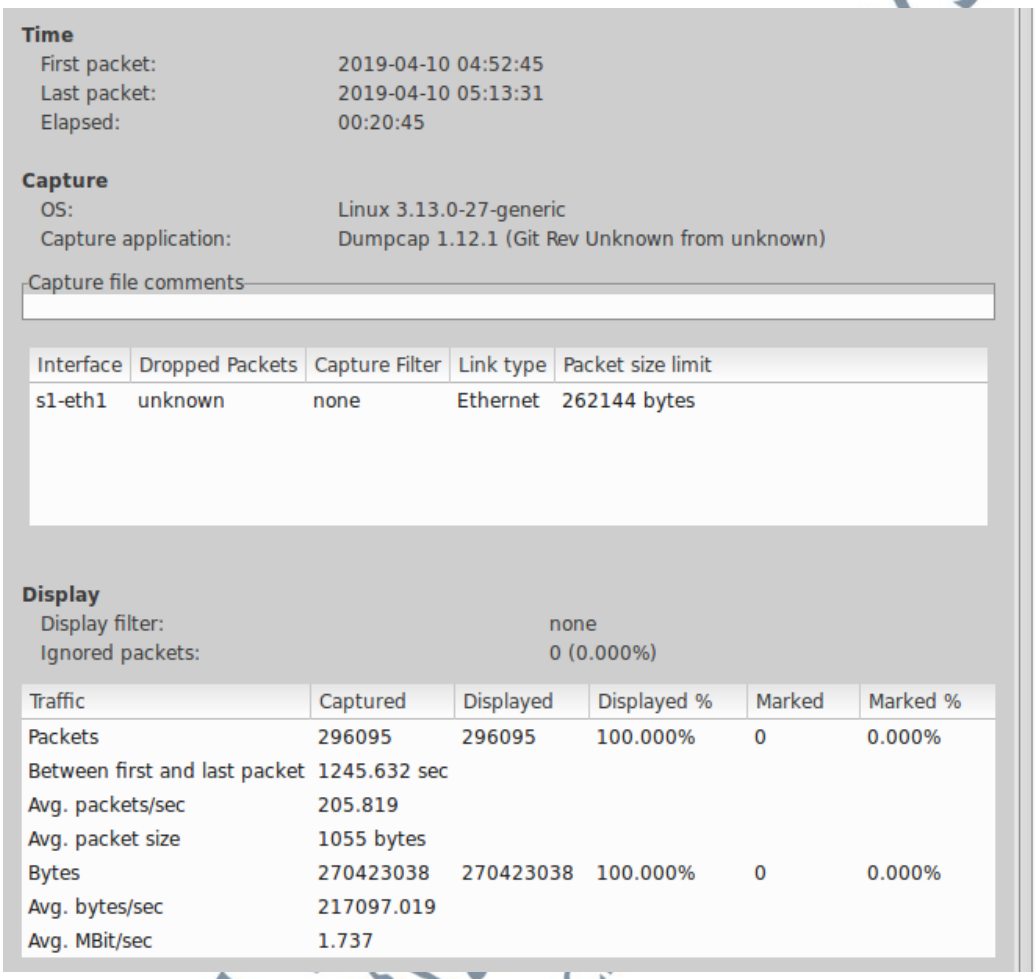


Figure 5.15: The list of destination IP addresses

Figure 5.16 shows that the mixture of traffic generation has last for twenty minutes and forty-five seconds and, the packets have sent at an average of two hundred and five packets per second.



Time

First packet:	2019-04-10 04:52:45
Last packet:	2019-04-10 05:13:31
Elapsed:	00:20:45

Capture

OS:	Linux 3.13.0-27-generic
Capture application:	Dumpcap 1.12.1 (Git Rev Unknown from unknown)

Capture file comments

Interface	Dropped Packets	Capture Filter	Link type	Packet size limit
s1-eth1	unknown	none	Ethernet	262144 bytes

Display

Display filter:	none
Ignored packets:	0 (0.000%)

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	296095	296095	100.000%	0	0.000%
Between first and last packet	1245.632 sec				
Avg. packets/sec	205.819				
Avg. packet size	1055 bytes				
Bytes	270423038	270423038	100.000%	0	0.000%
Avg. bytes/sec	217097.019				
Avg. MBit/sec	1.737				

Figure 5.16: Summary of the mixture traffic generation

5.2.1.4 Scheme Performance Evaluation (UDP flood scenario)

As mentioned in evaluation section in research methodology chapter, to evaluate the performance we test the scheme to obtain the results and then we compare these results with the results of all previous works provided in chapter two.

5.2.1.4.1 Overhead (CPU Usage)

To examine the effect of the proposed scheme on the controller's CPU, we left the simulation running with all the running processes on a Linux PC (running Ubuntu 14.04).

We left the system monitoring application of Linux (htop) running during all the time of the experiment. Figure 5.15 shows the CPU usage while the proposed scheme is in use at the time of traffic generation. The overhead measured is only 34.3%. The overhead measured in the network under the UDP flood scenario is the lowest CPU usage measured in SDN compared to all works presented in table 2.4, chapter two. This shows that the scheme has a positive reflection on the CPU usage of the SDN controller.

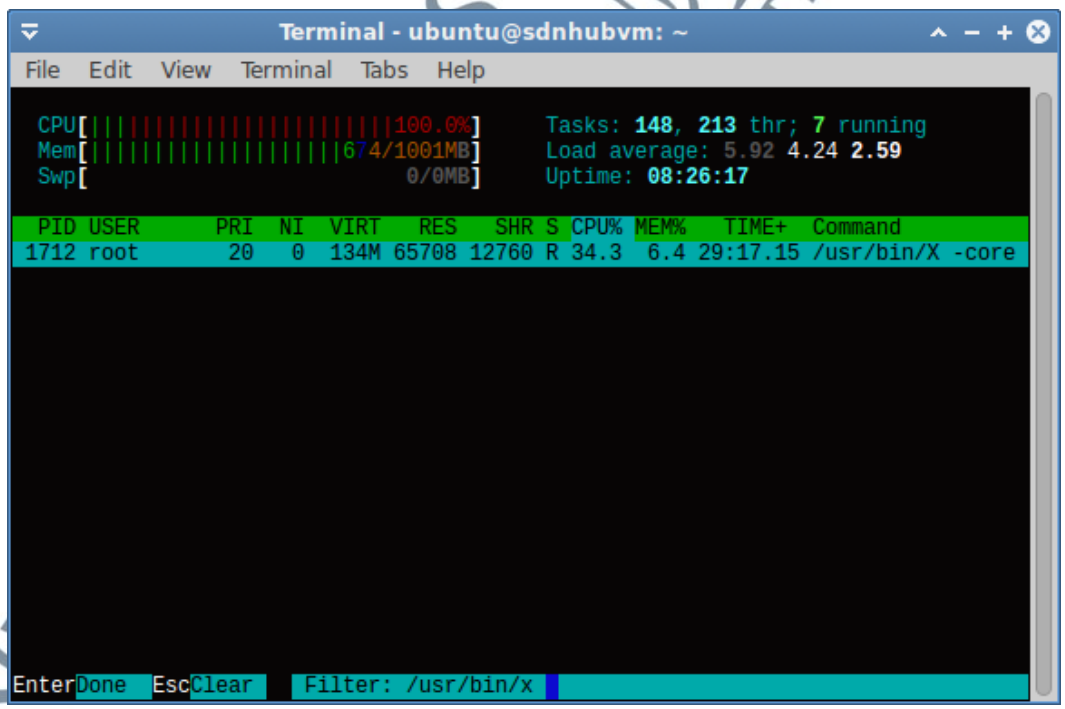


Figure 5.17: CPU usage under UDP flood scenario

5.2.1.4.2 Accuracy

The lowest measured overhead is coming with achieving high accuracy and low false alarms. The accuracy achieved by the proposed scheme under the experimental scenario (UDP flood and Normal traffic) is 99.27%. The accuracy is calculated as in chapter three, section 3.5.3.2 (293850-2125/293850*100 = 99.27%). By considering that the last number of attack packets has been appeared in last terminal was dropped by the scheme to be less than threshold and terminated the flows created by the low number of attack packets, the accuracy will be 100% and no DDoS attack passed unnoticed. This result shows that this scheme can, easily, detect both the attack packets and attack flows when it is destined to a victim in the network.

Since this scheme has been tested under this experimental scenario for many times, no false alarms have been appeared on the POX terminal at any time of the traffic generation. So, the false alarms produced by the scheme under this experimental scenario are 0.

The comparison between the accuracy achieved and the false alarms produced by the proposed scheme and the accuracy and false alarms rates of Machine Learning and Entropy solutions presented in literature review chapter are plotted in the Figure 5.18.

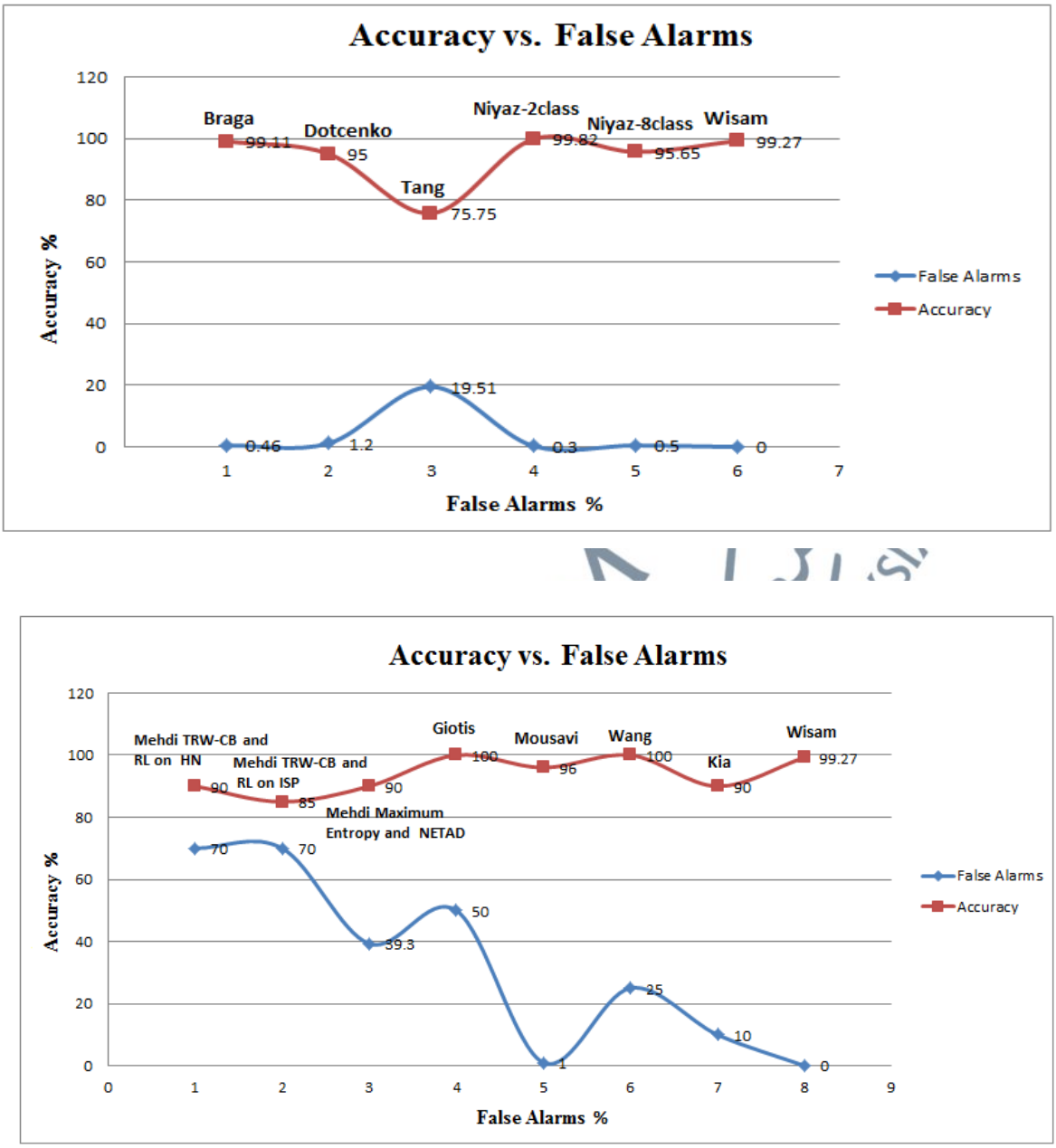


Figure 5.18: An elapsed-time based scheme vs. Solutions based on different techniques (machine learning technique, entropy technique)

Finally, we compare the results of this experimental scenario (overhead, accuracy and false alarms) to the results of all previous works presented in table 2.4 in chapter two. The comparison of overhead and accuracy produced by our scheme to the overhead and accuracy produced by previous works is provided in table 5.1 and table 5.2. The false alarms produced by the scheme are also provided. The results

in table 5.1 are compared to Machine Learning solutions while the results in table 5.2 are compared to Entropy solutions. The results in both tables are meant to show the performance enhancement achieved by our scheme compared to previous works.

Table 5.1: Comparison of results of the elapsed-time based scheme to various machine learning solutions

	Solution developer	**Parameters**	**Accuracy**	**Overhead**	**False Alarms**
Machine Learning based solutions	Braga	Average packets per flow, average bytes per flow, average duration per flow, percentage of pair flows, growth of single flow, and growth of single ports.	99.11%	Very high	0.46%
	Dillon	Number of packets, number of bytes, packet symmetry.	Unknown	Very high	Unknown
	Dotcenko	Connection initiations packets and response packets. Additionally input parameters for the system may contain statistical data from switches are required such as data speed of selected flows and ports and minimum and maximum number of packets per one IP.	95%	Measured on NOX.	1.2%
	Tang	duration, protocol_type, src_bytes, dst_bytes, count and srv_count.	75.75%	Very high	19.51%
	Niyaz	Headers extracted from TCP (Src IP, Window, Dst IP, SYN, Src Port, ACK, Dst Port,	(99.82% for the 2-class model)	Very high	(0.3% for the 2-class model)

		URG, Protocol, FIN, Data Size RST, TTL, PUSH) From UDP (Src IP, Dst IP, Src Port, Dst Port, Protocol, Data Size , TTL) And from ICMP (Src IP, Dst IP, ICMP Type, ICMP Code, Protocol, Data Size, and TTL) and a large number of features extracted from these packets headers.	(95.65% of for the 8-class model)		(0.5% for for the 8-class model)
An elapsed-time based scheme	**Wisam**	Number of packets, number of flows, Dst IP and packet arrival time.	**99.27%**	**34.3%**	**0%**

Table 5.2: Comparison of results of the elapsed-time based scheme to various entropy solutions

	Solution developer	**Parameters**	**Accuracy**	**Overhead**	**False Alarms**
Entropy based solutions	Mehdi	Parameters used in the four algorithms are connection initiations packets, response packets, protocol type, destination port number, all non-IP packets, all incoming traffic, TCP packets starting after the first 100 bytes and packets to any address/port/protocol combination if more than 16 are received in a minute.	(TRW-CB and Rate Limiting algorithms on the home network achieved 90% and on the ISP 85%) (Maximum Entropy and NETAD algorithms achieved 90%)	Measured on NOX	70%

	Giotis	Src IP, Dst IP, Src Port and Dst Port	100%	Measured on NOX	50%
	Mousavi	Number of packets and Dst IP	96%	78%	1%
	Wang	Dst IP, number of packets that have same Dst IP and number of times Dst IP is repeated.	100%	high	25%
	Kia	Number of packets, Dst IP, total number of Dst IP and number of times Dst IP is repeated.	90%	high	10%
An elapsed-time based scheme	**Wisam**	Number of packets, number of flows, Dst IP and packet arrival time.	**99.27%**	**34.3%**	**0%**

5.2.2 Experiment Two (Low-Rate SYN Scenario)

A SYN flood (half-open attack) is a type of DDoS attack which aims to make the target machine unavailable to legitimate traffic by consuming all available target's resources. By repeatedly sending initial connection request (SYN) packets, the attacker is able to overwhelm all available ports on a targeted machine, causing the targeted device to respond to legitimate traffic sluggishly or not at all.

In contrast to SYN flooding DDoS attacks, the low-rate DDoS attacker exploits the vulnerability of TCP's congestion control mechanism in two ways. First, periodically sending burst attack packets over short periods of time repeatedly. Second, continuously launching attack packets at a constant low-rate. To avoid being detected by existing detection solutions, these attacks reduce the average number of attack packets at a time of traffic generation (Zhou et al., 2017).

5.2.2.1 Test Case One (Normal Traffic Generation)

In this test case, the only traffic generated in the network is the normal TCP traffic. Figure 5.19 shows no indications of attack occurrence at the time of generation. A Python script is used to generate normal TCP traffic manually. Wireshark packets analyzer is used to capture the generated packets.

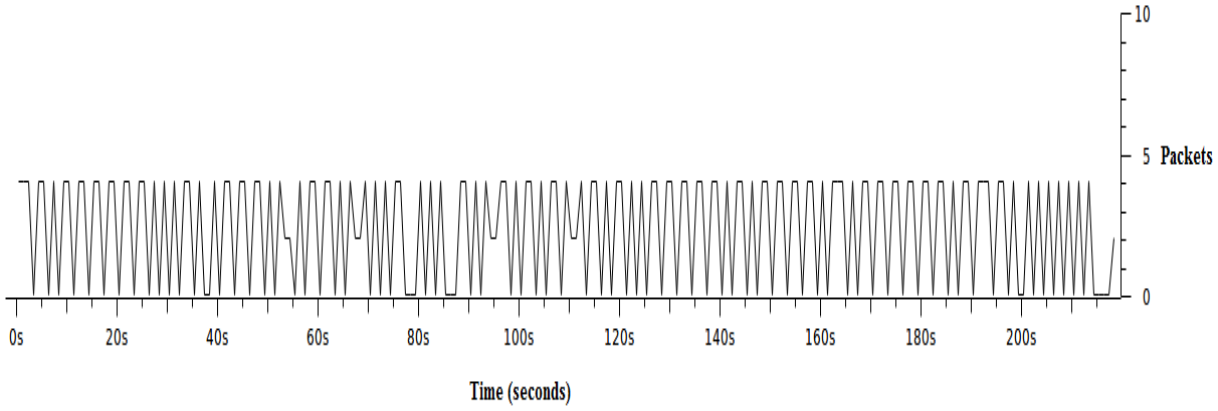
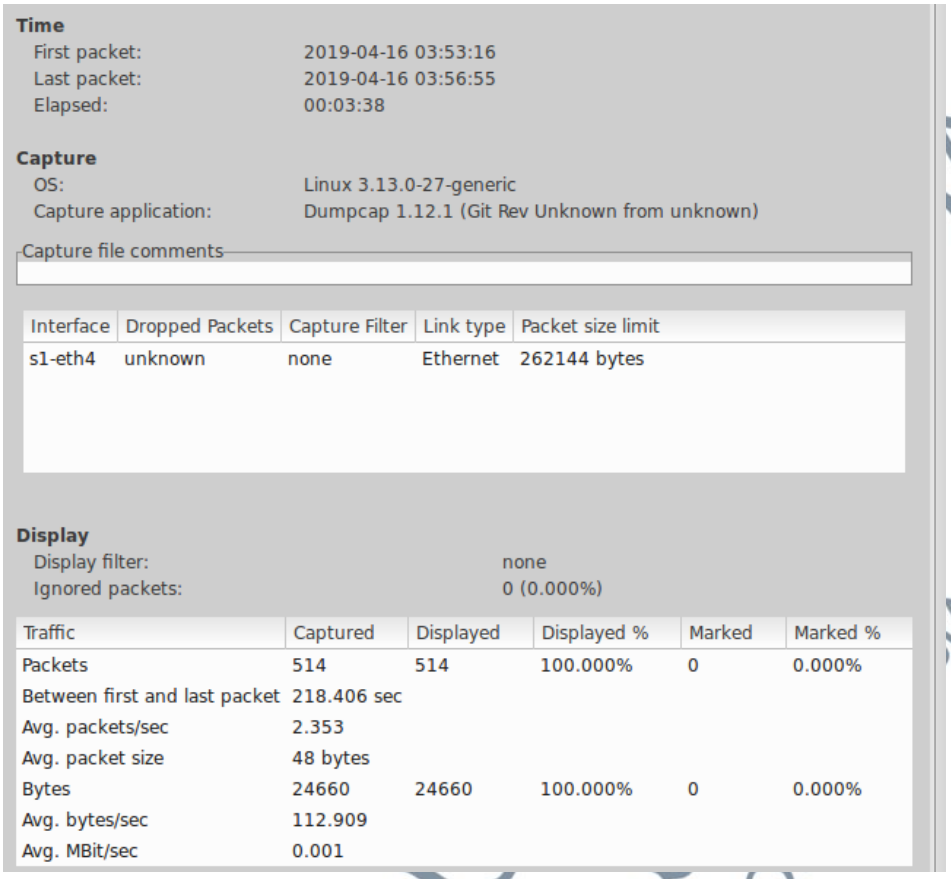


Figure 5.19: The performance under normal TCP traffic

As shown in Figure 5.20, the traffic has generated in three minutes and thirty-eight second and the average of packets sent in the network was less than three packets per second. The traffic generated is meant to establish normal connections. Meanwhile, we can observe the reaction of the scheme on these packets from Figure 5.21.



Time

First packet:	2019-04-16 03:53:16
Last packet:	2019-04-16 03:56:55
Elapsed:	00:03:38

Capture

OS:	Linux 3.13.0-27-generic
Capture application:	Dumpcap 1.12.1 (Git Rev Unknown from unknown)

Capture file comments

Interface	Dropped Packets	Capture Filter	Link type	Packet size limit
s1-eth4	unknown	none	Ethernet	262144 bytes

Display

Display filter:	none
Ignored packets:	0 (0.000%)

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	514	514	100.000%	0	0.000%
Between first and last packet	218.406 sec				
Avg. packets/sec	2.353				
Avg. packet size	48 bytes				
Bytes	24660	24660	100.000%	0	0.000%
Avg. bytes/sec	112.909				
Avg. MBit/sec	0.001				

Figure 5.20: Summary of normal TCP traffic

Figure 5.21 shows that no actions taken against the generated packets because the scheme realized that these packets were normal packets used to establish normal TCP handshake.

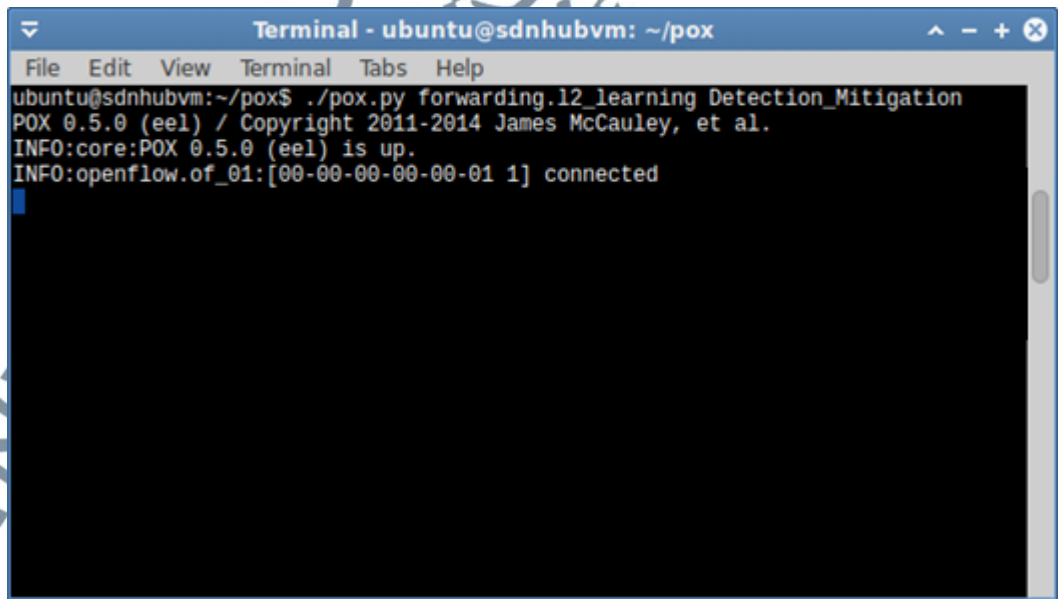


Figure 5.21: Results under normal TCP traffic

We can observe from the list appeared in Figure 5.22 that the traffic generated between hosts is normal TCP traffic. Figure 5.22 shows only two IP addresses with the number of packets that have sent between them.

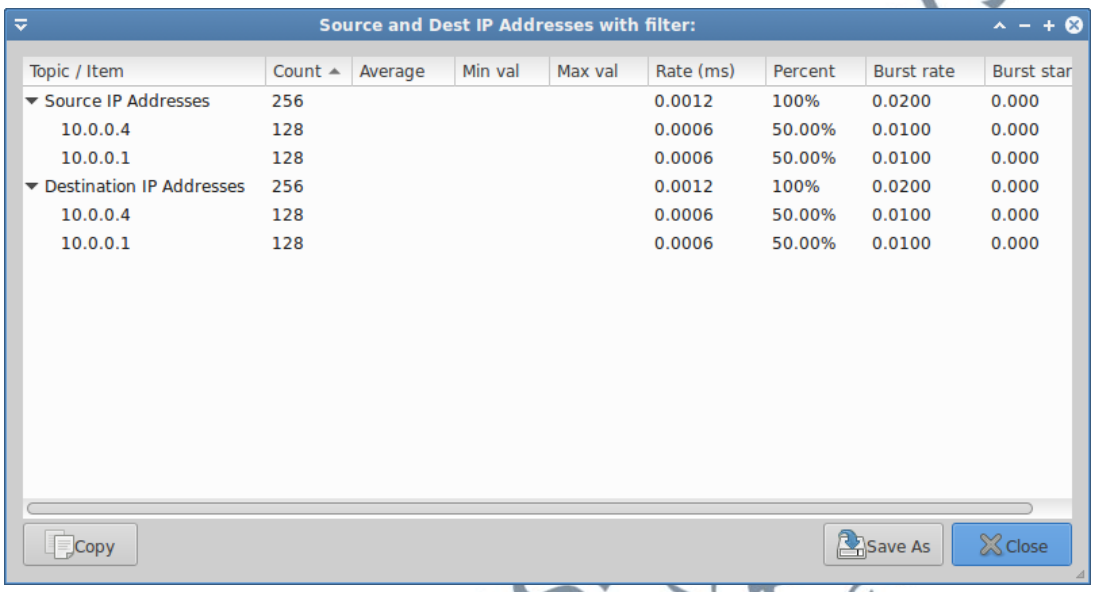


Figure 5.22: List of source and destination IP addresses

5.2.2.2 Test Case Two (Attack Traffic Generation)

In the contrast of test case two in experiment one, we will generate the low-rate attack traffic in two terms, a low rate of packets in the short term and another in the long term. SYN attack in this test case is meant to deny normal flows by overflowing the OpenFlow switch with lots of incomplete synchronization requests. This test case shows the importance of detecting the suspicious packets that have sent in a constant low-rate. Python scripts were run on the hosts manually to generate the low rate SYN attack.

5.2.2.2.1 Low-rate packets in the short term

To avoid being detected, a normal DDoS attack usually would not last for a very long time. For example, ten minutes is the normal duration of attack (Yu et al.,

2012). As a low-rate attack is continuously launching attack packets at a constant low-rate, therefore, it is essential to quickly detect short duration attacks (Zhou et al., 2017).

Three DDoS scripts run to generate low-rate attack in the short term. DDoS scripts run on h1, h2, and h3 to attack h5. Once the DDoS scripts begin generating the attack traffic, the scheme directly shows the detected attack flows as appeared in Figure 5.23.1. These flows will be deleted from the switch once they exceed the threshold (ten flows).

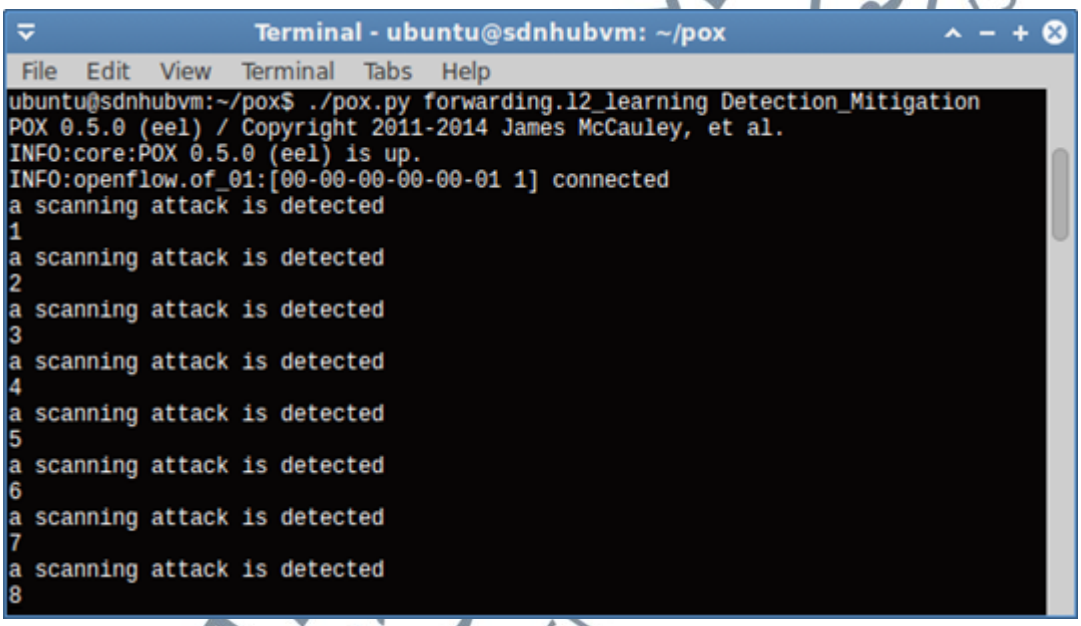


Figure 5.23.1: Detecting low-rate attack packets from the first second

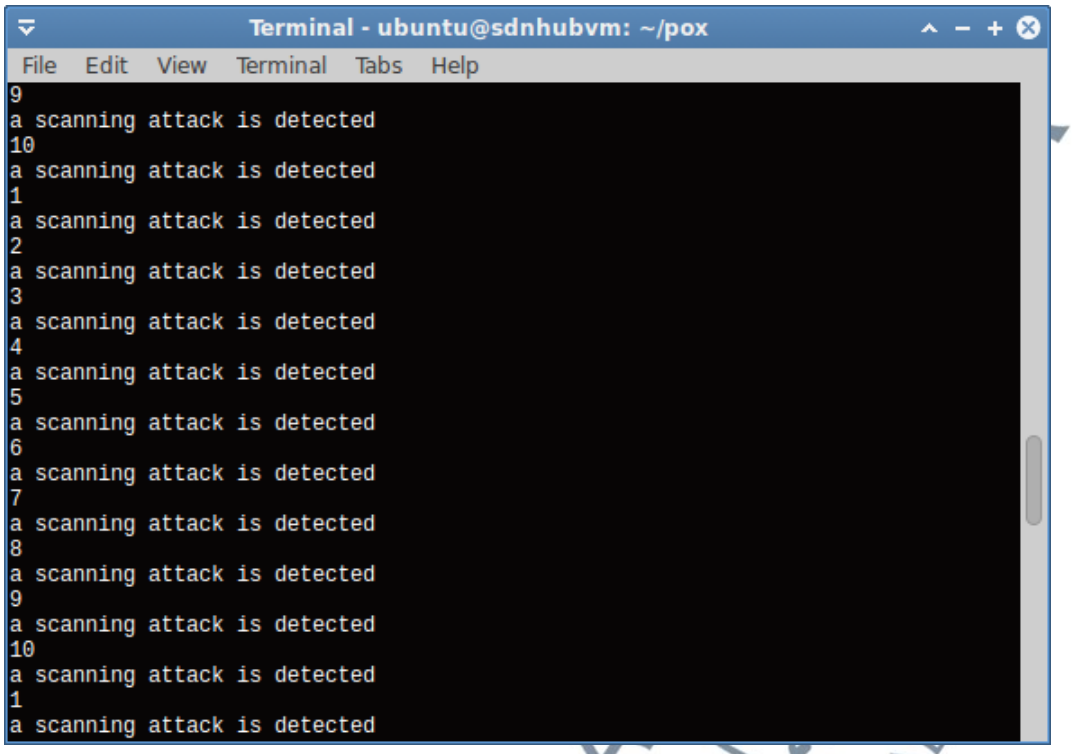


Figure 5.23.2: Results after three minutes

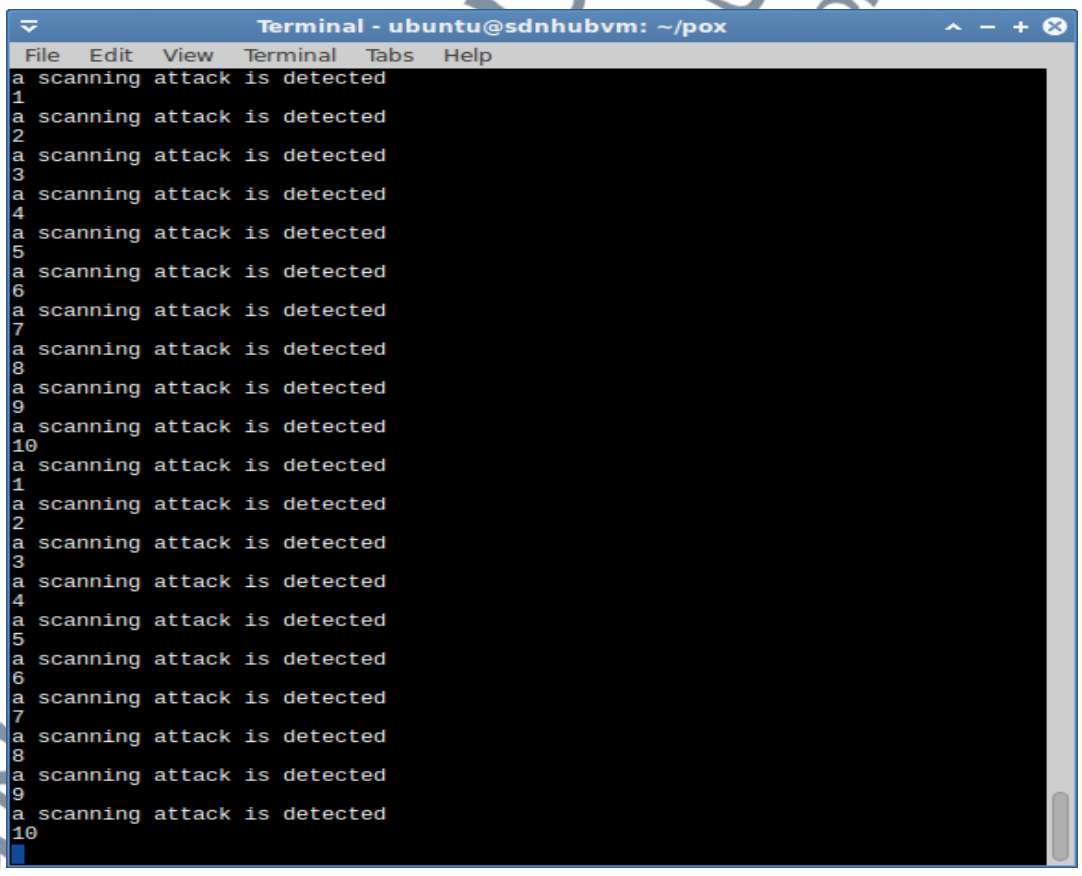


Figure 5.23.3: Results after five minutes

Figures 5.23.1, 5.23.2 and 5.23.3 show that the attack flows created by the low rate attack traffic has detected and removed. Furthermore, Figure 5.23.3 shows that the scheme has stopped taking actions once the attack hosts stop generating traffic. On a hand, the scheme proved that OpenFlow switch has protected from being overflowed by creating new flows, on the other hand, the scheme protected the controller which is failure point of SDN from bottlenecks and resources depleting due to increasing in packet-in messages caused by the new flows.

The graph in Figure 5.23.4 shows the performance under low-rate attack in the short term. The graph also shows that there is no degradation in the performance of the SDN components during the short term of low-rate attack launching. Figure 5.23.5 shows details about the low-rate attack packets in the short term.

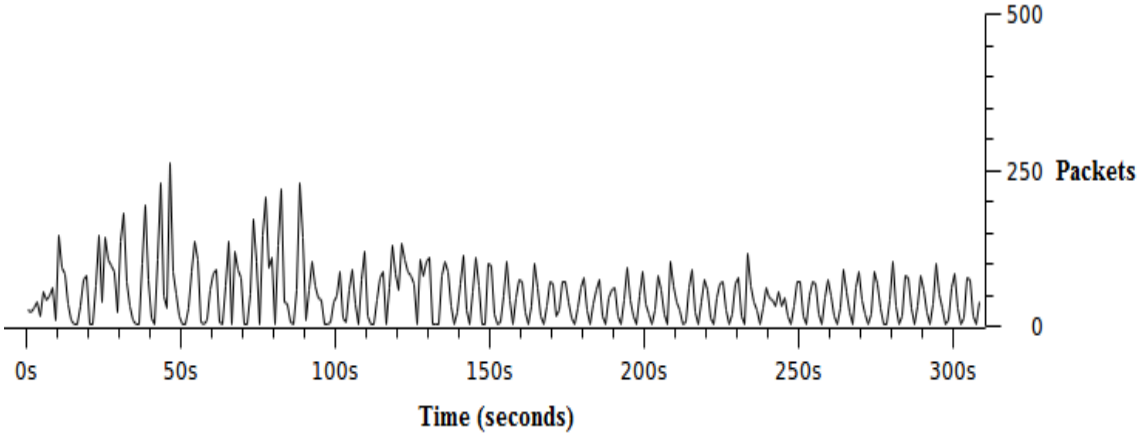
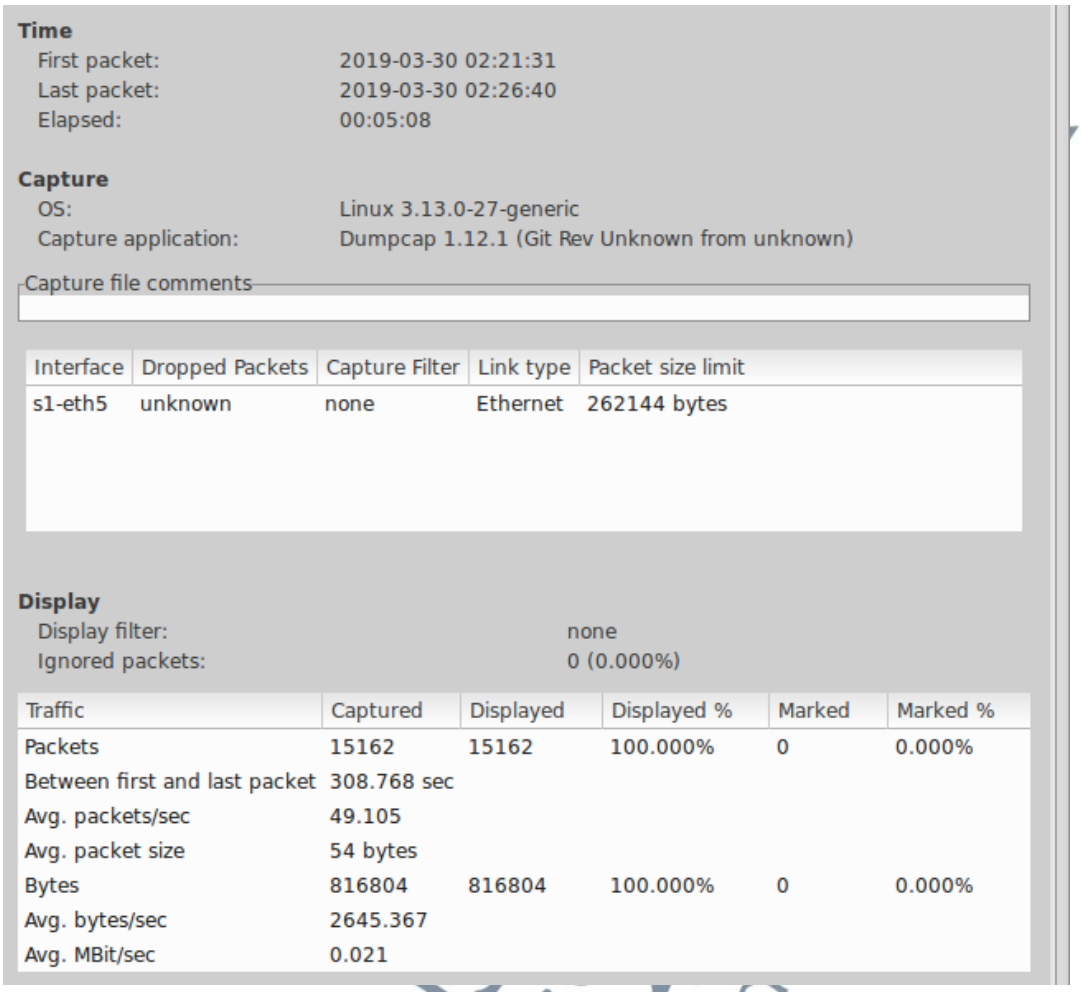


Figure 5.23.4: Performance under the low-rate attack in the short term



Time

First packet:	2019-03-30 02:21:31
Last packet:	2019-03-30 02:26:40
Elapsed:	00:05:08

Capture

OS:	Linux 3.13.0-27-generic
Capture application:	Dumpcap 1.12.1 (Git Rev Unknown from unknown)

Capture file comments

Interface	Dropped Packets	Capture Filter	Link type	Packet size limit
s1-eth5	unknown	none	Ethernet	262144 bytes

Display

Display filter:	none
Ignored packets:	0 (0.000%)

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	15162	15162	100.000%	0	0.000%
Between first and last packet	308.768 sec				
Avg. packets/sec	49.105				
Avg. packet size	54 bytes				
Bytes	816804	816804	100.000%	0	0.000%
Avg. bytes/sec	2645.367				
Avg. MBit/sec	0.021				

Figure 5.23.5: Summary of low-rate attack in the short term

The lists of source and destination IP addresses used to send and receive packets were provided in Figures 5.23.6 and 5.23.7, respectively. The total number of packets generated in the short term was fifteen thousand packets in five minutes and eight seconds. As appeared in Figure 5.24.6, single attack packets sent from different forged source IP addresses to increase the number of flows instead of packets. Consequently, the low-rate attack succeeded to maximize the number of flows to fifteen thousand flows as a separate flow for each packet.

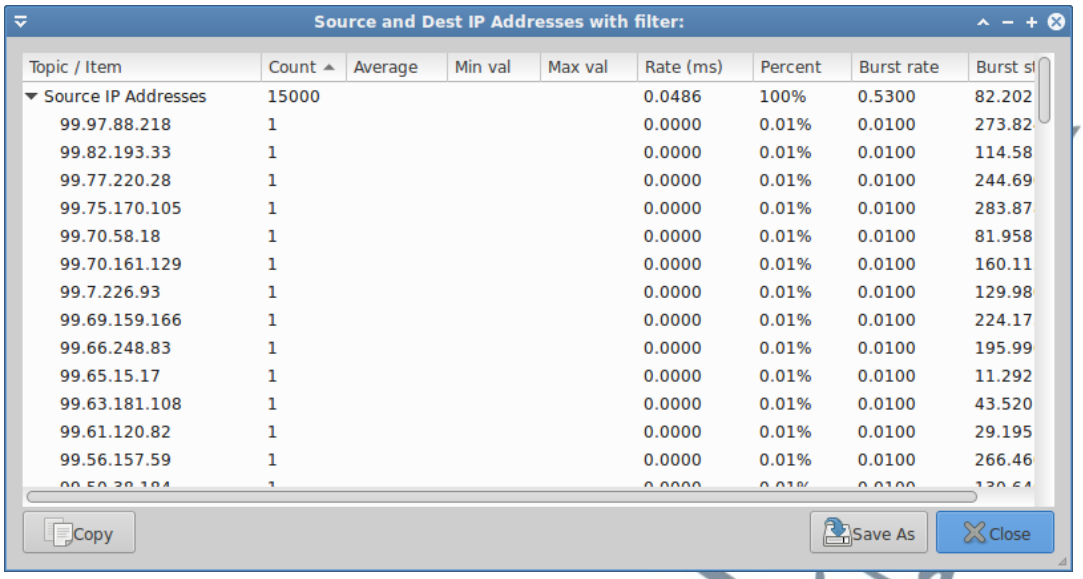


Figure 5.23.6: List of the source IP addresses

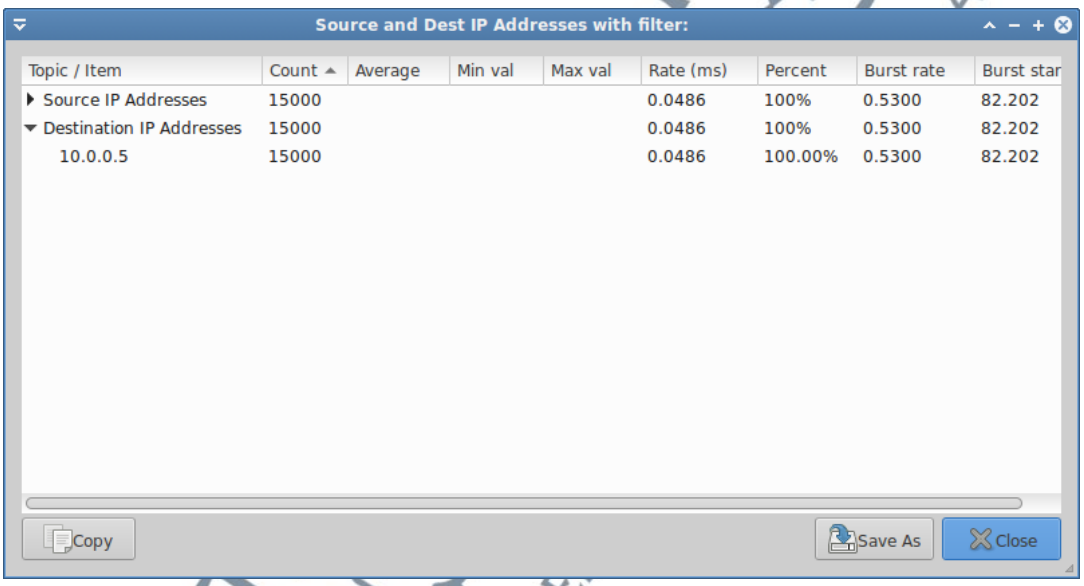


Figure 5.23.7: List of the destination IP addresses

5.2.2.2.2 Low rate packets in the long term

Because of the key feature of the attack: send the attack packets at a low average rate, a low-rate DDoS attacker may keep sending attack packets for a long time. For this, it is essential to test a low-rate DDoS detection solution not only in the short term but also in the long term (Zhou et al., 2017).

DDoS attack scripts run to launch low-rate attack packets towards host h6 which is the host plays the role of victim. Once the low-rate traffic begins generating towards the target machine, the scheme raises a warning of an attack occurs when it notices that the number of flows is getting increased while the number of packets does not exceed the threshold. The scheme directly reacts to the low-rate attack packets by terminating the flows that contain these packets as appeared in Figures from 5.24.1 to 5.24.5.

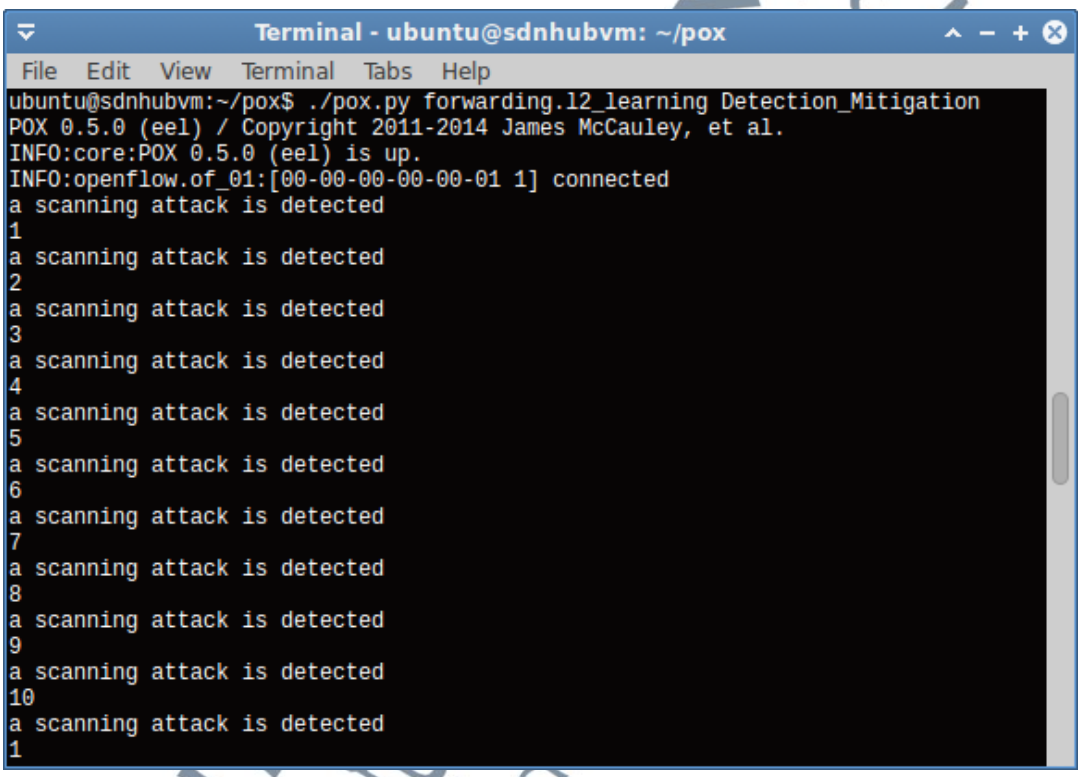


Figure 5.24.1: Detecting low-rate attack packets from the first second

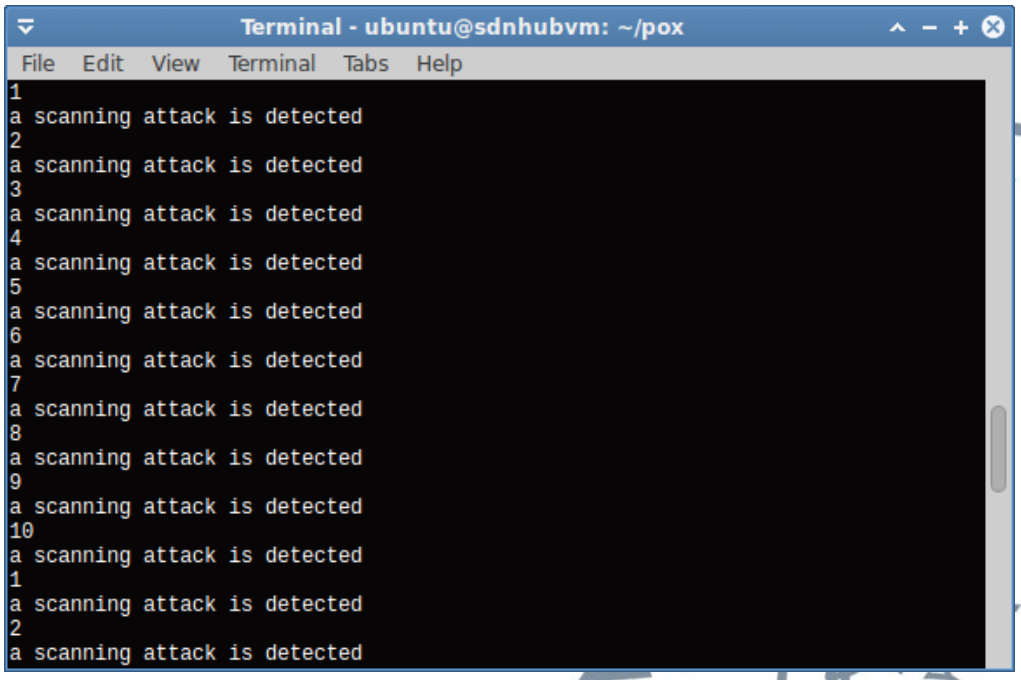


Figure 5.24.2: Results after five minutes

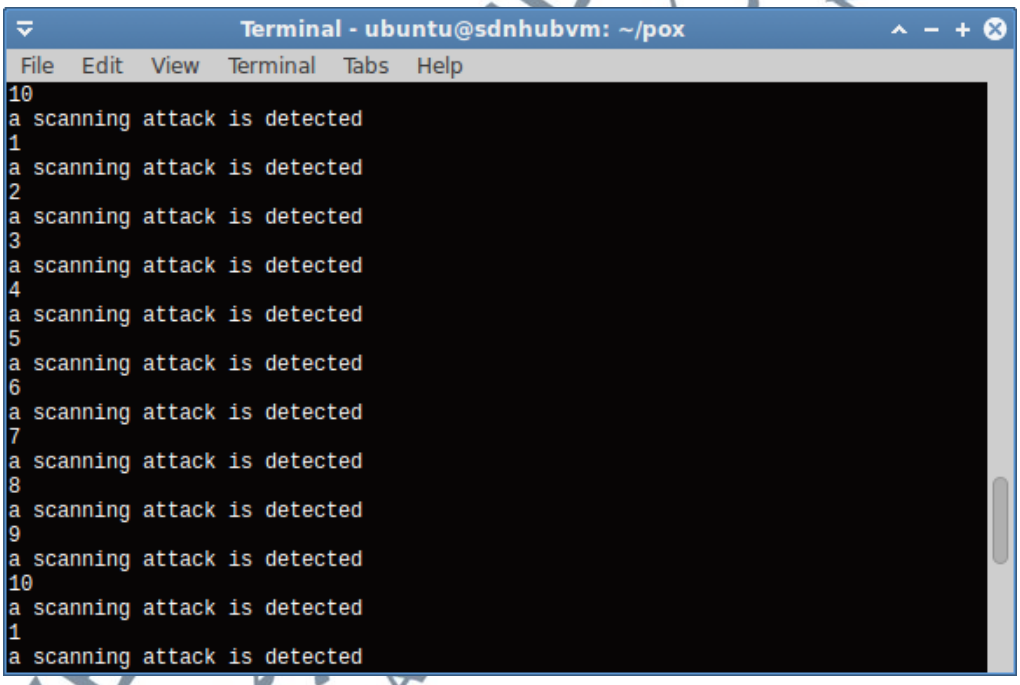


Figure 5.24.3: Results after fifteen minutes

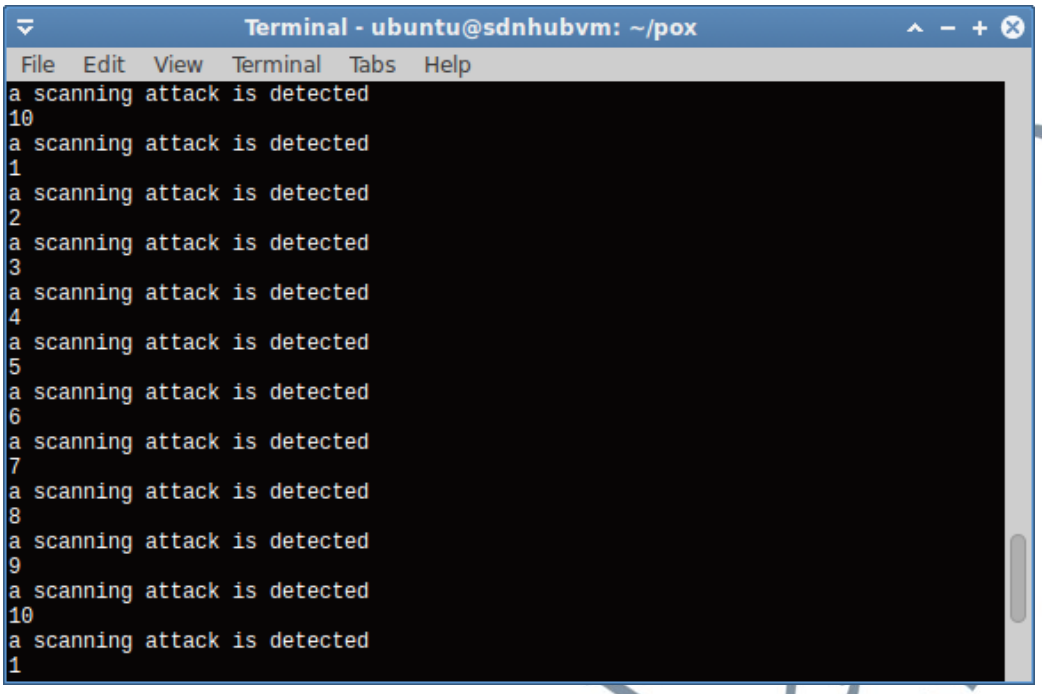


Figure 5.24.4: Results after twenty five minutes

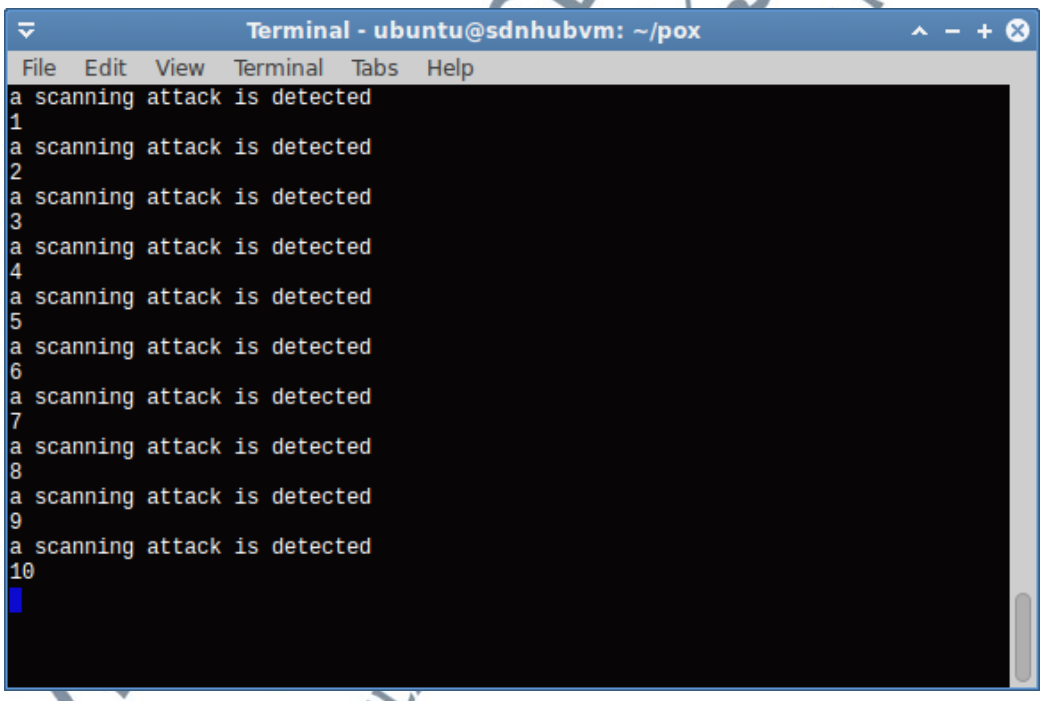


Figure 5.24.5: The last results

Noticing the increase in the number of the attack flows instead of packets while the number of packets does not exceed the threshold shows the importance of detecting the low-rate attacks at the time of no signs of a flooding attack appeared.

We increased the number of low-rate attack packets that accordingly increase the packet-in messages to examine the proposed scheme in the case of low-rate attack tries to congest the controller in the long term. The spoofed source IP addresses that send packets separately to increase the number of single flows appeared in Figure 5.24.6. The list of destination IP addresses is shown in Figure 5.24.7.

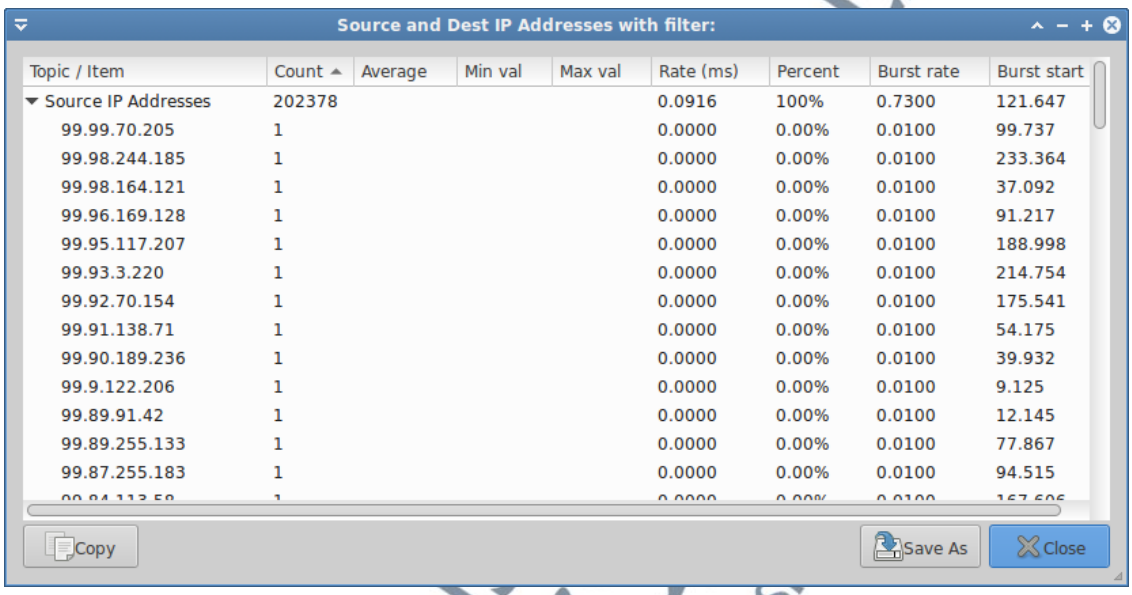


Figure 5.24.6: List of the source IP addresses

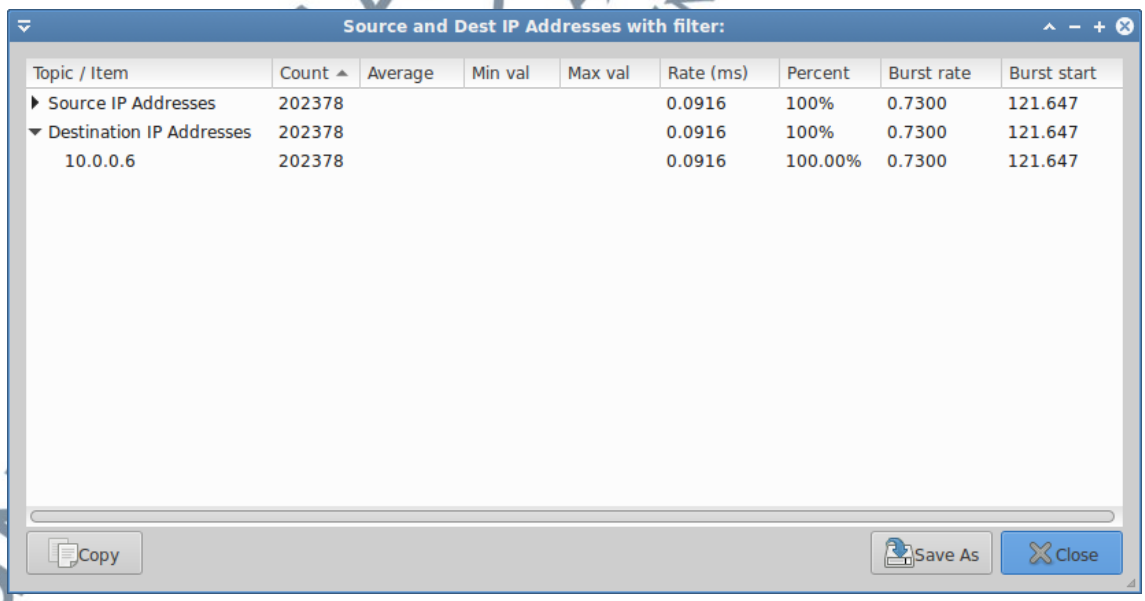
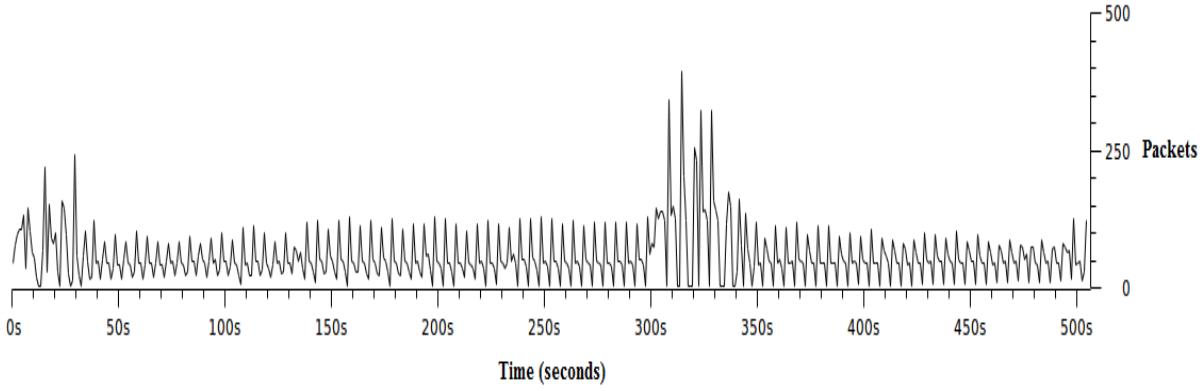
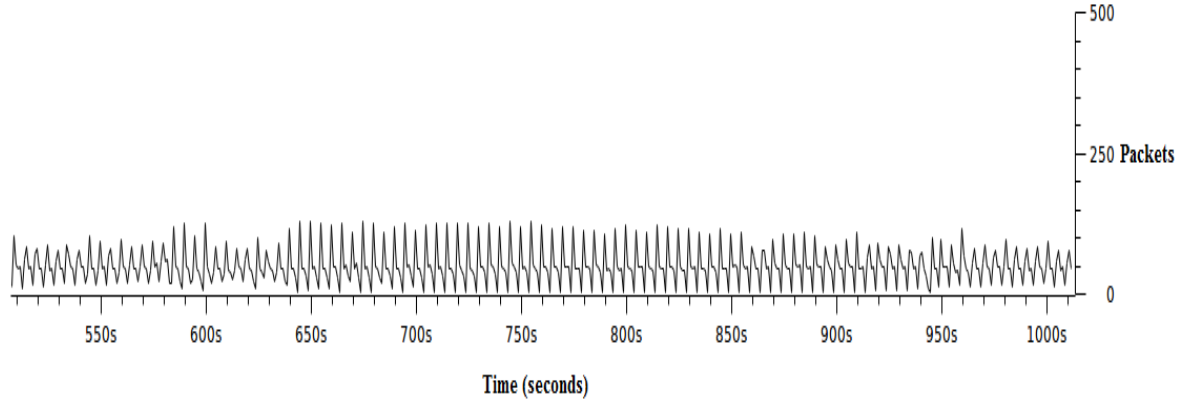
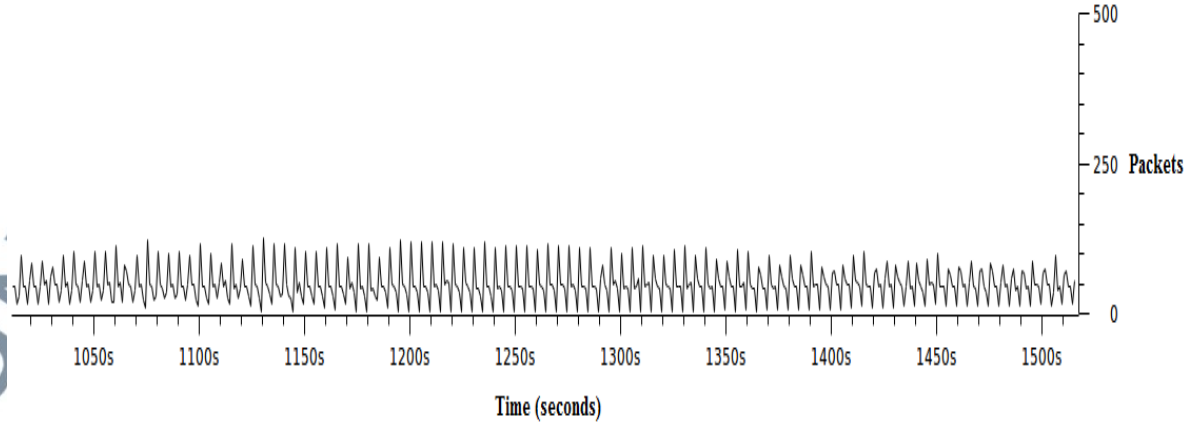


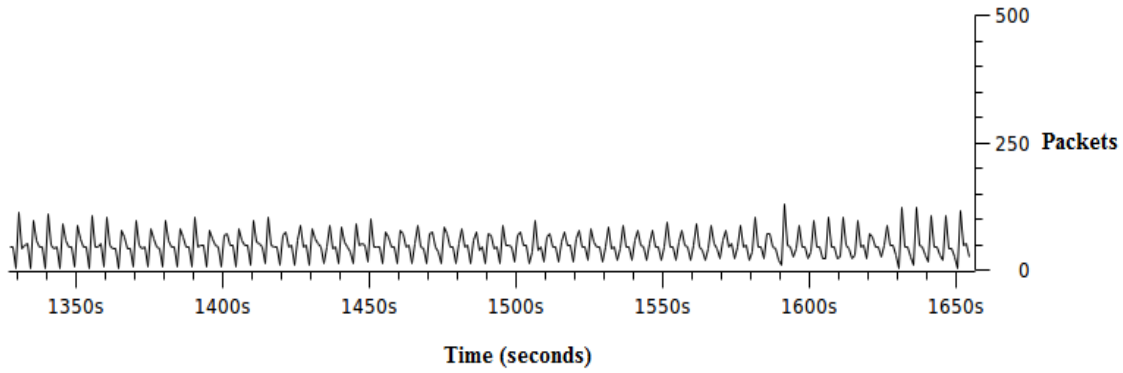
Figure 5.24.7: List of the destination IP addresses

The graphs in Figure 5.24.8 illustrate the performance of the SDN components under an increased low-rate attack in the long term. The X-axis represents the number of packets while the Y-axis represents the time in seconds. The graphs show that the scheme has protected the controller and the switch from performance degradation along the period of time of the attack generation.









Figures 5.24.8: Performance under the low-rate attack in the long term (0-500s, 500s-1010s, 1010s-1510, 1510s-1650s)

Figure 5.24.9 shows a summary of the increased low-rate attack in the long term. We can see that the time for attack generation has increased and the average of packets has maximized.

Time

First packet:	2019-04-02 20:31:40
Last packet:	2019-04-02 20:59:35
Elapsed:	00:27:55

Capture

OS:	Linux 3.13.0-27-generic
Capture application:	Dumpcap 1.12.1 (Git Rev Unknown from unknown)

Capture file comments

Interface	Dropped Packets	Capture Filter	Link type	Packet size limit
s1-eth6	unknown	none	Ethernet	262144 bytes

Display

Display filter:	none
Ignored packets:	0 (0.000%)

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	202378	202378	100.000%	0	0.000%
Between first and last packet	1675.293 sec				
Avg. packets/sec	120.802				
Avg. packet size	54 bytes				
Bytes	10899876	10899876	100.000%	0	0.000%
Avg. bytes/sec	6506.252				
Avg. MBit/sec	0.052				

Figure 5.24.9: Summary of low-rate attack in the long term

5.2.2.3 Test Case Three (Mixture of low-rate Attack and Normal Traffic in long term)

In this test case, low-rate attack and normal traffic mixed in the same traffic. This mixture of traffic generated in the long term. Hosts h2 and h5 designated to launch the low-rate attack and host h1 to generate normal traffic. Host h4 is the victim.

On the POX controller terminal, no actions have displayed by the scheme although scripts that generate the normal traffic are currently running. Figure 5.25 shows the terminal for the first twenty seconds of normal traffic generation.

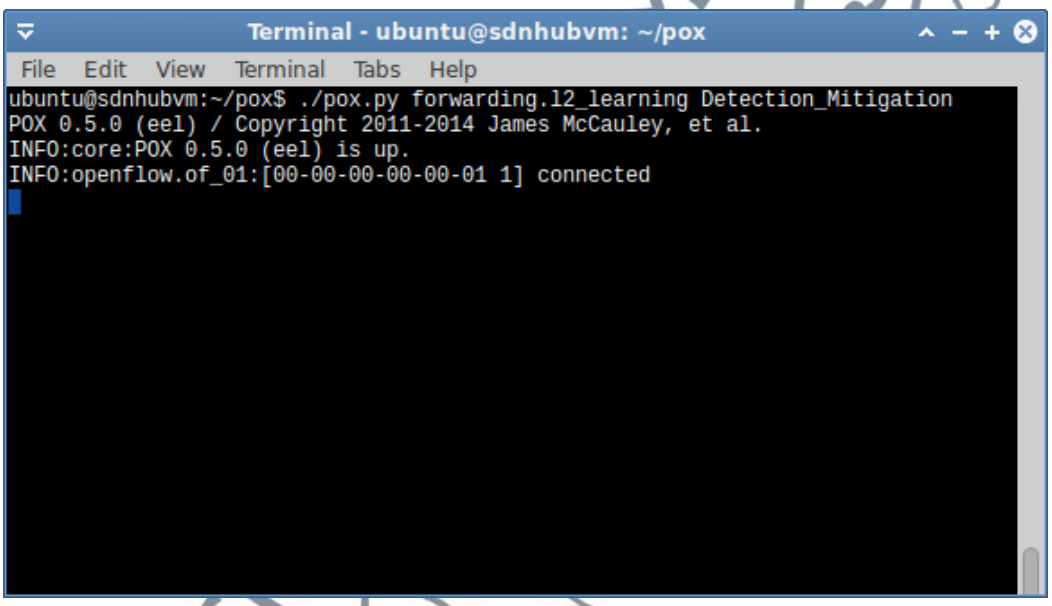


Figure 5.25: POX controller terminal within the first twenty seconds

After twenty seconds of generating normal traffic, low-rate attack launched from attack hosts along with normal traffic that is still generating. Figures from 5.26 to 5.30 show the detection and mitigation processes of the low-rate attack packets at different time slots after the attack traffic began generating.

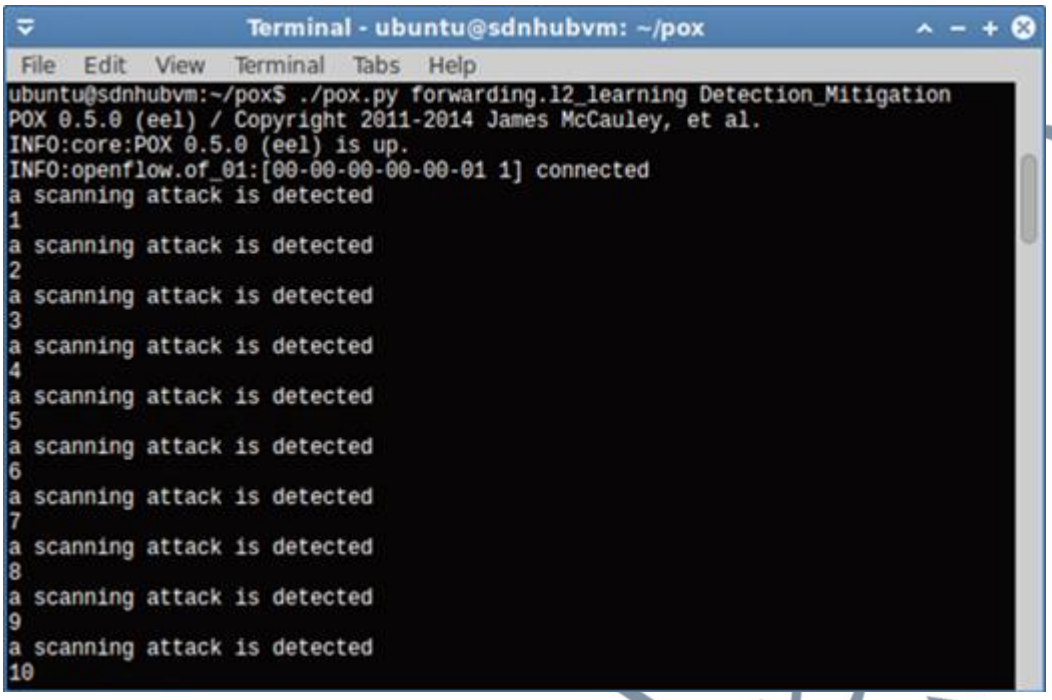


Figure 5.26: Results after twenty seconds

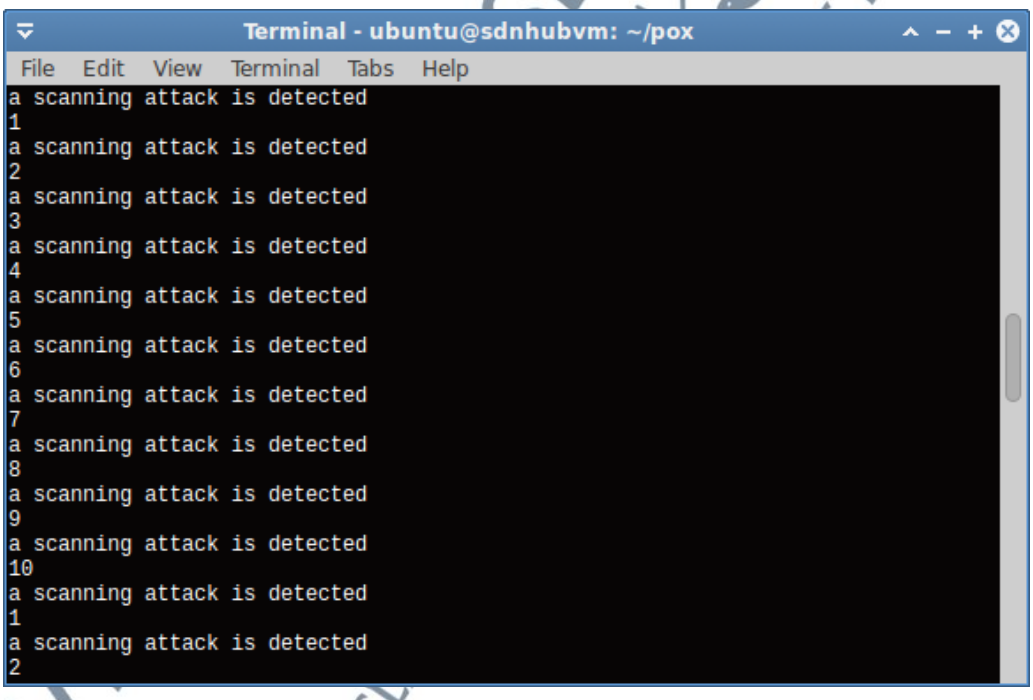


Figure 5.27: Results after ten minutes

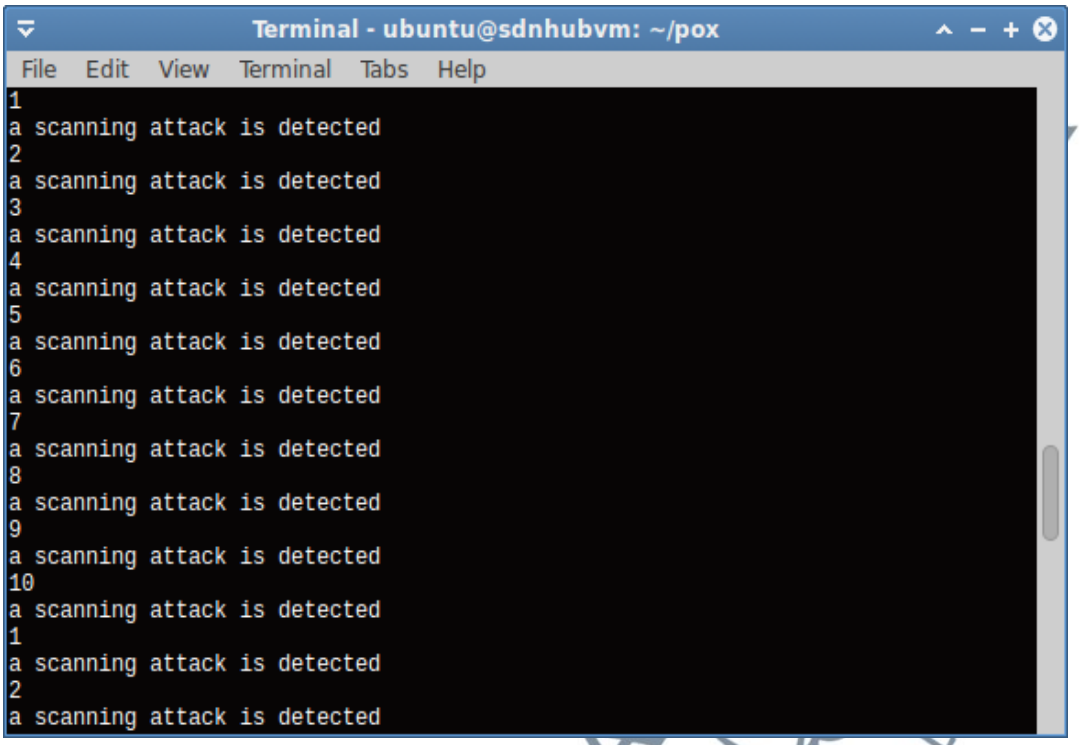


Figure 5.28: Results after fifteen minutes

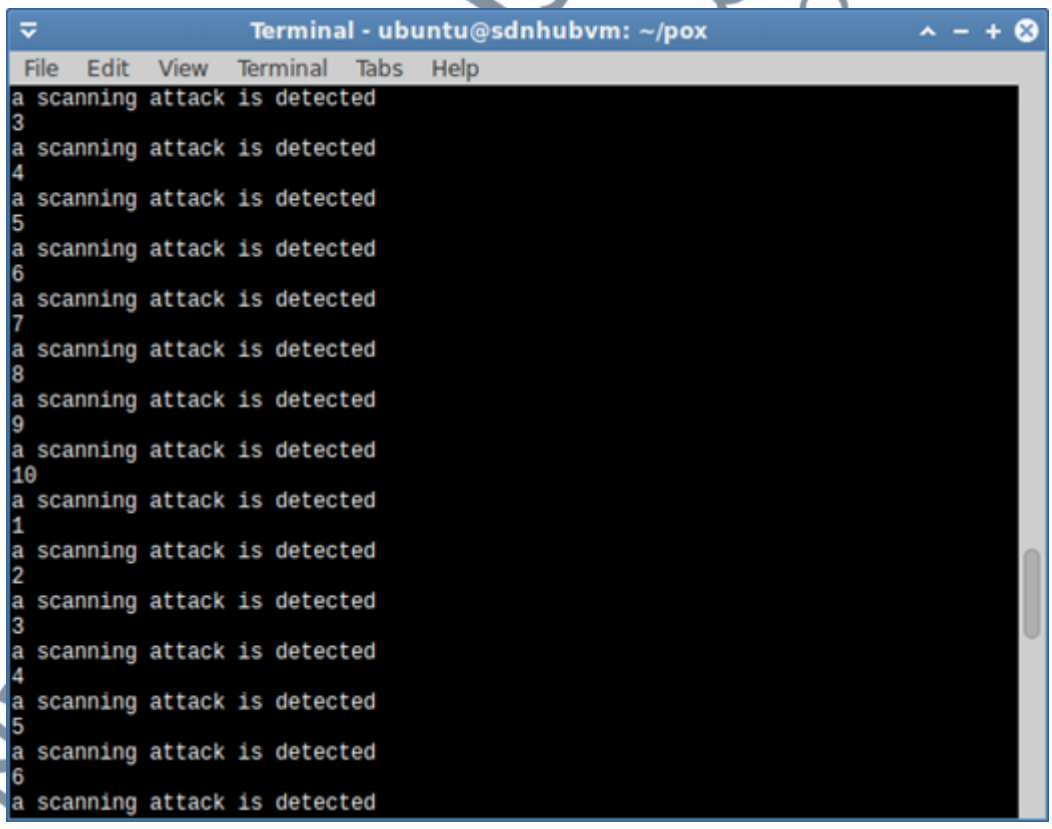


Figure 5.29: Results after twenty minutes

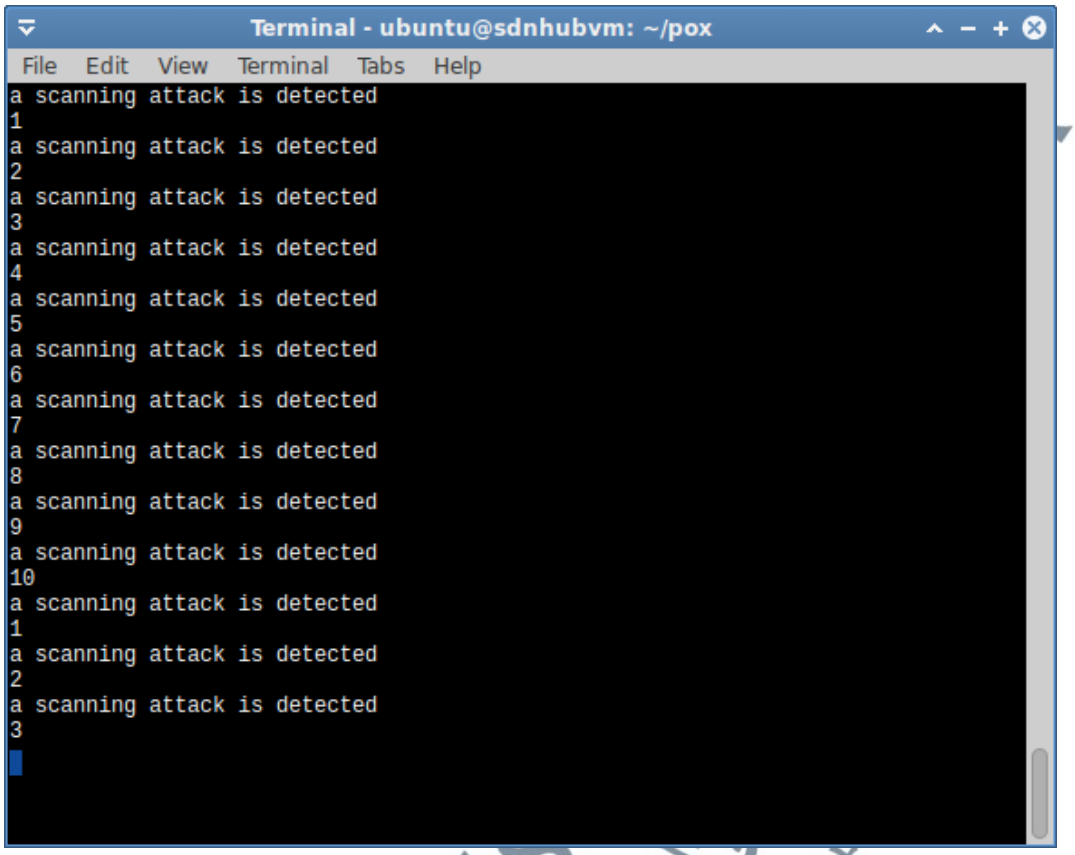


Figure 5.30: The last results

As appeared in Figure 5.30, the scheme stops detecting attack packets as long as the attack packets stop generated. Although the normal traffic is still generating, the scheme has stopped detecting packets. This gives proof that the scheme efficiently prevents producing false positives and false negatives.

List of source IP addresses appeared in Figure 5.31 shows that spoofed source IP addresses have sent the low-rate attack packets separately in the long term to create a vast number of flows. This is noticed from the different way used by the legitimate source IP address and spoofed IP addresses to send packets. As clearly shown in the list, the number of packets has sent by the legitimate host (10.0.0.1) was only two hundred and sixty-four packets, sent as a group of packets while rest of packets have sent separately.

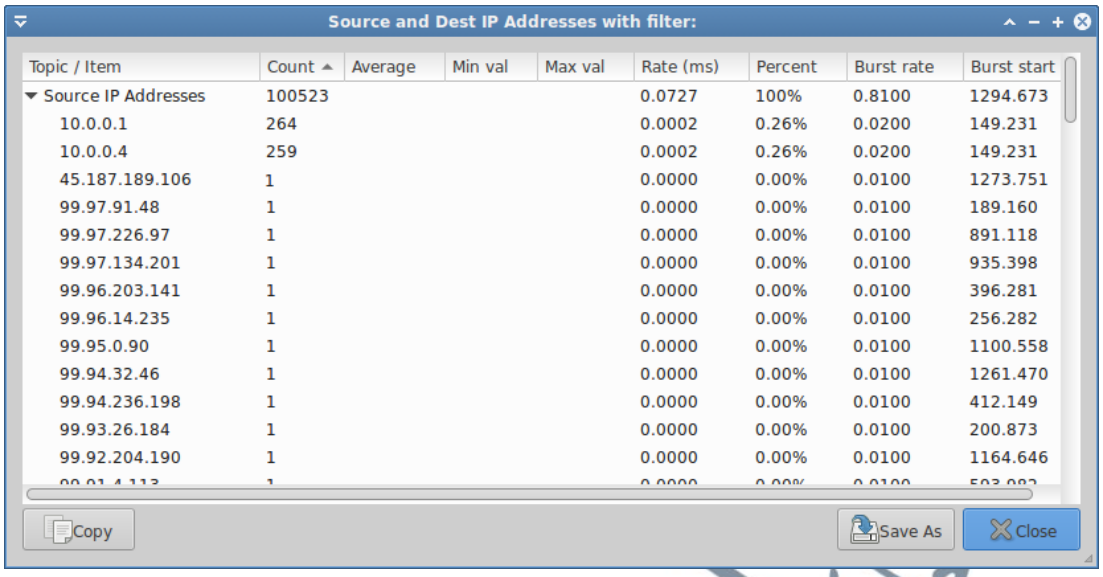


Figure 5.31: The list of the source IP addresses

Figure 5.32 shows the list of destination IP addresses. To avoid misunderstanding about appearing two destinations IP addresses in the list, the two IP addresses are the IP addresses of hosts used to establish the normal TCP handshake.

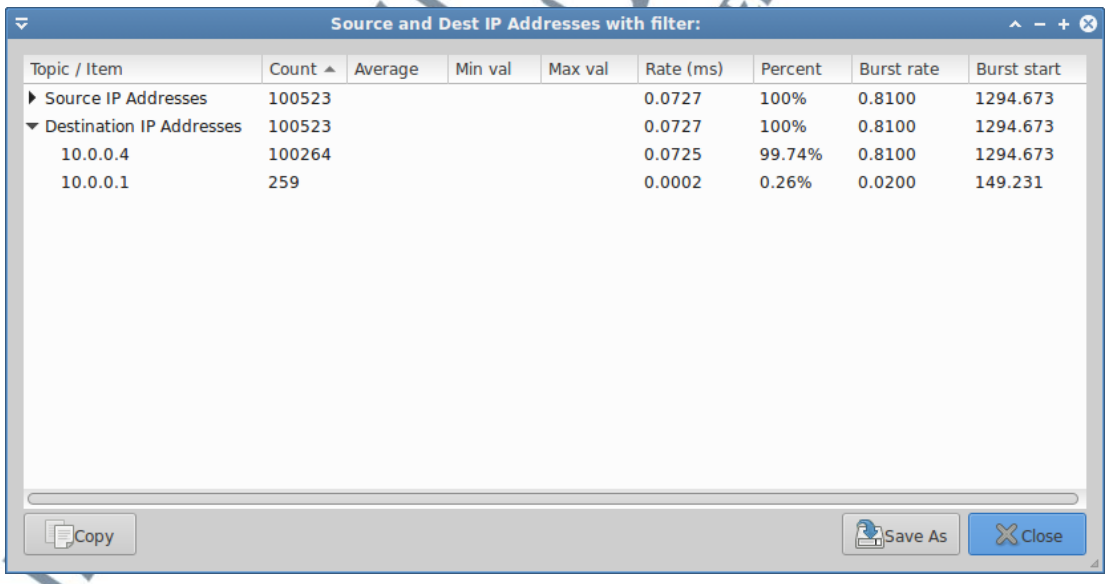
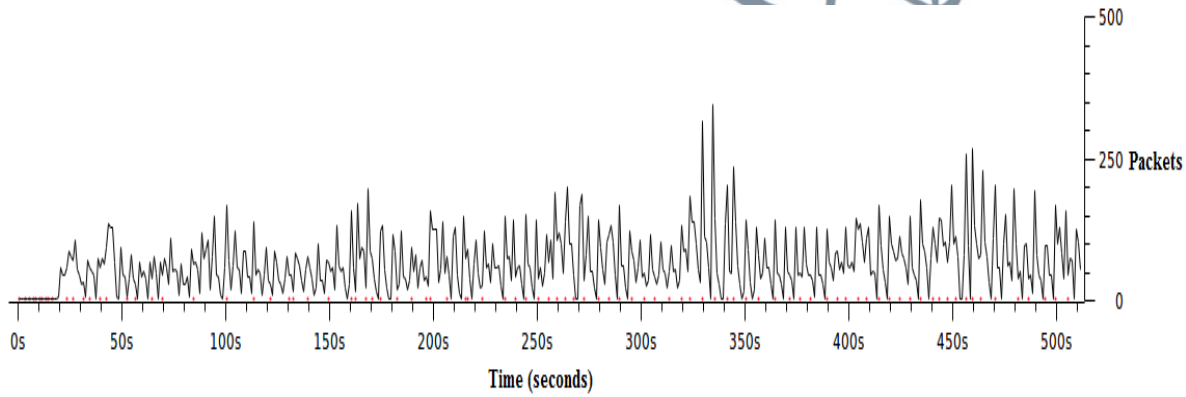
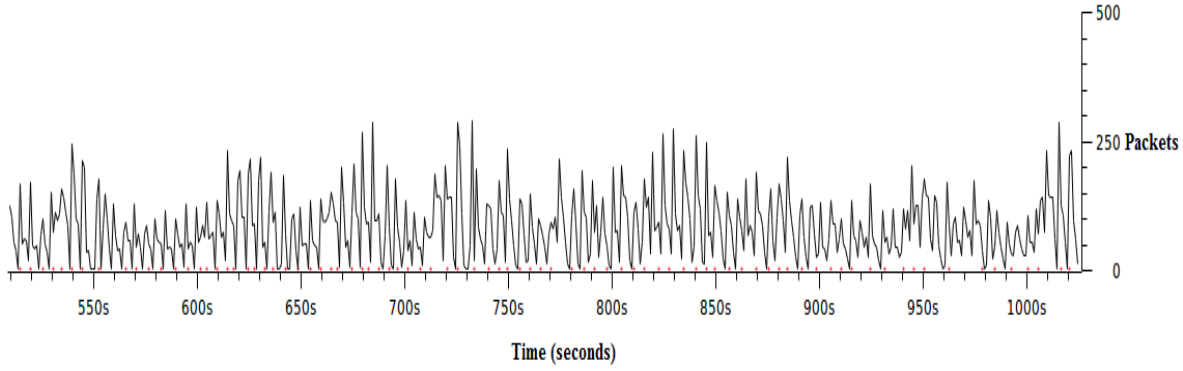


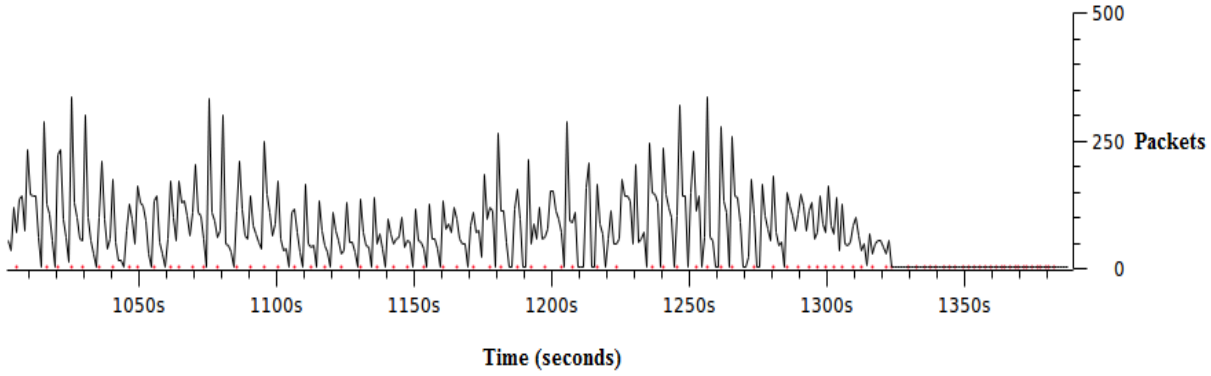
Figure 5.32: The list of destination IP addresses

In all graphs shown in Figure 5.33, the red dots represent the normal traffic and, the black line represents the low-rate attack. In these graphs, we used dots

instead of line to represent the normal traffic due to sending normal packets at an average of ten packets per second, and we need to clarify packets appearance. The graphs prove that the OpenFlow switch and the SDN controller have not affected by low-rate attack packets and were running smoothly at the time of the attack generation. These graphs practically prove that the scheme is efficiently protecting the OpenFlow switch from being overflown by a vast number of low-rate attack flows, and this is eventually protecting the SDN controller from bottlenecks and resources depleting.







Figures 5.33: The performance under the mixture of traffic in the long term

Figure 5.34 shows that the mixture of traffic generation has last for twenty-three minutes and seven seconds. Also, the summary provides details about packets that have sent in the mixture traffic scenario.

Time

First packet:	2019-04-28 00:16:40
Last packet:	2019-04-28 00:39:47
Elapsed:	00:23:07

Capture

OS:	Linux 3.13.0-27-generic
Capture application:	Dumpcap 1.12.1 (Git Rev Unknown from unknown)

Capture file comments

Interface	Dropped Packets	Capture Filter	Link type	Packet size limit
s1-eth4	unknown	none	Ethernet	262144 bytes

Display

Display filter:	none
Ignored packets:	0 (0.000%)

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	102257	102257	100.000%	0	0.000%
Between first and last packet	1387.326 sec				
Avg. packets/sec	73.708				
Avg. packet size	54 bytes				
Bytes	5501070	5501070	100.000%	0	0.000%
Avg. bytes/sec	3965.234				
Avg. MBit/sec	0.032				

Figure 5.34: Summary of the mixture traffic in the long term

5.2.2.4 Scheme Performance Evaluation (low-rate SYN attack scenario)

Due to, researches we found were few and do not provide the percentage of the performance enhancements achieved we provide performance achieved by the proposed scheme under low-rate DDoS attack in terms of CPU usage, accuracy, and false alarms.

5.2.2.4.1 Overhead (CPU Usage)

To examine the effect of the proposed scheme on the controller's CPU, we left the simulation running with all the running processes on a Linux PC (running Ubuntu 14.04).

We left the system monitoring application of Linux (htop) running during all the time of the experiment. Figure 5.35 shows the CPU usage while the proposed scheme is in use at the time of the low-rate attack traffic generation. The overhead measured is only 2.5%. The low overhead measured shows the scheme has a positive reflection on the CPU usage of the SDN controller.

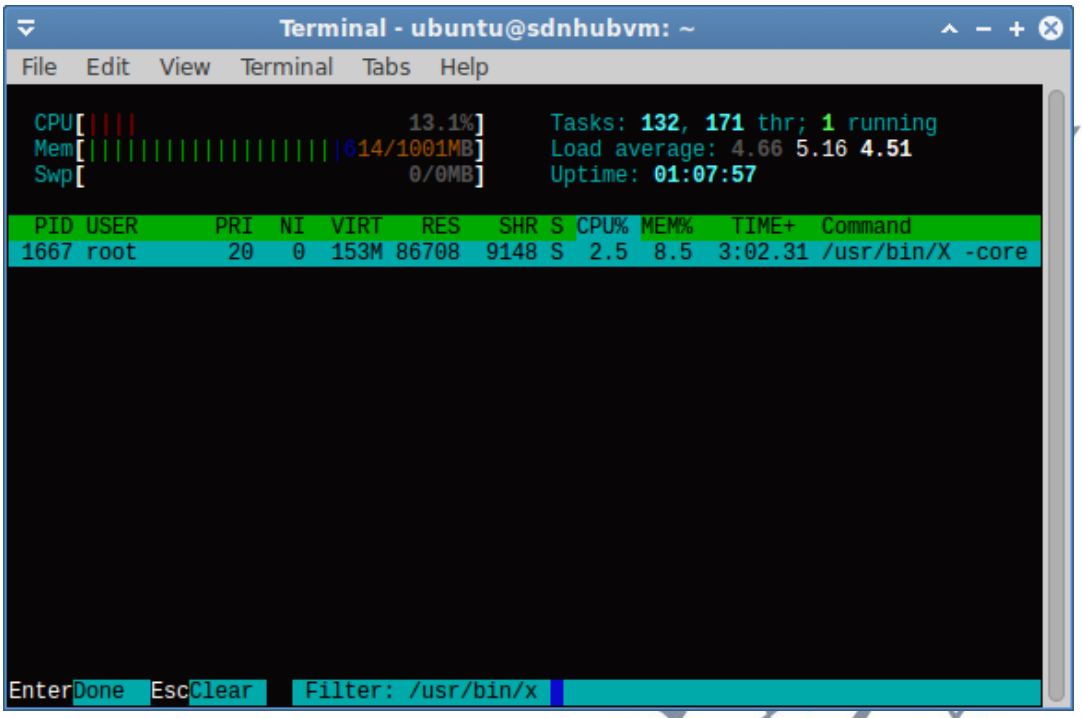


Figure 5.35: CPU usage under low-rate attack scenario

5.2.2.4.2 Accuracy

The accuracy achieved by the proposed scheme under the experimental scenario (Low-rate SYN and Normal traffic) is 99.47%. The accuracy is calculated as in chapter three, section 3.5.3.2. By considering that there are no false positives and false negatives, then the accuracy of detecting the low-rate attack packets will be 100% and no DDoS attack packet goes unnoticed. This result shows that this scheme can, easily and accurately, detect the attack packets that have been sent separately to overflow the switch by creating a vast number of flows.

Since this scheme has been tested under this experimental scenario for many times, no false positives have been appeared on the POX terminal at any time of the traffic generation. So, the false alarms produced by the scheme under this experimental scenario are 0.

5.2.3 Experiment Three (UDP Flood, Low-rate SYN and Normal traffic Scenario – long term)

In this experiment, a mixture of different DDoS attacks generated by different attack categories with normal traffic mixed in the same traffic generation. DDoS attacks launched in this experiment were UDP flood DDoS attack and low-rate SYN DDoS attack. Python scripts used to launch the attacks from hosts h4, h5 and h6 manually. While the host h4 used to launch the low-rate SYN attack, hosts h5 and h6 used to launch the UDP flood attack. The normal traffic generated in this experiment is normal TCP traffic and normal UDP traffic. Python scripts used to generate the normal traffic from hosts h1 and h3. The host h1 has used to generate normal TCP traffic whereas host h3 has used to generate normal UDP traffic.

This experiment is meant to evaluate the performance of the proposed scheme under different DDoS attacks attacking at the same time and, their packets are mixed with normal packets in the same traffic generation.

It is important to mention here that the proposed scheme will be under a low-rate DDoS attack and a flooding DDoS attack (high rate attack). This means that the scheme has to protect the SDN components from performance degradation with high accuracy and low overhead.

On the POX controller terminal, no actions have displayed by the scheme although scripts that generate the normal traffic are currently running. Figure 5.36 shows the terminal at the first eighty seconds of normal traffic generation.

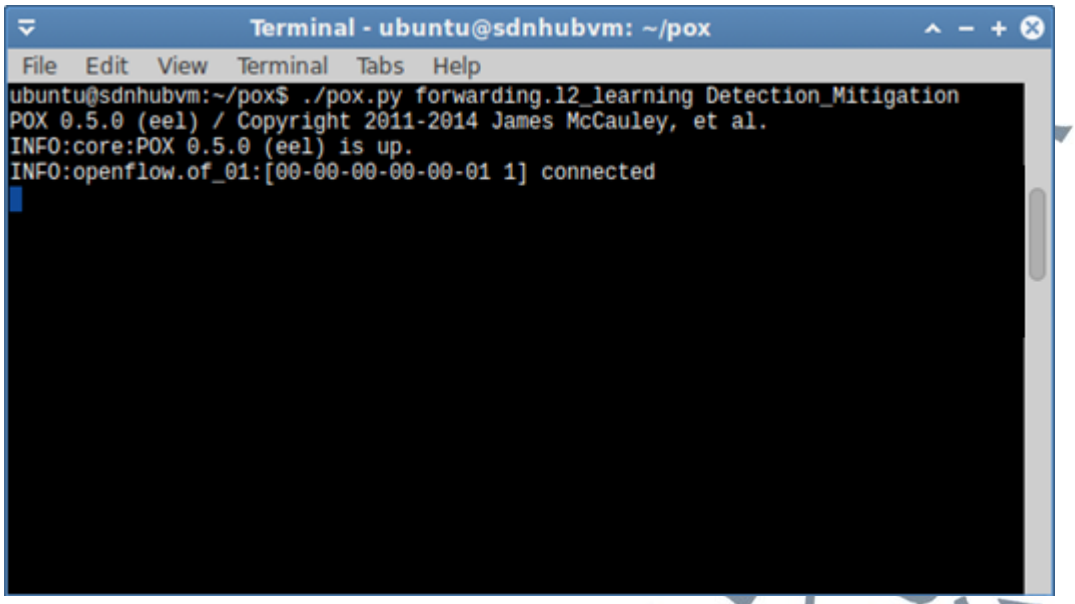


Figure 5.36: POX controller terminal within the first eighty seconds

After eighty-two seconds of generating normal traffic, mixture traffic of low-rate DDoS attack packets and flooding DDoS attack packets have launched from attack hosts along with normal traffic that is still generating. Figures from 5.37 to 5.46 show the detection and mitigation processes of both the low-rate attack packets and the flooding attack packets at different time slots after the attack traffic began generating.

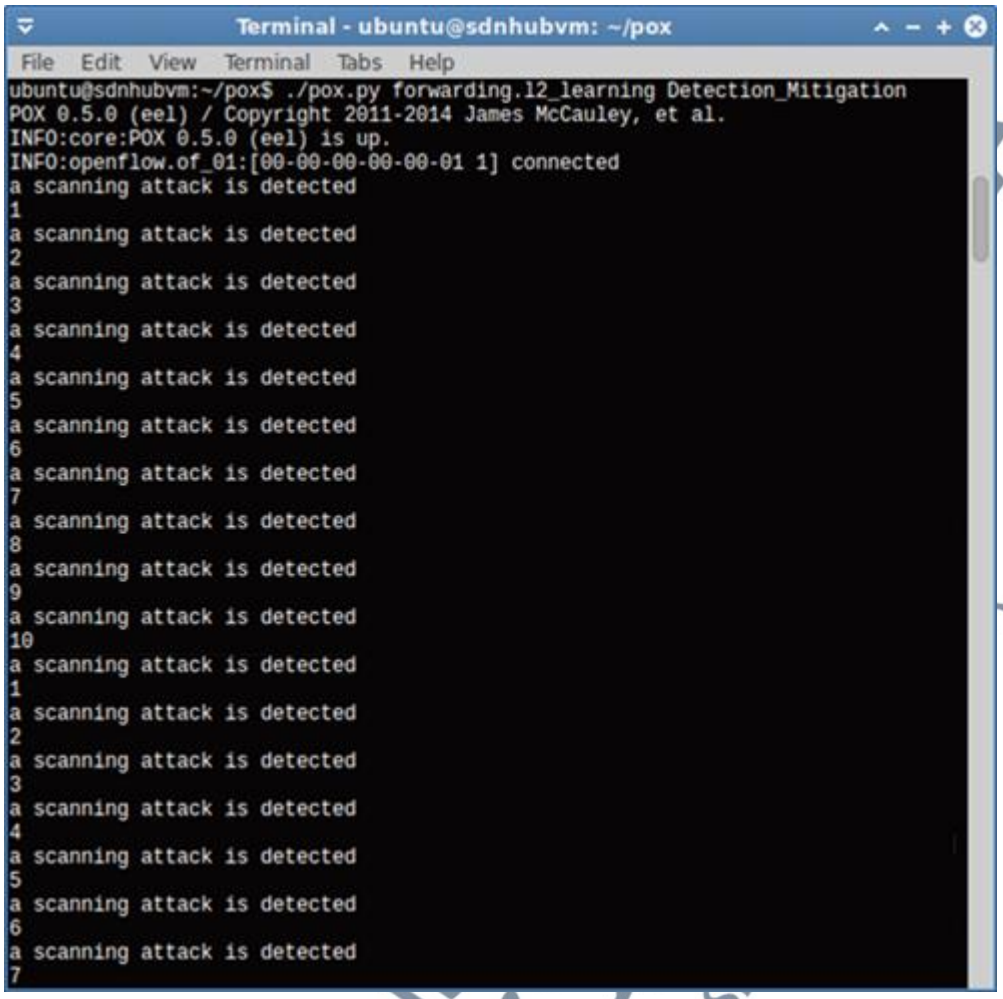


Figure 5.37: Results after eighty-two seconds

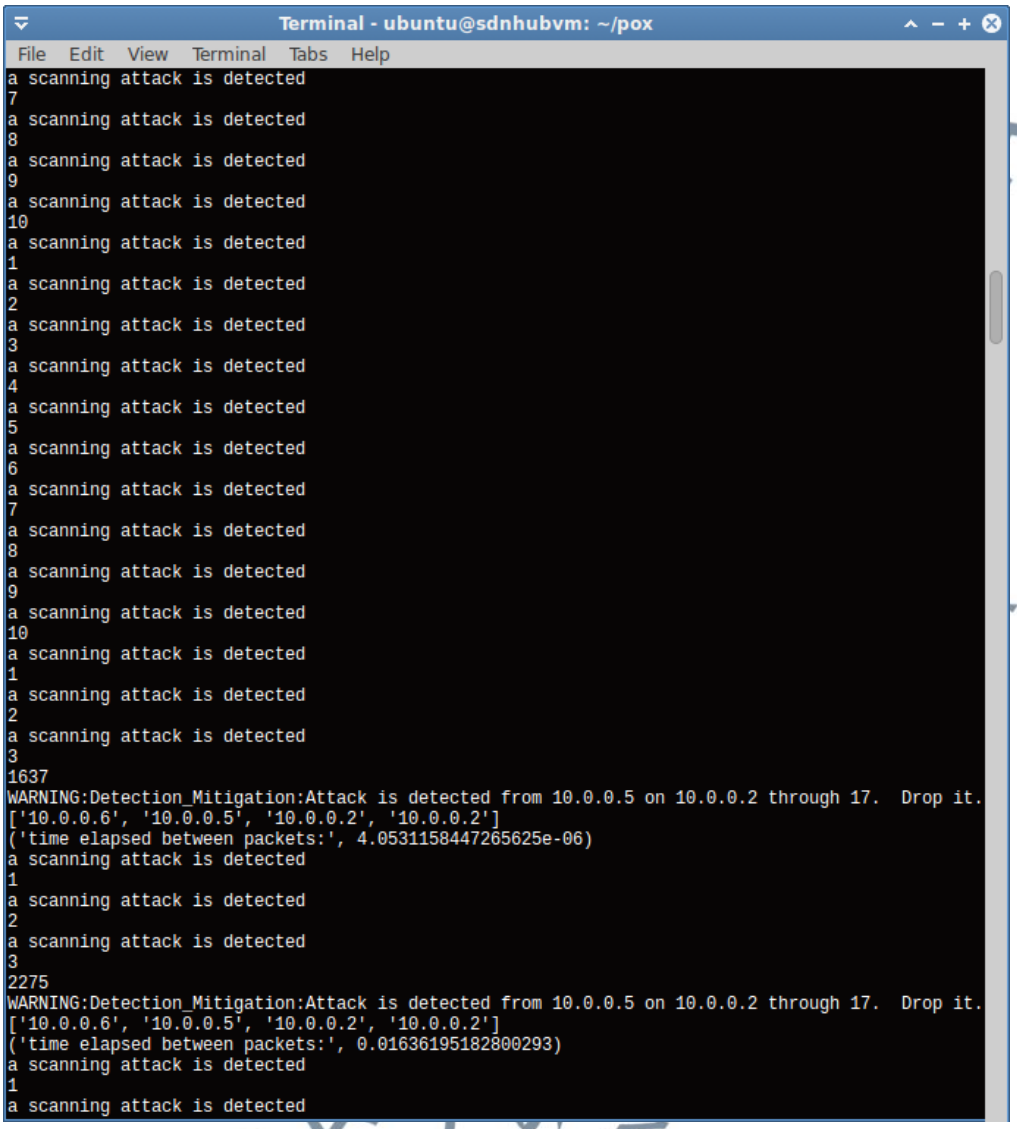
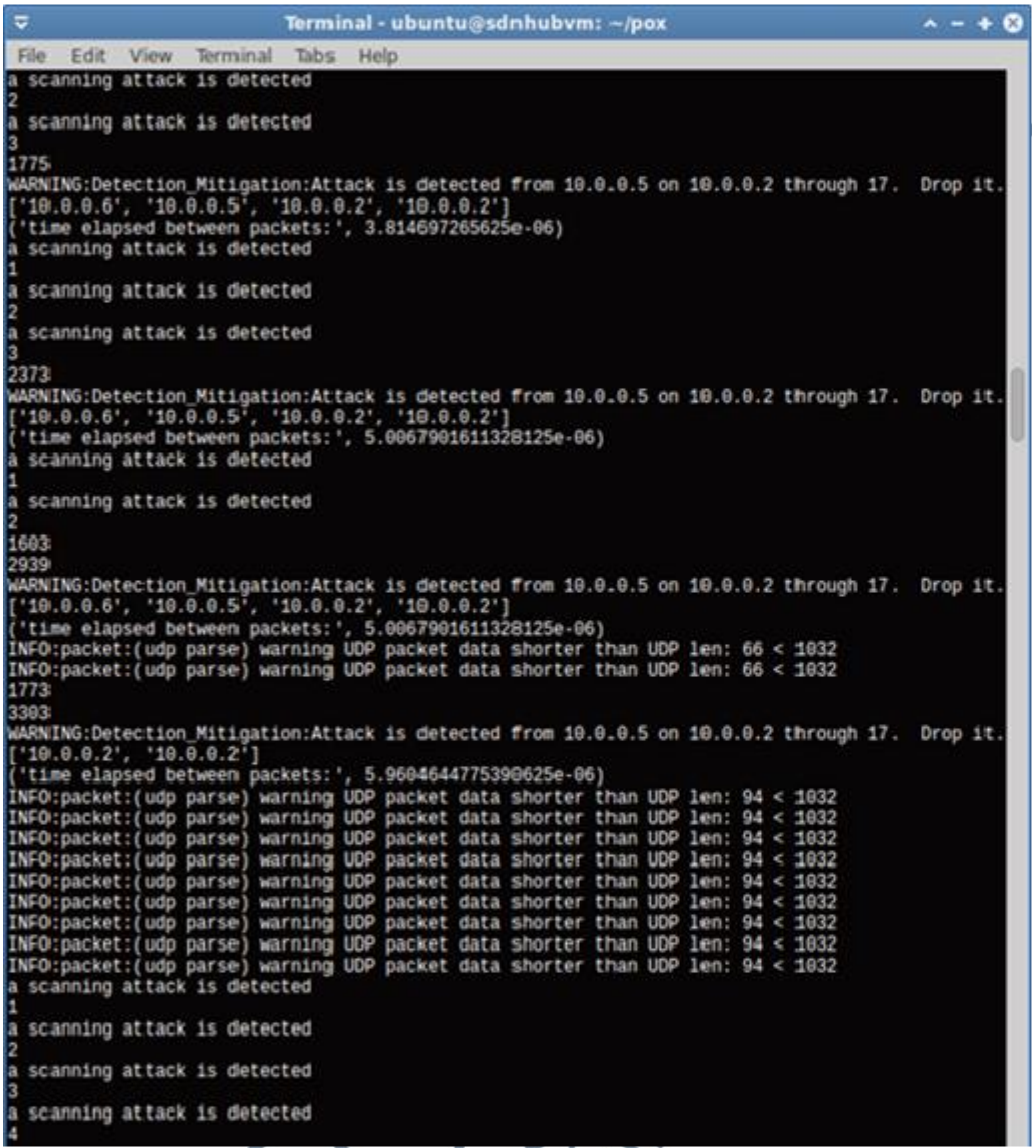


Figure 5.38: Results after three minutes



```
Terminal - ubuntu@sdnhubvm: ~/pox
File  Edit  View  Terminal  Tabs  Help
a scanning attack is detected
2
a scanning attack is detected
3
1775
WARNING:Detection_Mitigation:Attack is detected from 10.0.0.5 on 10.0.0.2 through 17.  Drop it.
['10.0.0.6', '10.0.0.5', '10.0.0.2', '10.0.0.2']
('time elapsed between packets:', 3.814697265625e-06)
a scanning attack is detected
1
a scanning attack is detected
2
a scanning attack is detected
3
2373
WARNING:Detection_Mitigation:Attack is detected from 10.0.0.5 on 10.0.0.2 through 17.  Drop it.
['10.0.0.6', '10.0.0.5', '10.0.0.2', '10.0.0.2']
('time elapsed between packets:', 5.0067901611328125e-06)
a scanning attack is detected
1
a scanning attack is detected
2
1603
2939
WARNING:Detection_Mitigation:Attack is detected from 10.0.0.5 on 10.0.0.2 through 17.  Drop it.
['10.0.0.6', '10.0.0.5', '10.0.0.2', '10.0.0.2']
('time elapsed between packets:', 5.0067901611328125e-06)
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 66 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 66 < 1032
1773
3303
WARNING:Detection_Mitigation:Attack is detected from 10.0.0.5 on 10.0.0.2 through 17.  Drop it.
['10.0.0.2', '10.0.0.2']
('time elapsed between packets:', 5.9604644775390625e-06)
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
INFO:packet:(udp parse) warning UDP packet data shorter than UDP len: 94 < 1032
a scanning attack is detected
1
a scanning attack is detected
2
a scanning attack is detected
3
a scanning attack is detected
4
```

Figure 5.39: Results after five minutes

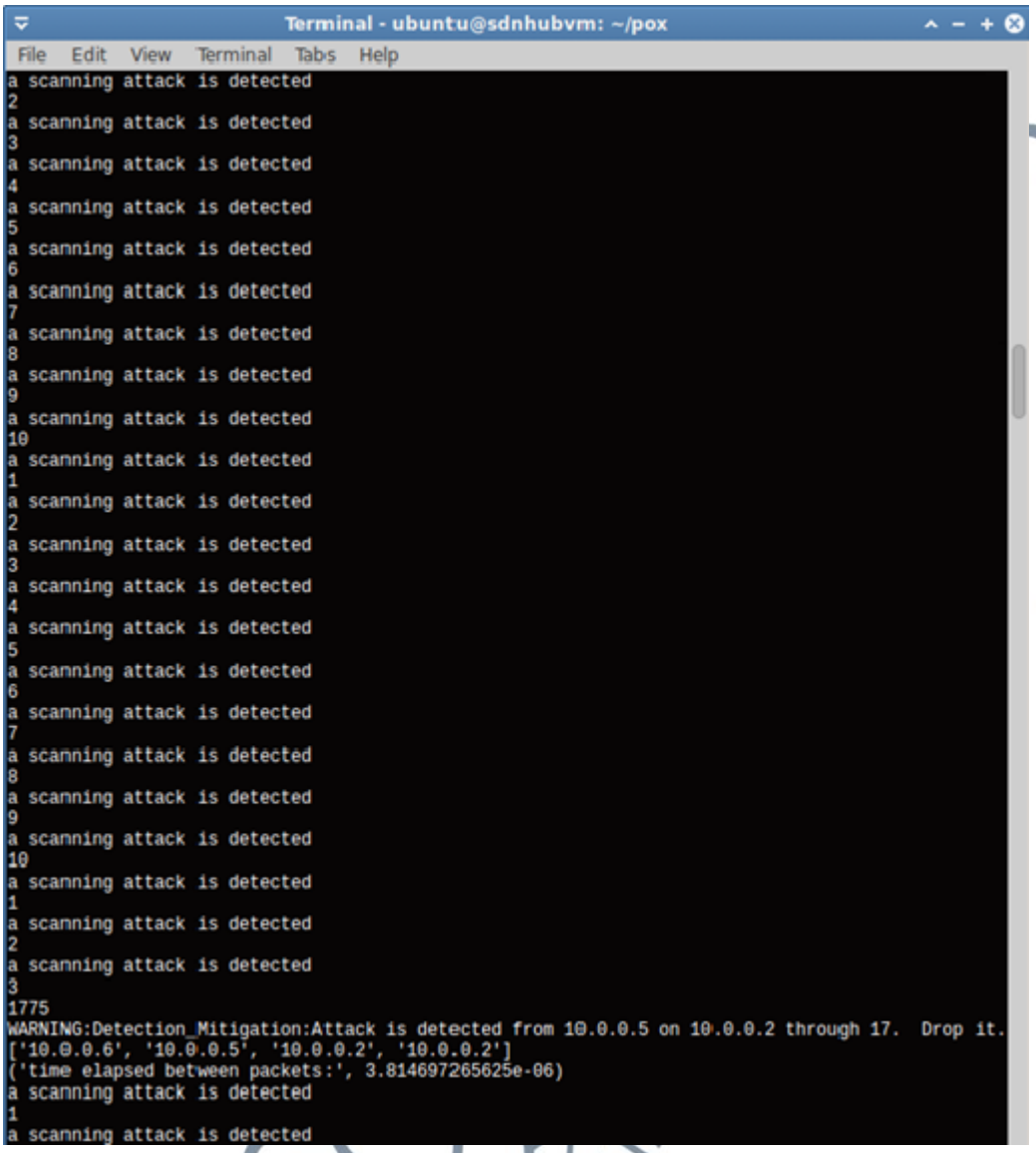


Figure 5.40: Results after seven minutes

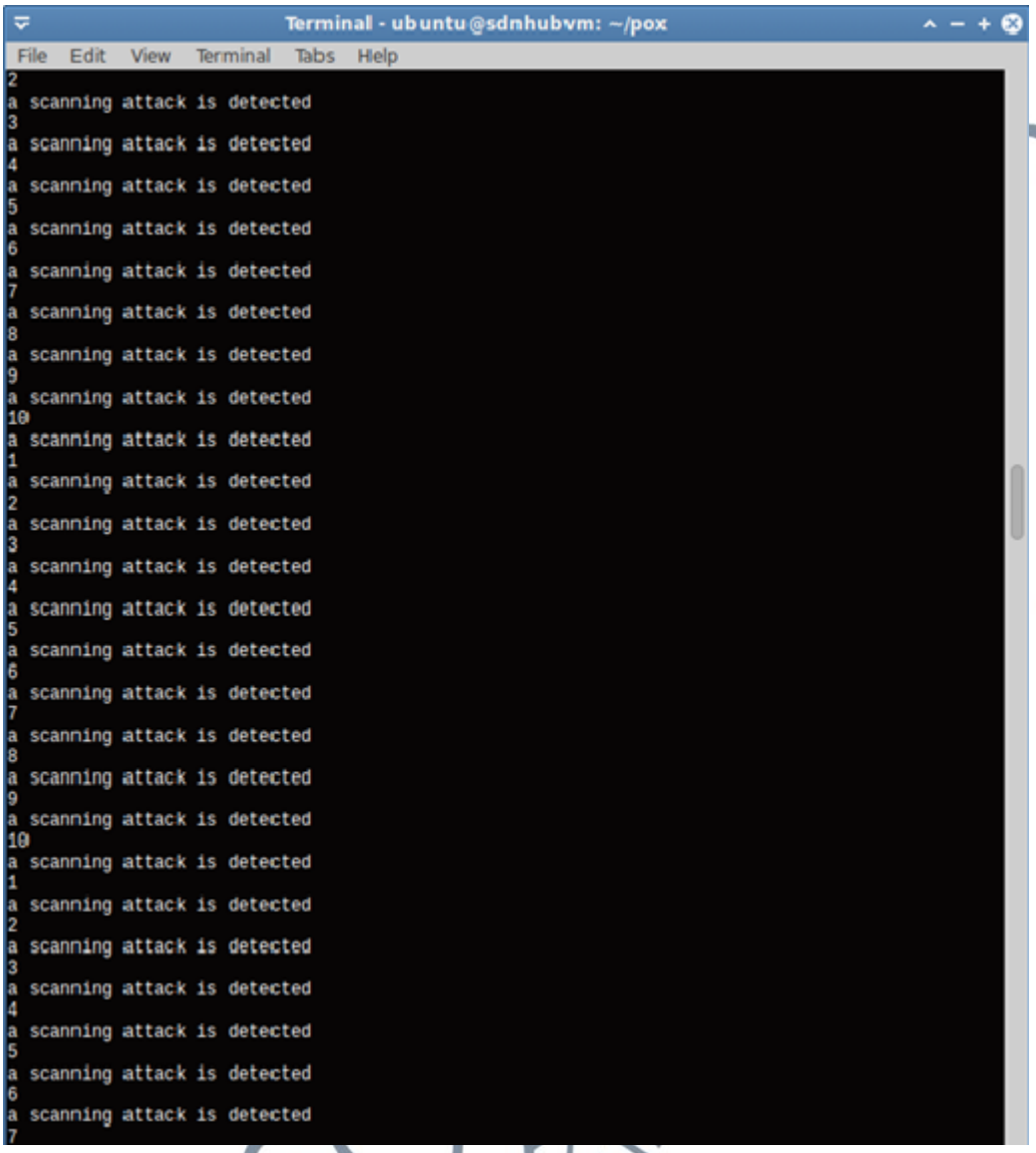


Figure 5.41: Results after ten minutes

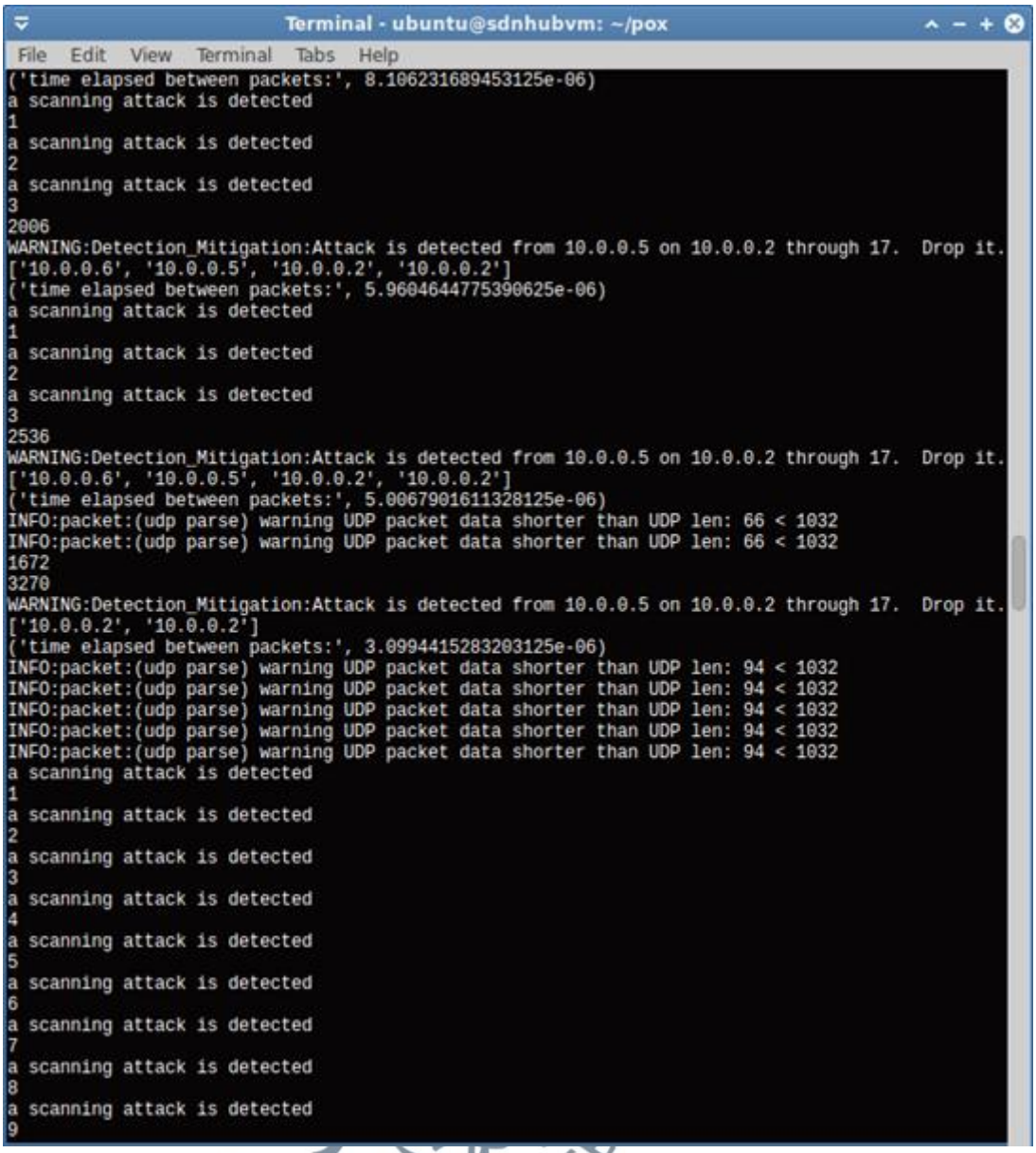


Figure 5.42: Results after thirteen minutes

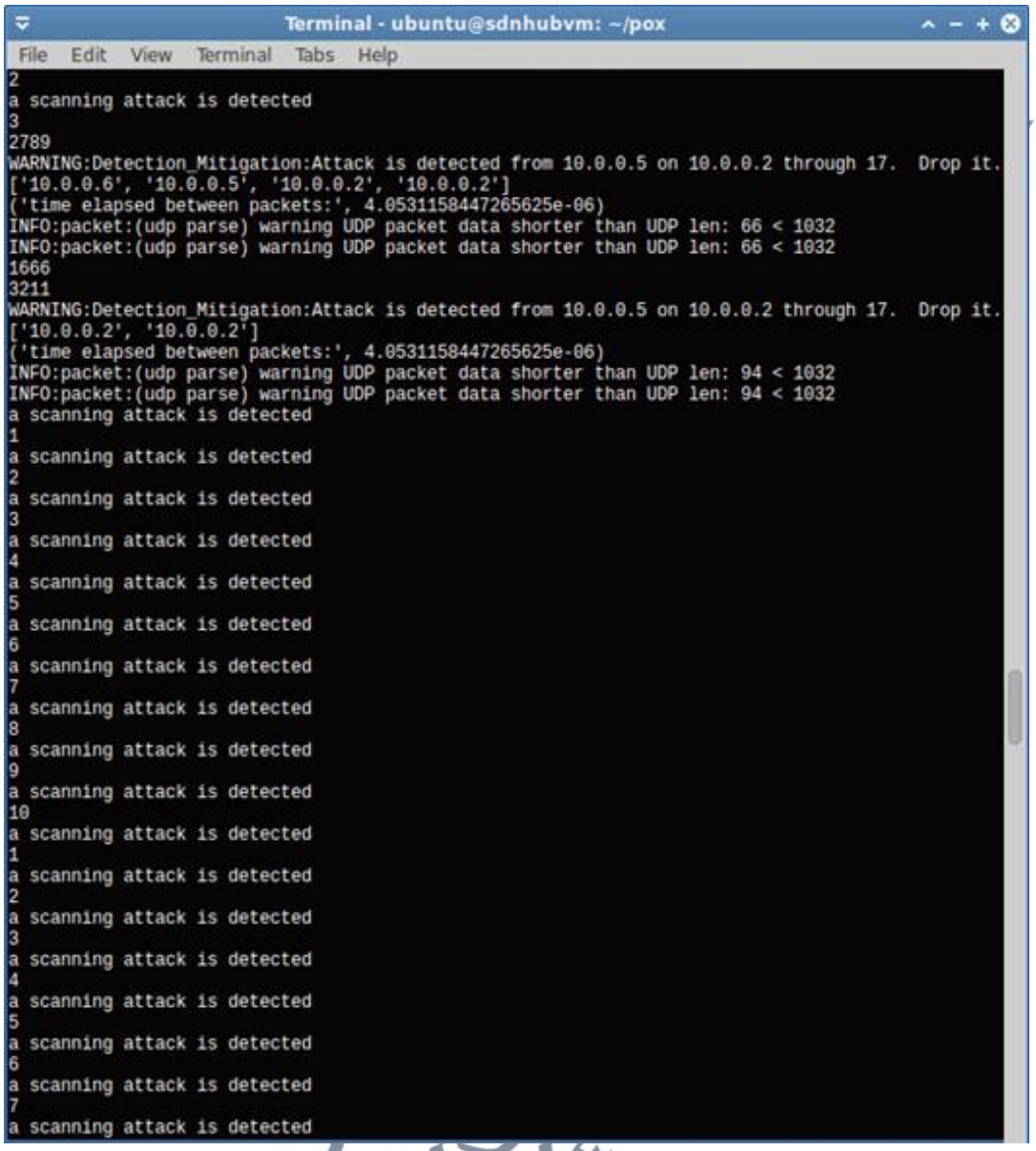


Figure 5.43: Results after fifteen minutes

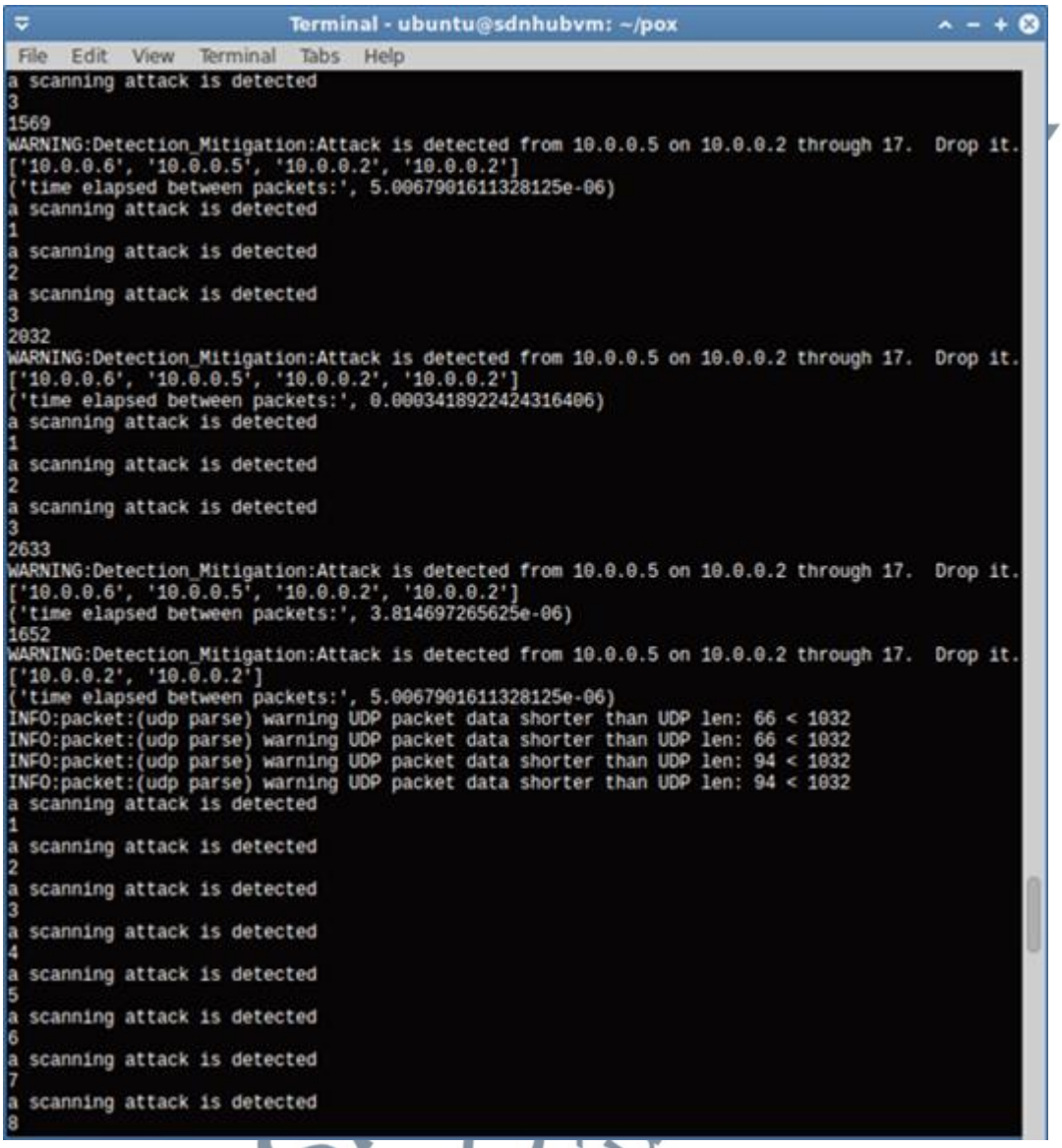


Figure 5.44: Results after eighteen minutes

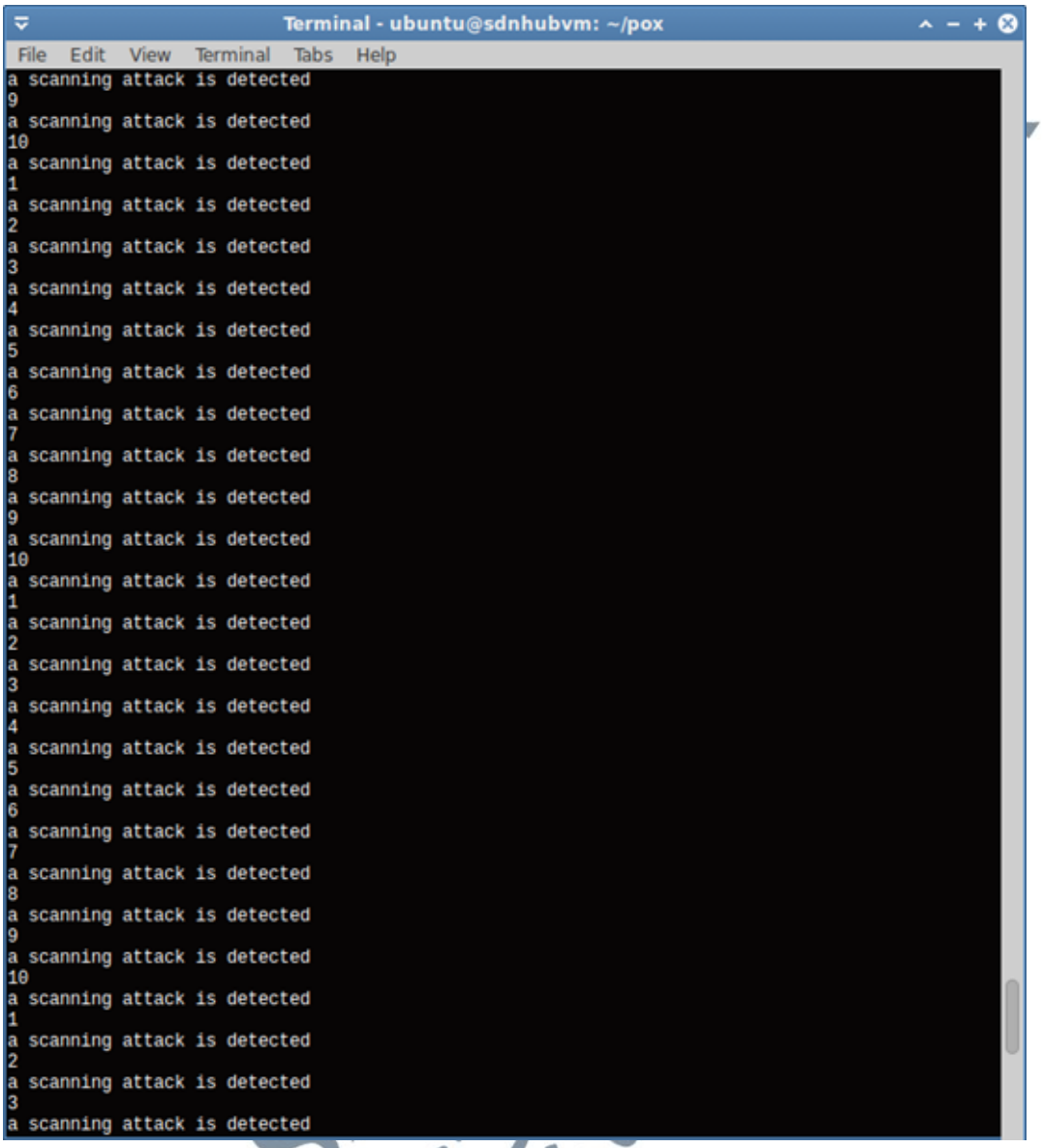


Figure 5.45: Results after nineteen minutes

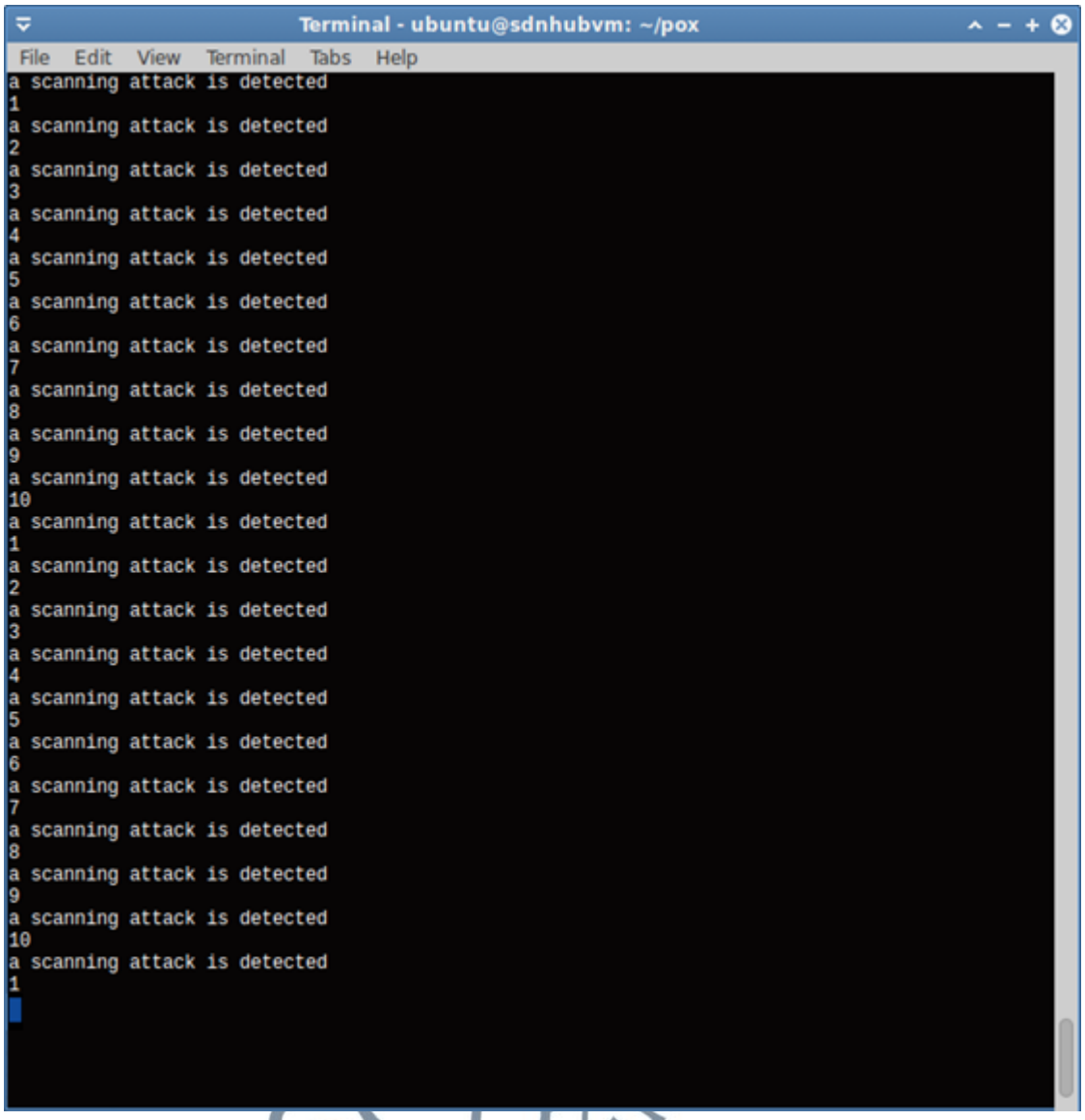
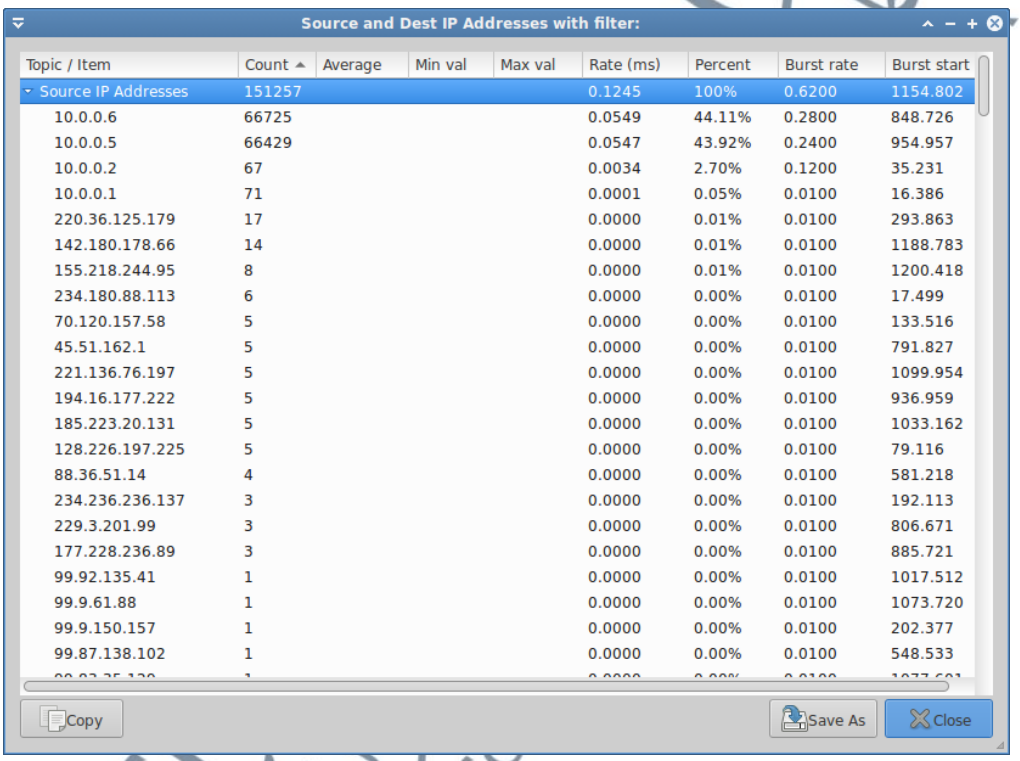


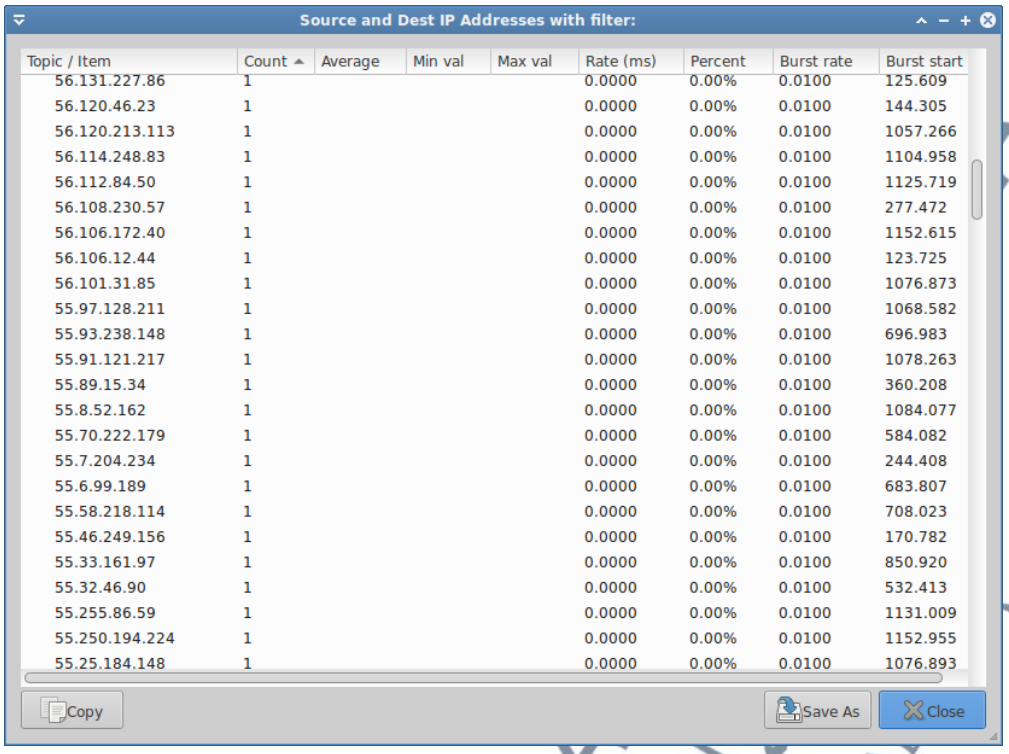
Figure 5.46: The last results

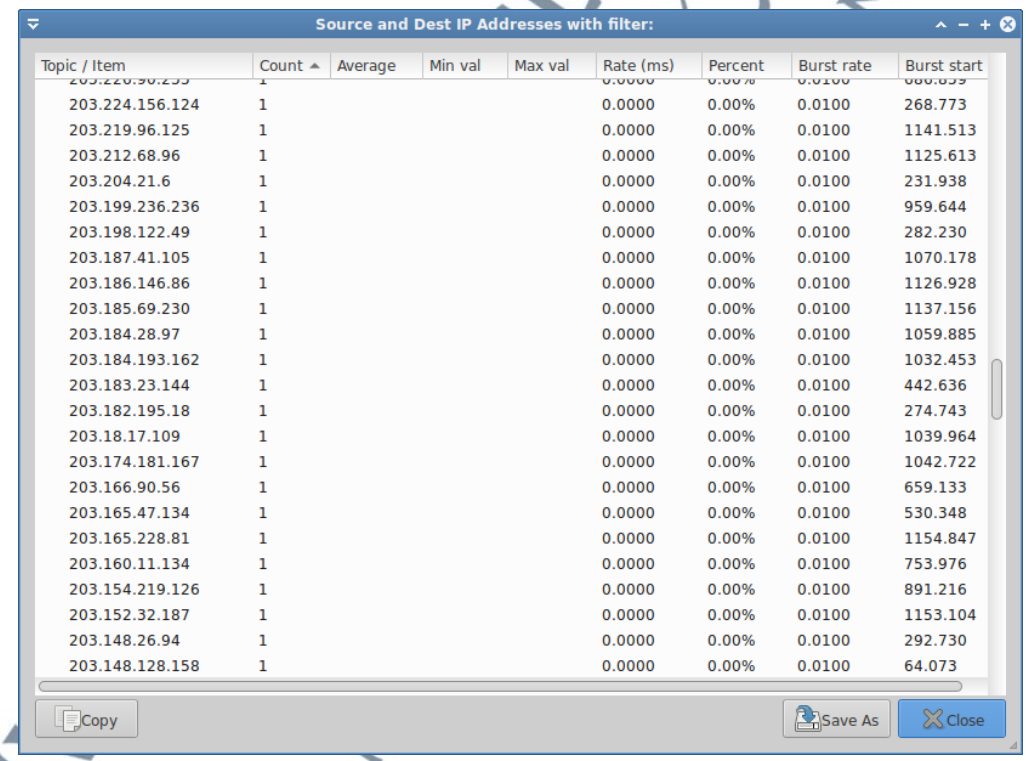
As shown in the Figures from 5.37 to 5.46, the scheme detects and mitigates both the low-rate attack packets and flooding attack packets. Although the normal traffic is generated in the network, the IP addresses of the hosts h1 and h3 (10.0.0.1 and 10.0.0.3) did not appear in the list provided by the scheme during all the time of traffic generation at all while the IP addresses of the hosts h5 and h6 used for generating the flooding attacks appeared. This means that no false alarms have been produced at any second of the traffic generation. The host h4 is the attack machine responsible for launching the low-rate attack packets but, it did not appear on any

figure of the mentioned figures. The reason is that host h4 launched the low-rate attack packets separately at a low rate to increase the number of attack flows. For this, we notice lots of attack flows have been detected and removed as shown on the terminals.

The list of the source IP addresses appeared in Figures 5.47 show that the packets that have sent in the network have generated from different spoofed source IP addresses.




Source and Dest IP Addresses with filter:
Topic / Item
Count
Average
Min val
Max val
Rate (ms)
Percent
Burst rate
Burst start
56.131.227.86 1 0.0000 0.00% 0.0100 125.609
56.120.46.23 1 0.0000 0.00% 0.0100 144.305
56.120.213.113 1 0.0000 0.00% 0.0100 1057.266
56.114.248.83 1 0.0000 0.00% 0.0100 1104.958
56.112.84.50 1 0.0000 0.00% 0.0100 1125.719
56.108.230.57 1 0.0000 0.00% 0.0100 277.472
56.106.172.40 1 0.0000 0.00% 0.0100 1152.615
56.106.12.44 1 0.0000 0.00% 0.0100 123.725
56.101.31.85 1 0.0000 0.00% 0.0100 1076.873
55.97.128.211 1 0.0000 0.00% 0.0100 1068.582
55.93.238.148 1 0.0000 0.00% 0.0100 696.983
55.91.121.217 1 0.0000 0.00% 0.0100 1078.263
55.89.15.34 1 0.0000 0.00% 0.0100 360.208
55.8.52.162 1 0.0000 0.00% 0.0100 1084.077
55.70.222.179 1 0.0000 0.00% 0.0100 584.082
55.7.204.234 1 0.0000 0.00% 0.0100 244.408
55.6.99.189 1 0.0000 0.00% 0.0100 683.807
55.58.218.114 1 0.0000 0.00% 0.0100 708.023
55.46.249.156 1 0.0000 0.00% 0.0100 170.782
55.33.161.97 1 0.0000 0.00% 0.0100 850.920
55.32.46.90 1 0.0000 0.00% 0.0100 532.413
55.255.86.59 1 0.0000 0.00% 0.0100 1131.009
55.250.194.224 1 0.0000 0.00% 0.0100 1152.955
55.25.184.148 1 0.0000 0.00% 0.0100 1076.893
Copy
Save As
Close


Source and Dest IP Addresses with filter:
Topic / Item
Count
Average
Min val
Max val
Rate (ms)
Percent
Burst rate
Burst start
203.224.156.124 1 0.0000 0.00% 0.0100 268.773
203.219.96.125 1 0.0000 0.00% 0.0100 1141.513
203.212.68.96 1 0.0000 0.00% 0.0100 1125.613
203.204.21.6 1 0.0000 0.00% 0.0100 231.938
203.199.236.236 1 0.0000 0.00% 0.0100 959.644
203.198.122.49 1 0.0000 0.00% 0.0100 282.230
203.187.41.105 1 0.0000 0.00% 0.0100 1070.178
203.186.146.86 1 0.0000 0.00% 0.0100 1126.928
203.185.69.230 1 0.0000 0.00% 0.0100 1137.156
203.184.28.97 1 0.0000 0.00% 0.0100 1059.885
203.184.193.162 1 0.0000 0.00% 0.0100 1032.453
203.183.23.144 1 0.0000 0.00% 0.0100 442.636
203.182.195.18 1 0.0000 0.00% 0.0100 274.743
203.18.17.109 1 0.0000 0.00% 0.0100 1039.964
203.174.181.167 1 0.0000 0.00% 0.0100 1042.722
203.166.90.56 1 0.0000 0.00% 0.0100 659.133
203.165.47.134 1 0.0000 0.00% 0.0100 530.348
203.165.228.81 1 0.0000 0.00% 0.0100 1154.847
203.160.11.134 1 0.0000 0.00% 0.0100 753.976
203.154.219.126 1 0.0000 0.00% 0.0100 891.216
203.152.32.187 1 0.0000 0.00% 0.0100 1153.104
203.148.26.94 1 0.0000 0.00% 0.0100 292.730
203.148.128.158 1 0.0000 0.00% 0.0100 64.073
Copy
Save As
Close

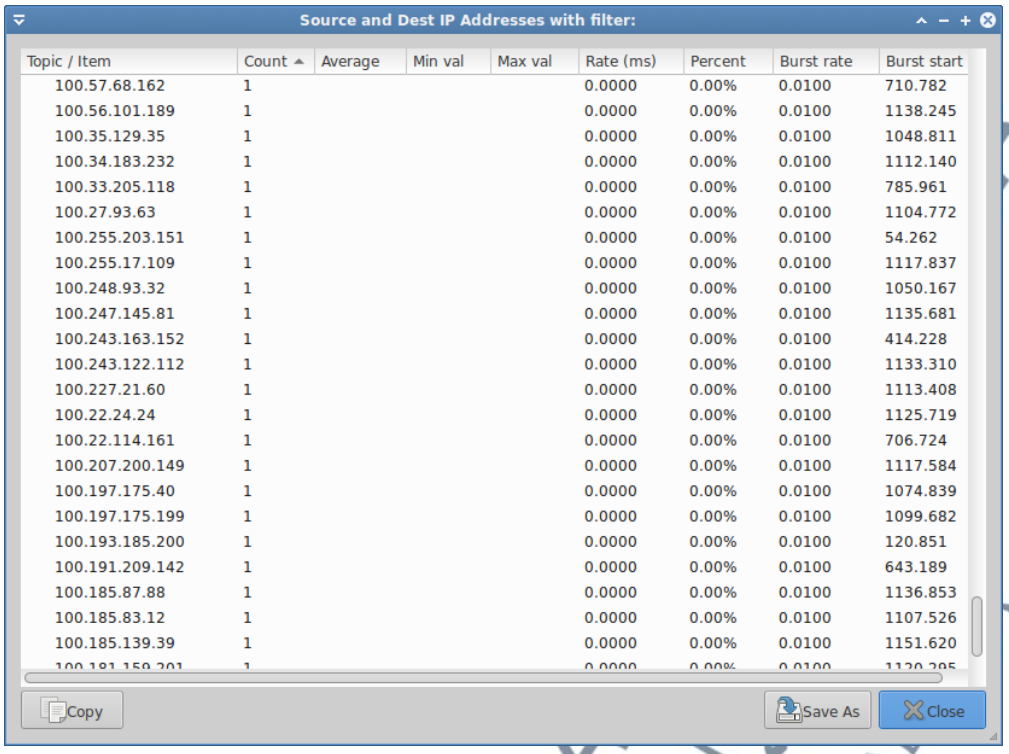


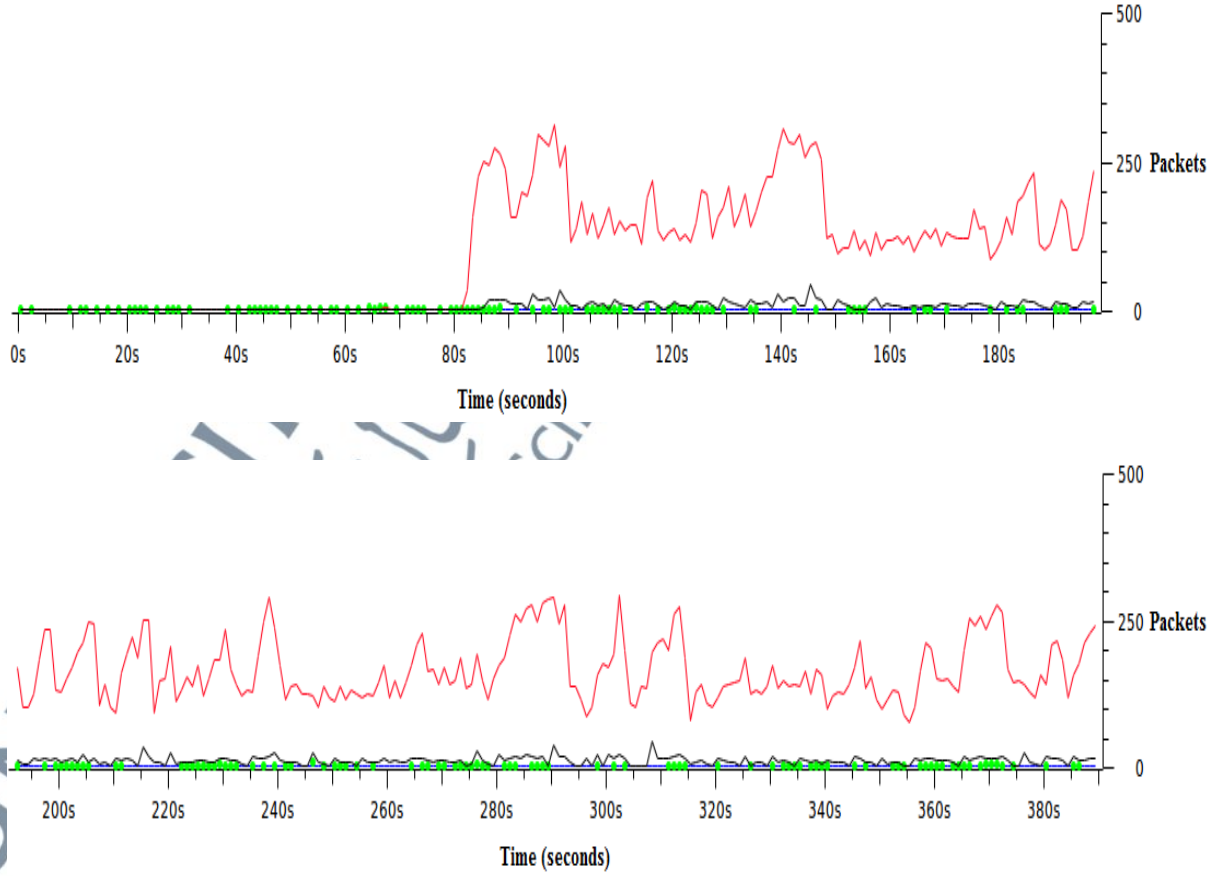
Figure 5.47: List of the source IP addresses

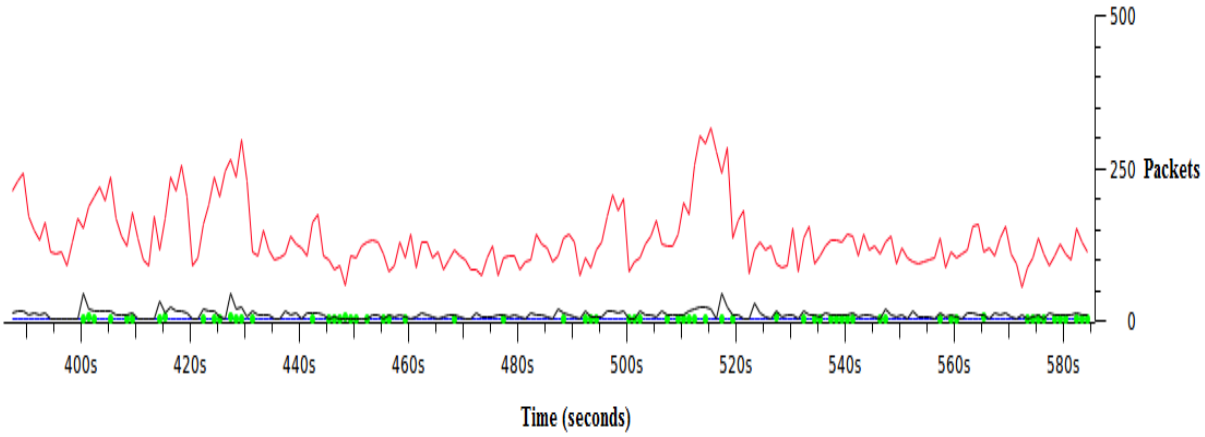
The two source IP addresses (10.0.0.6 and 10.0.0.5) that sent a large number of packets were the IP addresses of hosts that generated the flooding attack traffic while the next two source IP addresses in the list (10.0.0.2 and 10.0.0.1), the IP addresses that established normal TCP handshake. It is important to mention that IP address 10.0.0.2 is the victim's IP address but it appears in the list as a source IP address because it is one of the two sides in the TCP connection establishment.

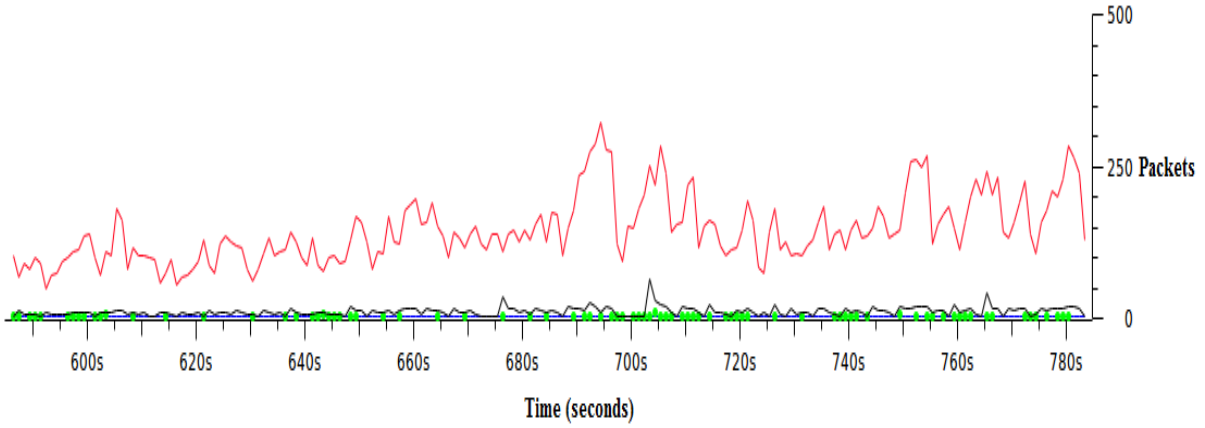
The IP address 10.0.0.1 is the IP address generates normal TCP traffic and the next source IP addresses until the IP address 177.228.236.89 are the IP addresses generate normal UDP traffic. The rest of the spoofed source IP addresses that appeared in the list are the IP addresses that generate the low-rate SYN attack packets. These packets sent separately to increase the number of attack flows.

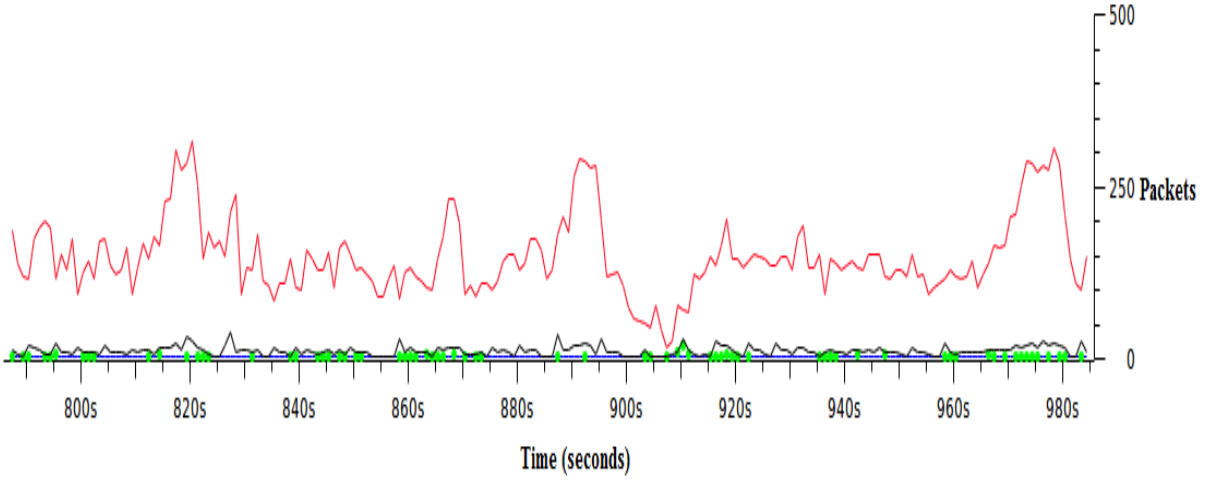
In all graphs appeared in Figure 5.48, the red line represents the UDP flood attack and the black line represents the low-rate SYN attack. In these graphs, we also

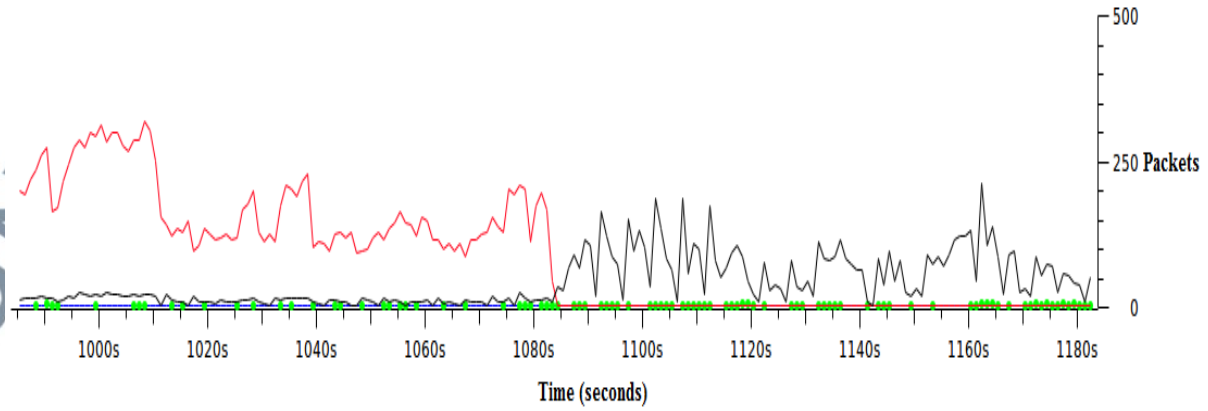
used another two different colours to represent the normal traffic. The blue line represents the normal TCP traffic and the green represents the normal UDP traffic. In all graphs, we can notice that no sharp increase in the green dots or the blue line during the time of traffic generation due to generating the normal packets at an average of not greater than ten packets per second from true source IP addresses. We can also notice that sharp increase presented by the graphs clearly appeared in the red lines more than the black lines due to the basic difference between the flooding attacks and the low-rate attacks in the attack rate. The flooding attacks send the attack packets in a large number of packets at a high average of rate whereas the low-rate attacks send the attack packets at a low average of rate. In addition, the low-rate attack used in the experimental scenarios is designed to send the attack packets separately to increase the number of flows.




500
250 Packets
0
400s
420s
440s
460s
480s
500s
520s
540s
560s
580s
Time (seconds)


500
250 Packets
0
600s
620s
640s
660s
680s
700s
720s
740s
760s
780s
Time (seconds)


500
250 Packets
0
800s
820s
840s
860s
880s
900s
920s
940s
960s
980s
Time (seconds)


500
250 Packets
0
1000s
1020s
1040s
1060s
1080s
1100s
1120s
1140s
1160s
1180s
Time (seconds)

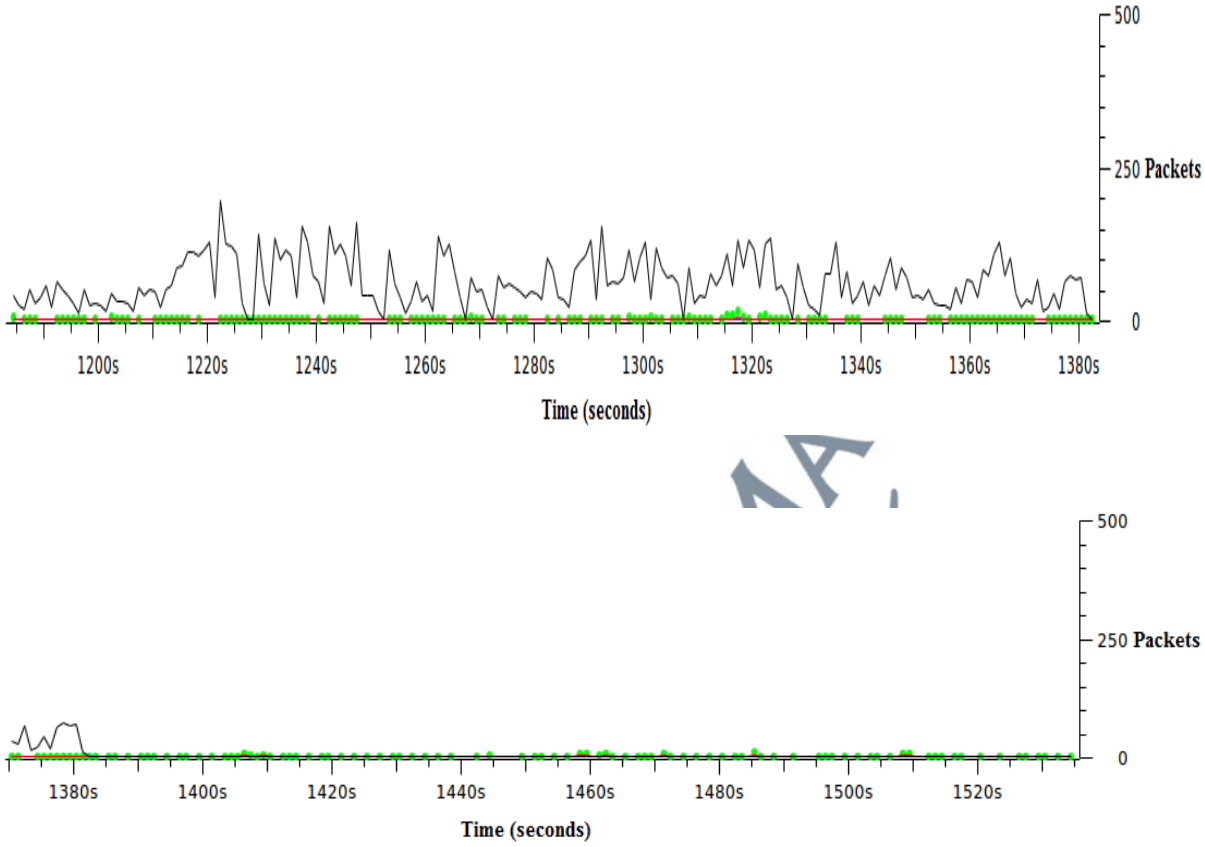


Figure 5.48: The performance under the mixture traffic (long term)

We can observe from all graphs that the scheme succeeded to protect the OpenFlow switch and the SDN controller from different types of DDoS attacks each of which follows a different strategy of attack during the time of traffic generation. Keep the SDN controller and the OpenFlow switch running smoothly during the time of the attack generation proves that the scheme accurately detects and mitigates the attack packets. The scheme stops taking actions against incoming packets once the DDoS scripts stop.

Figure 5.49 shows that the mixture of traffic generation has last for twenty-five minutes and thirty-four seconds. Also, the summary provides details about packets that have sent in the mixture traffic scenario.

Time

First packet:	2019-04-29 02:46:30
Last packet:	2019-04-29 03:12:04
Elapsed:	00:25:34

Capture

OS:	Linux 3.13.0-27-generic
Capture application:	Dumpcap 1.12.1 (Git Rev Unknown from unknown)

Capture file comments

Interface	Dropped Packets	Capture Filter	Link type	Packet size limit
s1-eth2	unknown	none	Ethernet	262144 bytes

Display

Display filter:	none
Ignored packets:	0 (0.000%)

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	151979	151979	100.000%	0	0.000%
Between first and last packet	1534.231 sec				
Avg. packets/sec	118.612				
Avg. packet size	890 bytes				
Bytes	162049958	162049958	100.000%	0	0.000%
Avg. bytes/sec	105622.889				
Avg. MBit/sec	0.845				

Figure 5.49: Summary of the mixture traffic (long term)

5.2.3.1 Scheme Performance Evaluation (UDP flood, Low-rate SYN and Normal traffic scenario – long term)

5.2.3.1.1 Overhead (CPU Usage)

To examine the effect of the proposed scheme on the controller's CPU, we left the simulation running with all the running processes on a Linux PC (running Ubuntu 14.04).

We left the system monitoring application of Linux (htop) running during all the time of the experiment. Figure 5.50 shows the CPU usage while the proposed scheme is in use at the time of mixture traffic (UDP flood, low-rate SYN and Normal traffic) generation. The overhead measured is only 29.9%. The low overhead

measured shows the scheme has a positive reflection on the CPU usage of the SDN controller.

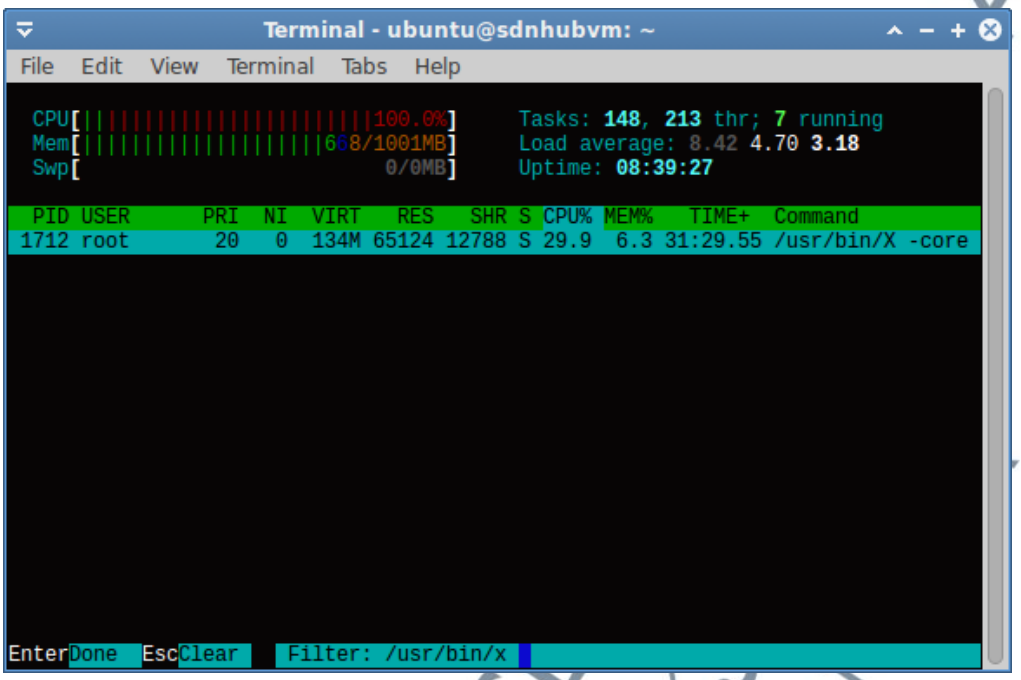


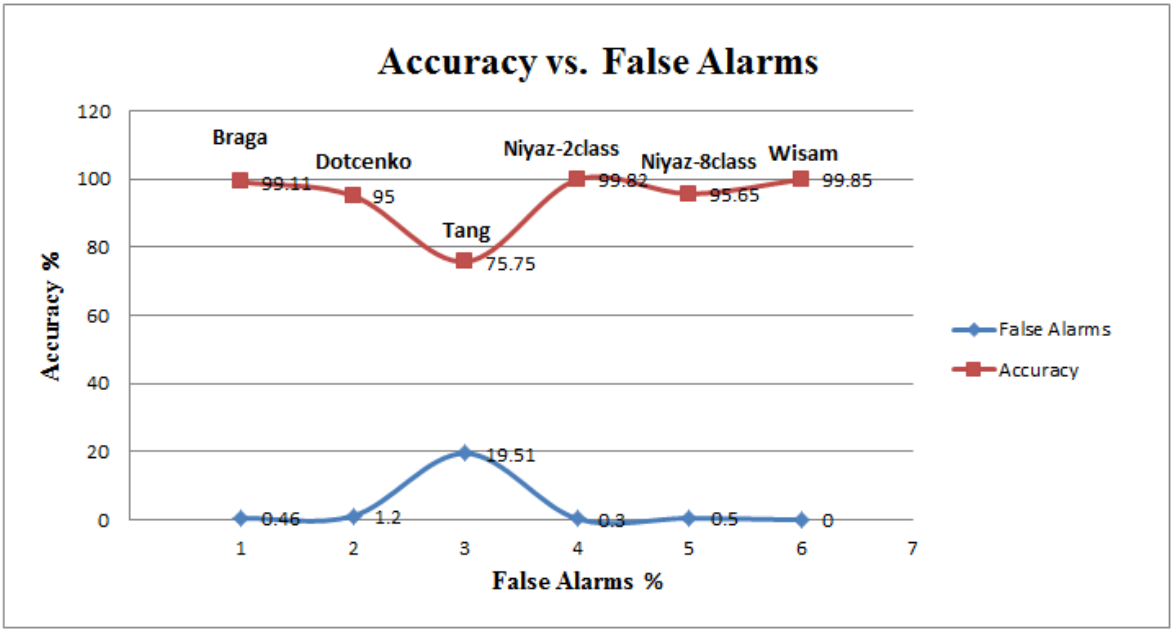
Figure 5.50: CPU usage under the mixture traffic

5.2.3.1.2 Accuracy

The accuracy achieved by the proposed scheme under the experimental scenario (UDP flood, low-rate SYN, and Normal traffic) is 99.85%. The accuracy is calculated as in chapter three, section 3.5.3.2. By considering that the last number of attack packets appeared in the last terminal was dropped by the scheme to be less than the threshold, the accuracy achieved in this scenario will be 100%. This result shows that this scheme can, easily, detect both the attack packets and attack flows when it is destined to a victim in the network.

Since this scheme has been tested under this experimental scenario many times, no false positives or false negatives have appeared on the POX terminal at any time of the traffic generation. So, the false alarms produced by the scheme under this experimental scenario are 0.

The comparison between the accuracy achieved and the false alarms produced by the proposed scheme and, the accuracy and false alarms rates of Machine Learning and Entropy solutions presented in literature review chapter are plotted in the Figure 5.51.



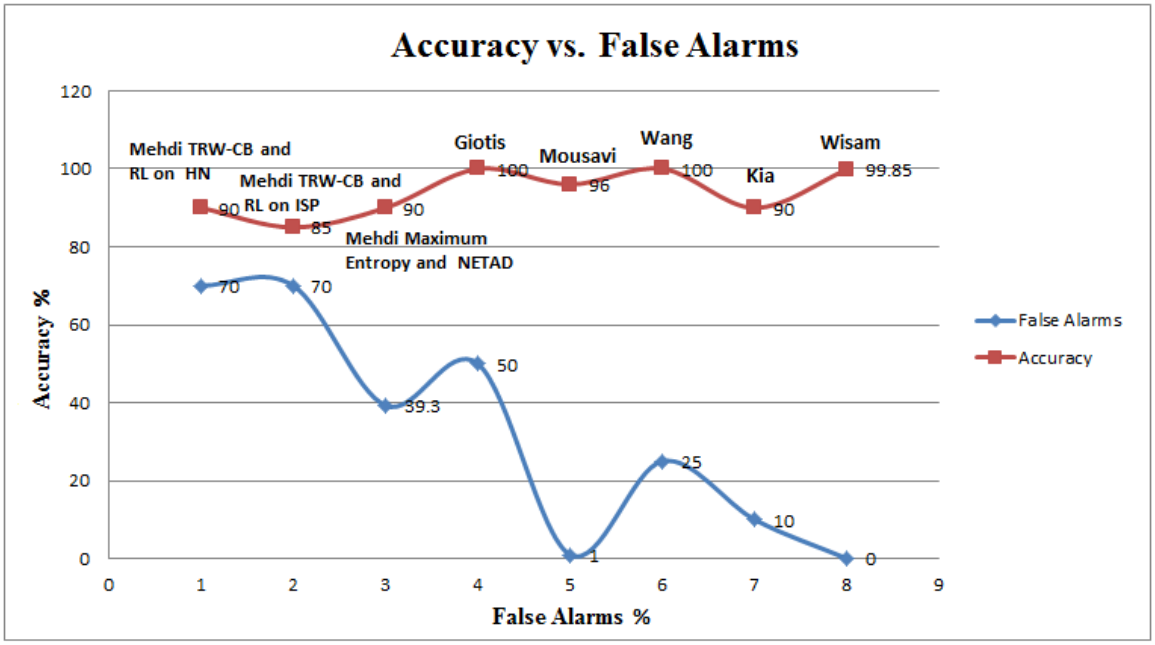


Figure 5.51: An elapsed-time based scheme vs. Solutions based on different techniques (machine learning technique, entropy technique)

Finally, we compare the results of this experimental scenario (overhead, accuracy and false alarms) to the results of all previous works presented in table 2.4 in chapter two. The comparison of overhead and accuracy produced by our scheme to the overhead and accuracy produced by previous works is provided in table 5.3 and table 5.4. The false alarms produced by the scheme are also provided. The results in table 5.3 are compared to Machine Learning solutions while the results in table 5.4 are compared to Entropy solutions. The results in both tables are meant to show the performance enhancement achieved by our scheme compared to previous works.

The results in both tables are meant to show the enhancement achieved by our scheme in terms of decreasing the overhead, increasing the accuracy and producing low false alarms compared to previous works.

Table 5.3: Comparison of results of the elapsed-time based scheme to various machine learning solutions

	Solution developer	**Parameters**	**Accuracy**	**Overhead**	**False Alarms**
Machine Learning based solutions	Braga	Average packets per flow, average bytes per flow, average duration per flow, percentage of pair flows, growth of single flow, and growth of single ports.	99.11%	Very high	0.46%
	Dillon	Number of packets, number of bytes, packet symmetry.	Unknown	Very high	Unknown
	Dotcenko	Connection initiations packets and response packets. Additionally input parameters for the system may contain statistical data from switches are	95%	Measured on NOX.	1.2%

		required such as data speed of selected flows and ports and minimum and maximum number of packets per one IP.			
	Tang	duration, protocol_type, src_bytes, dst_bytes, count and srv_count.	75.75%	Very high	19.51%
	Niyaz	Headers extracted from TCP (Src IP, Window, Dst IP, SYN, Src Port, ACK, Dst Port, URG, Protocol, FIN, Data Size RST, TTL, PUSH) From UDP (Src IP, Dst IP, Src Port, Dst Port, Protocol, Data Size , TTL) And from ICMP (Src IP, Dst IP, ICMP Type, ICMP Code, Protocol, Data Size, and TTL) and a large number of features extracted from these packets headers.	(99.82% for the 2-class model) (95.65% of for the 8-class model)	Very high	(0.3% for the 2-class model) (0.5% for for the 8-class model)
An elapsed-time based scheme	**Wisam**	Number of packets, number of flows, Dst IP and packet arrival time.	**99.85%**	**29.9%**	**0%**

Table 5.4: Comparison of results of the elapsed-time based scheme to various entropy solutions

	Solution developer	Parameters	Accuracy	Overhead	False Alarms
Entropy based solutions	Mehdi	Parameters used in the four algorithms are connection initiations packets, response packets, protocol type, destination port number, all non-IP packets, all incoming traffic, TCP packets starting after the first 100 bytes and packets to any address/port/protocol combination if more than 16 are received in a minute.	(TRW-CB and Rate Limiting algorithms on the home network achieved 90% and on the ISP 85%) (Maximum Entropy and NETAD algorithms achieved 90%)	Measured on NOX	70%
	Giotis	Src IP, Dst IP, Src Port and Dst Port	100%	Measured on NOX	50%
	Mousavi	Number of packets and Dst IP	96%	78%	1%
	Wang	Dst IP, number of packets that have same Dst IP and number of times Dst IP is repeated.	100%	high	25%
	Kia	Number of packets, Dst IP, total number of Dst IP and number of times Dst IP is repeated.	90%	high	10%
An elapsed-time based scheme	**Wisam**	Number of packets, number of flows, Dst IP and packet arrival time.	**99.85%**	**29.9%**	**0%**

5.2.4 Experiment Four (UDP Flood, Low-rate SYN and Normal traffic Scenario – short term)

In this experiment, the mixture of traffic is the same as the mixture of traffic in experiment three but with a difference in the time of traffic generation. Mixture traffic generated in experiment three was generated in the long term but in this experiment will be generated in the short term. Also, thresholds used in the scheme in experiment three were 1500 packets for the flooding DDoS attacks detection and 10 flows for detecting low rate DDoS attacks but, in this experiment, they will change to 5000 packets for flooding DDoS and 30 for flows for low rate DDoS. DDoS attacks launched in this experiment will be the same as attacks launched in experiment three. Python scripts used to launch the attacks will be the same as scripts used in experiment three. Hosts used to launch the attacks in experiment three will be the same in this experiment. The normal traffic generated in this experiment is the same as experiment three. Hosts used to generate the normal traffic in experiment three will be the same in this experiment.

This experiment is meant to evaluate the scheme performance under a more critical situation in terms of minimizing the time of the attack and maximizing the number of attack packets allowed to reach the controller. This means that the scheme has to prove performance enhancement even if the situations are different and more critical.

On the POX controller terminal, no actions have displayed by the scheme although scripts that generate the normal traffic are currently running. Figure 5.52 shows the terminal at the first eighty seconds of normal traffic generation.

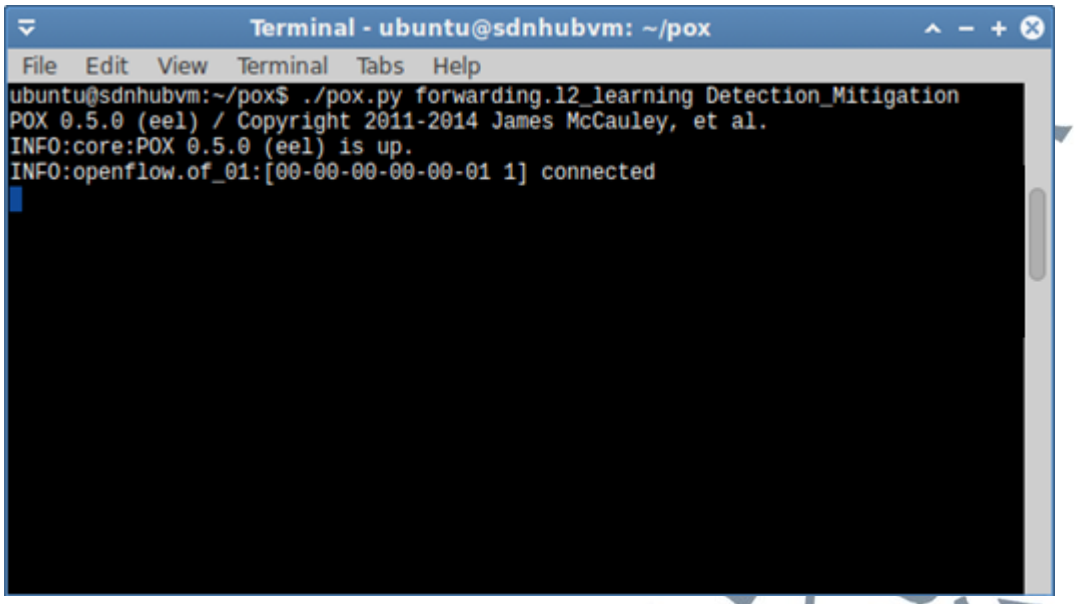


Figure 5.52: POX controller terminal within the first hundred seconds

After hundred-five seconds of generating normal traffic, mixture traffic of low-rate DDoS attack packets and flooding DDoS attack packets have launched from attack hosts along with normal traffic that is still generating. Figures from 5.53 to 5.59 show the detection and mitigation processes of both the low-rate attack packets and the flooding attack packets at different time slots after the attack traffic began generating.

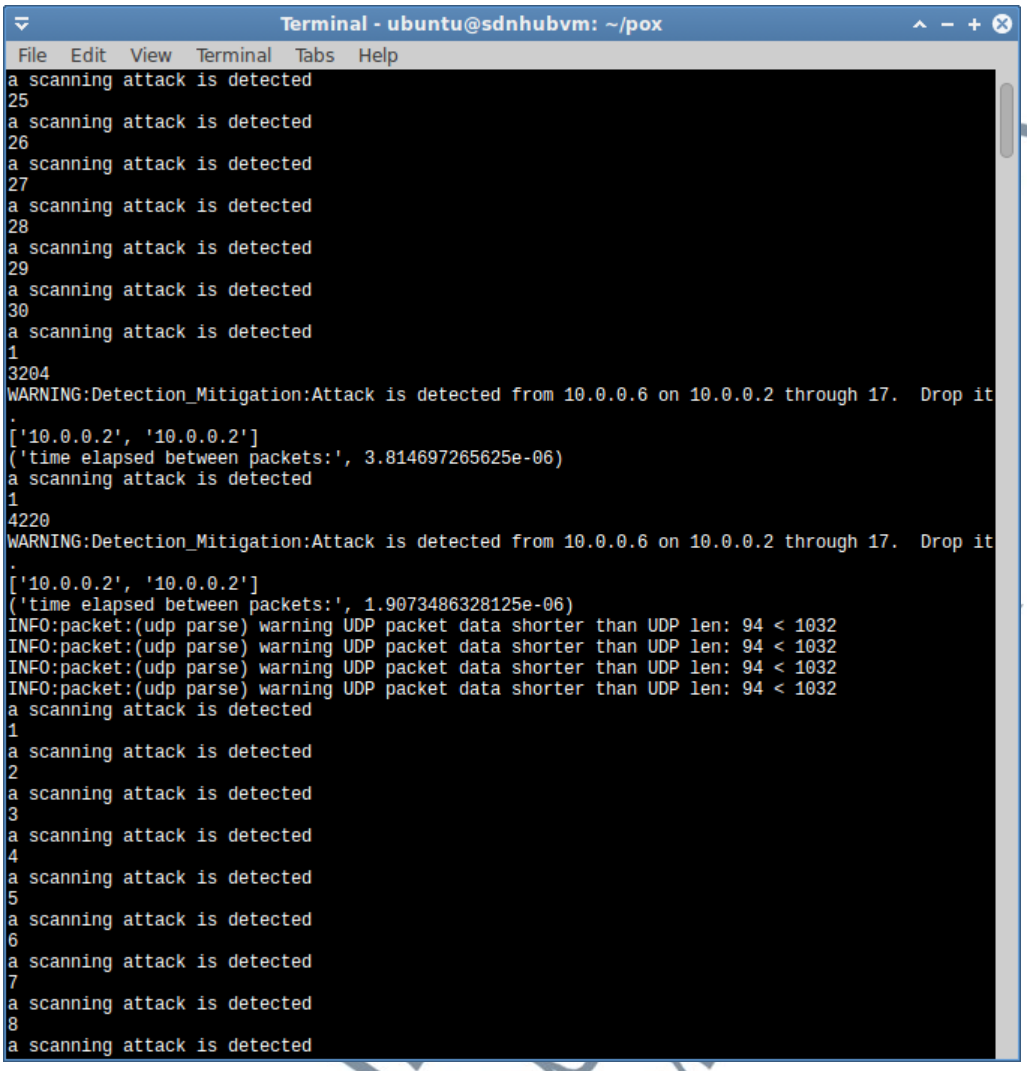


Figure 5.53: Results after hundred-five seconds

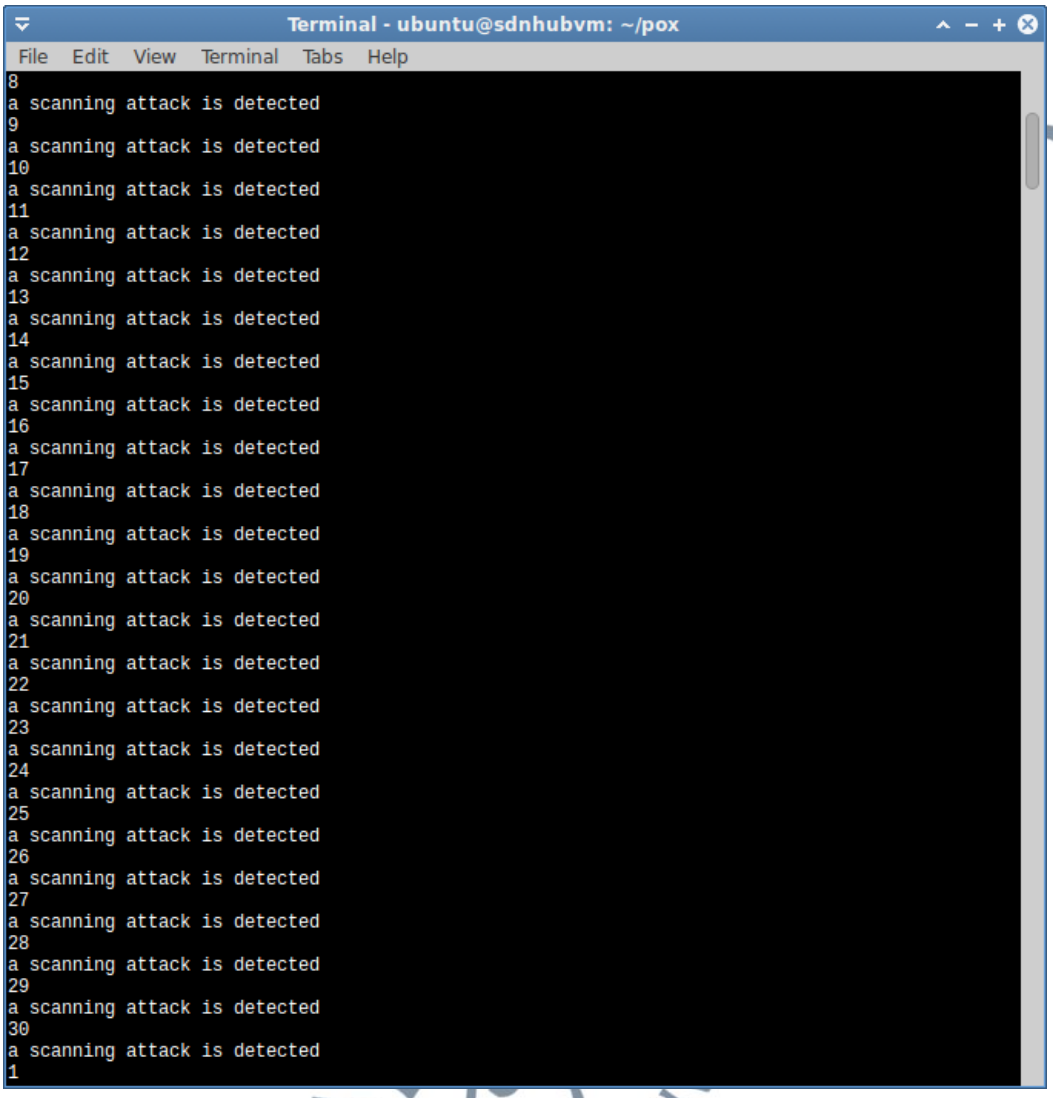


Figure 5.54: Results after two minutes

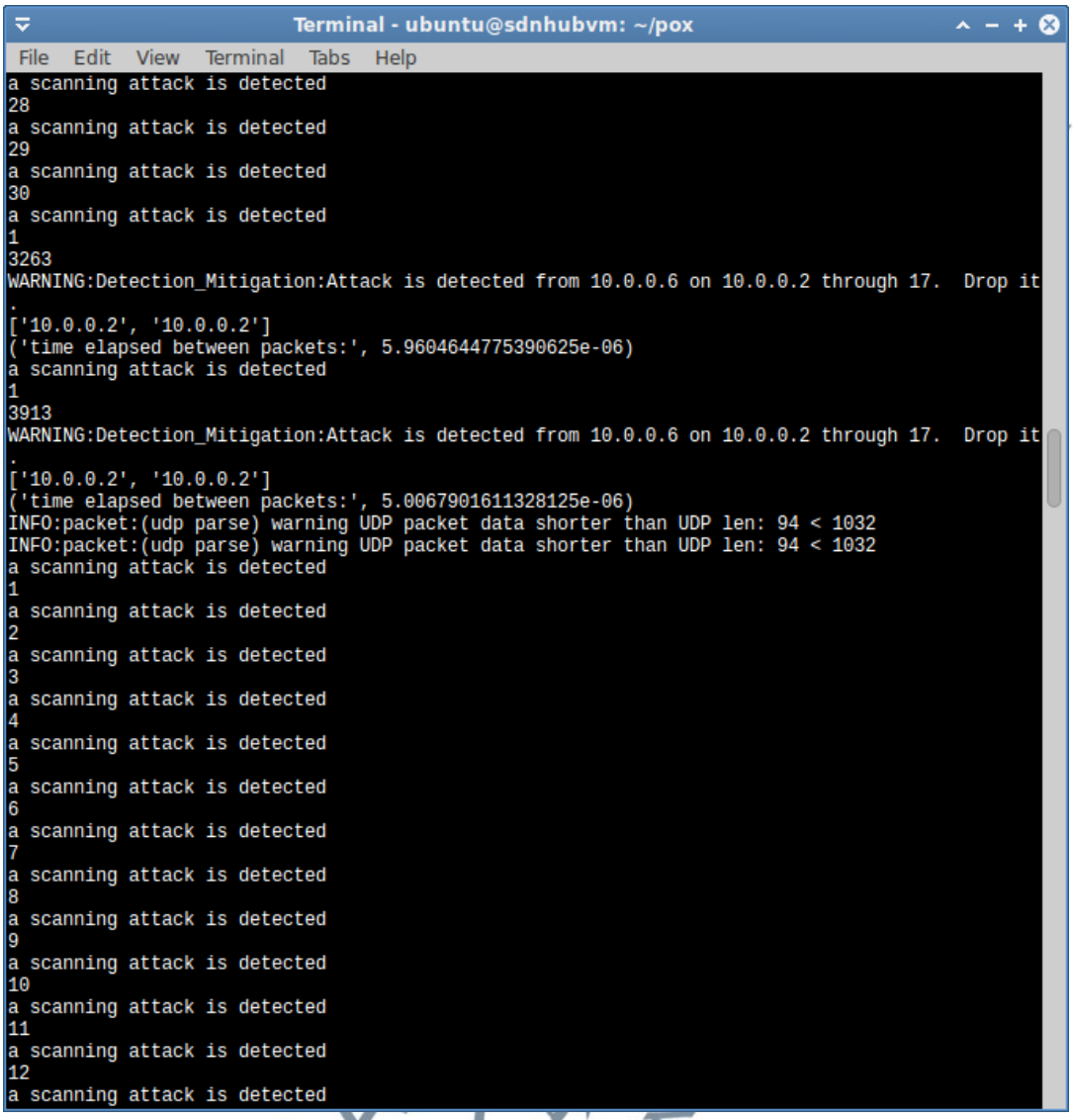


Figure 5.55: Results after three minutes

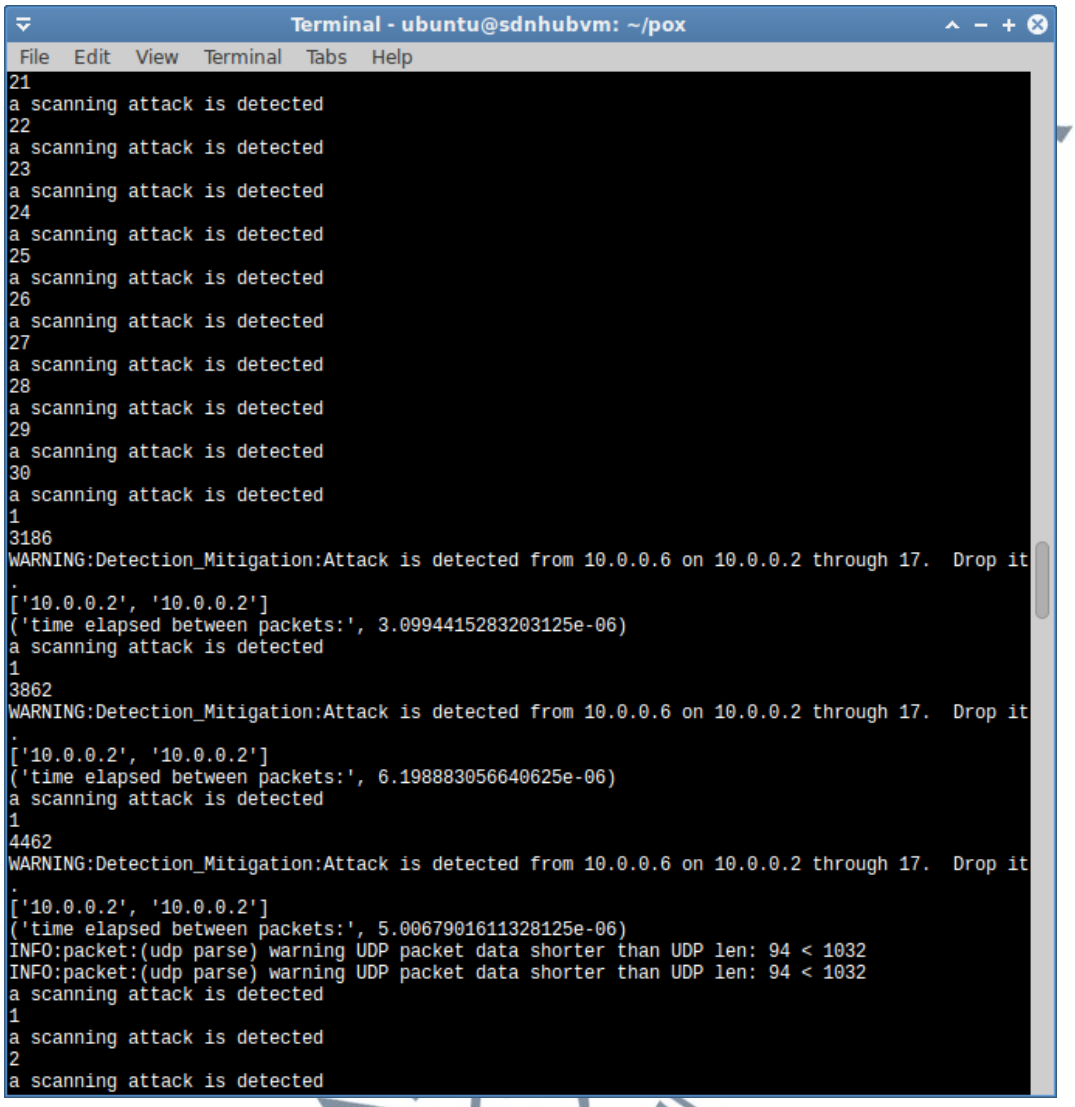


Figure 5.56: Results after five minutes

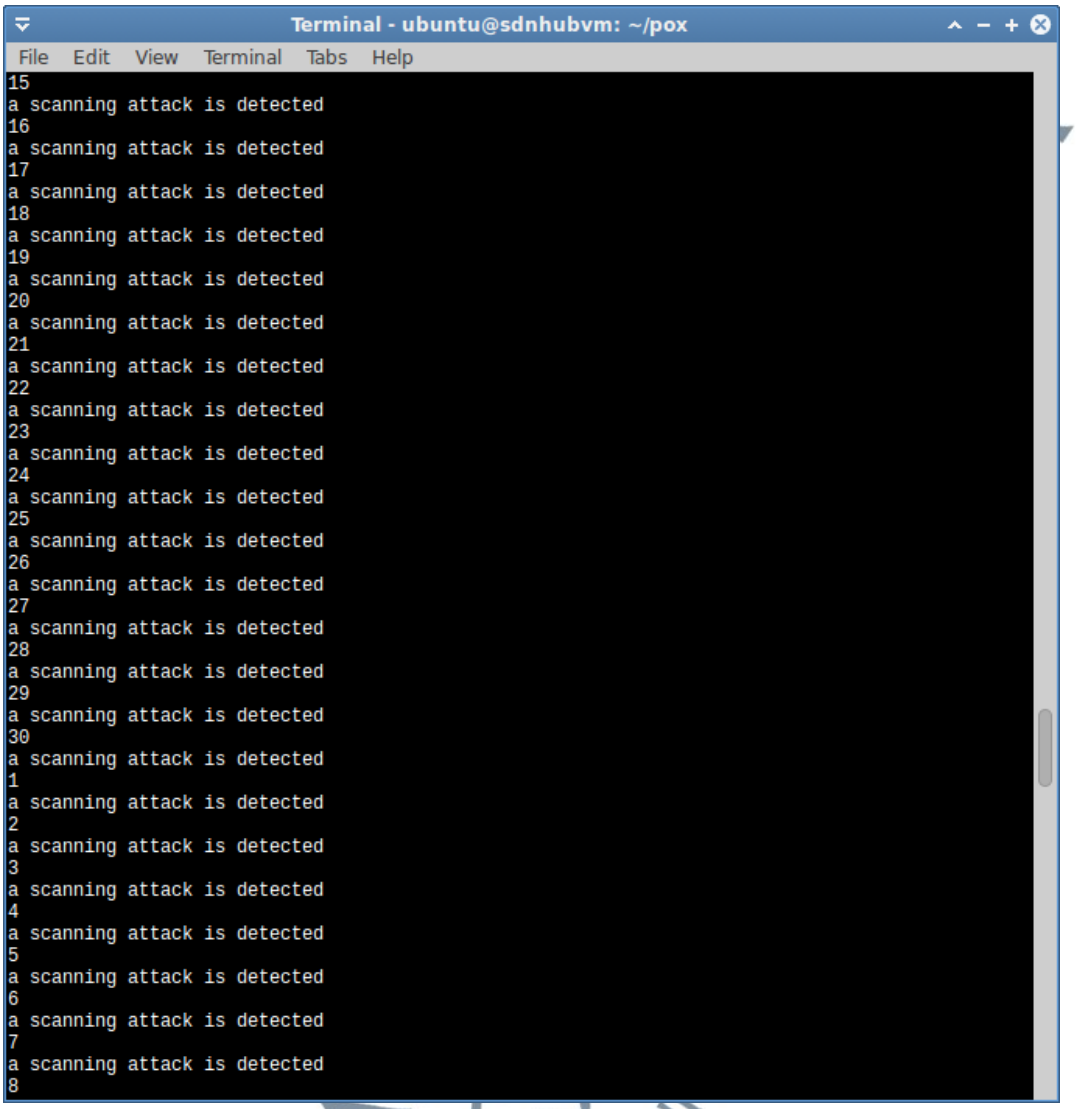


Figure 5.57: Results after seven minutes

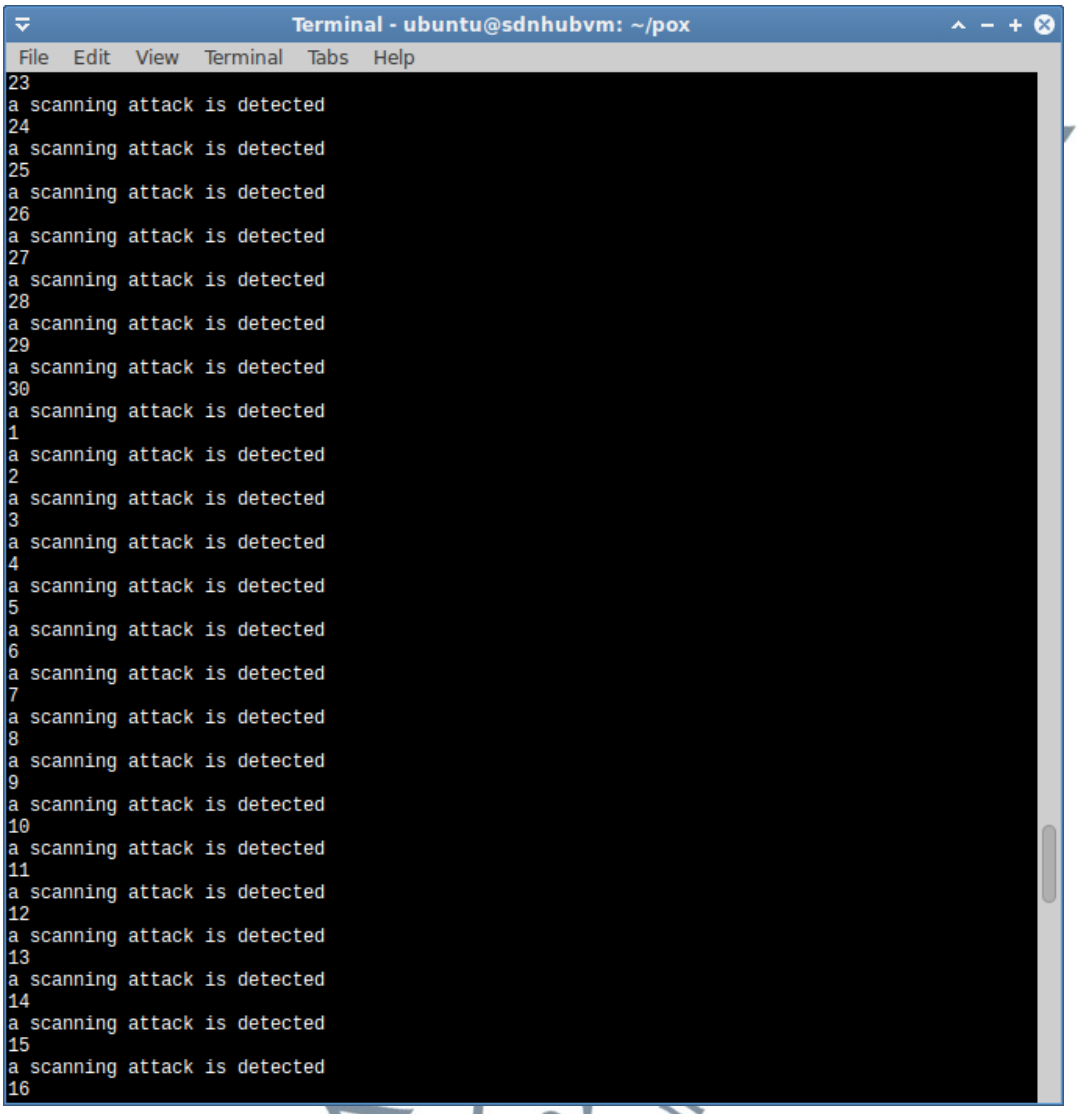


Figure 5.58: Results after eight minutes

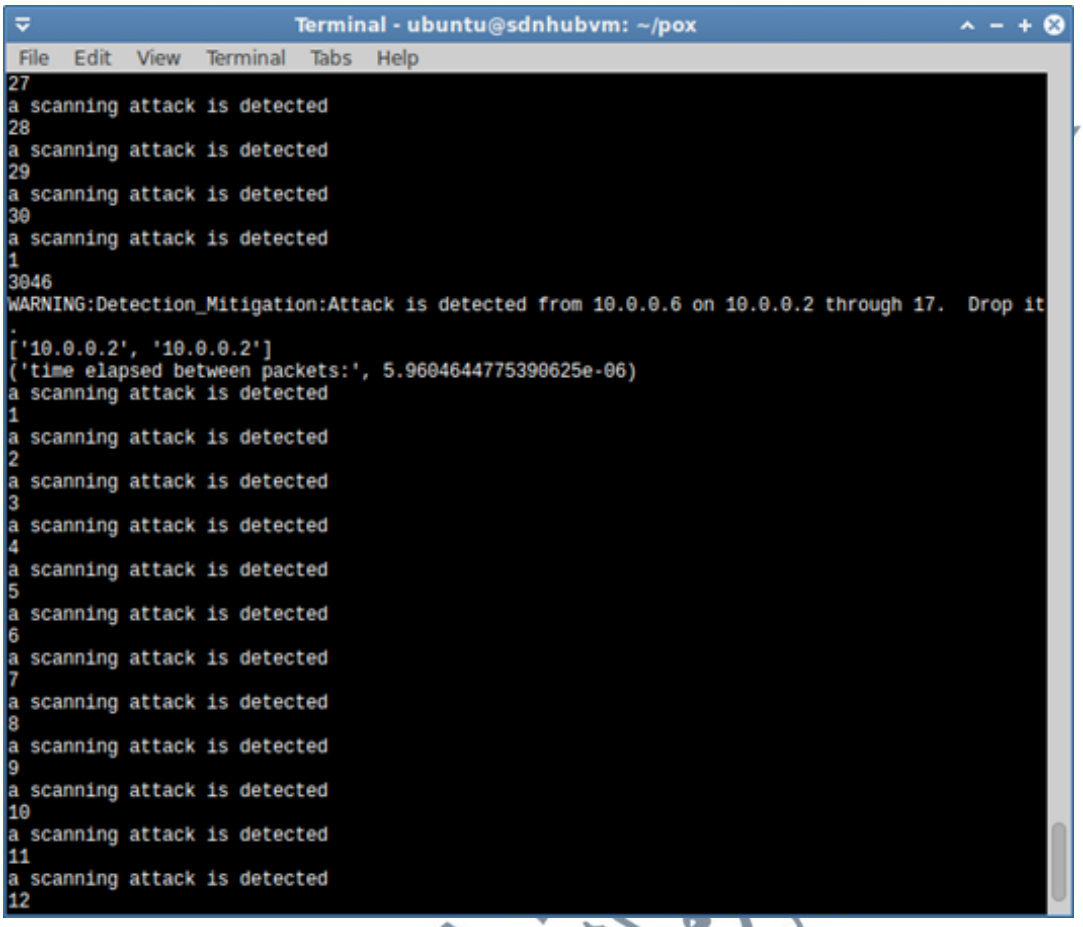
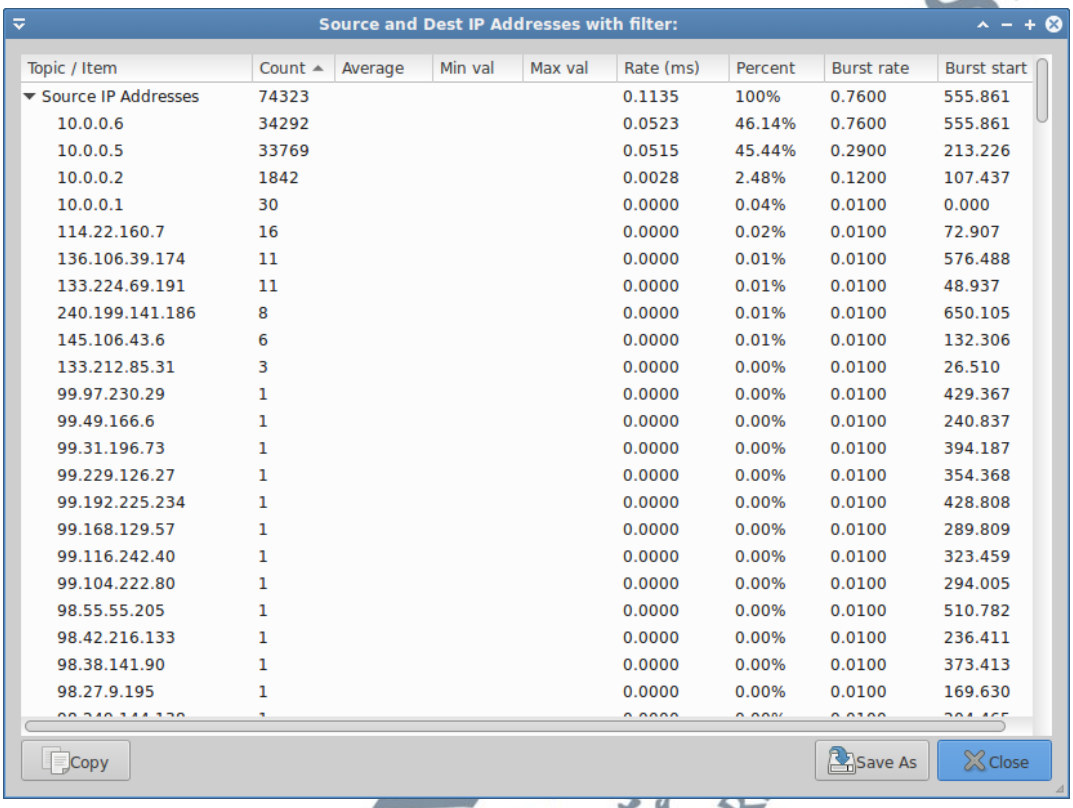
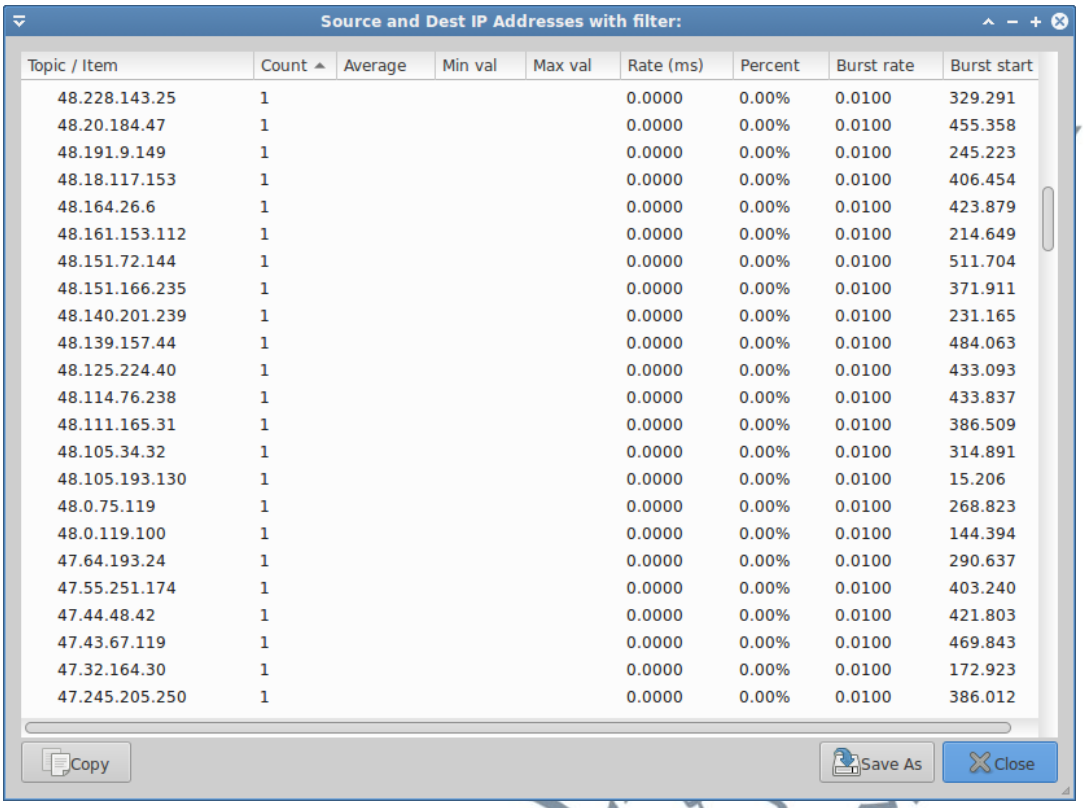


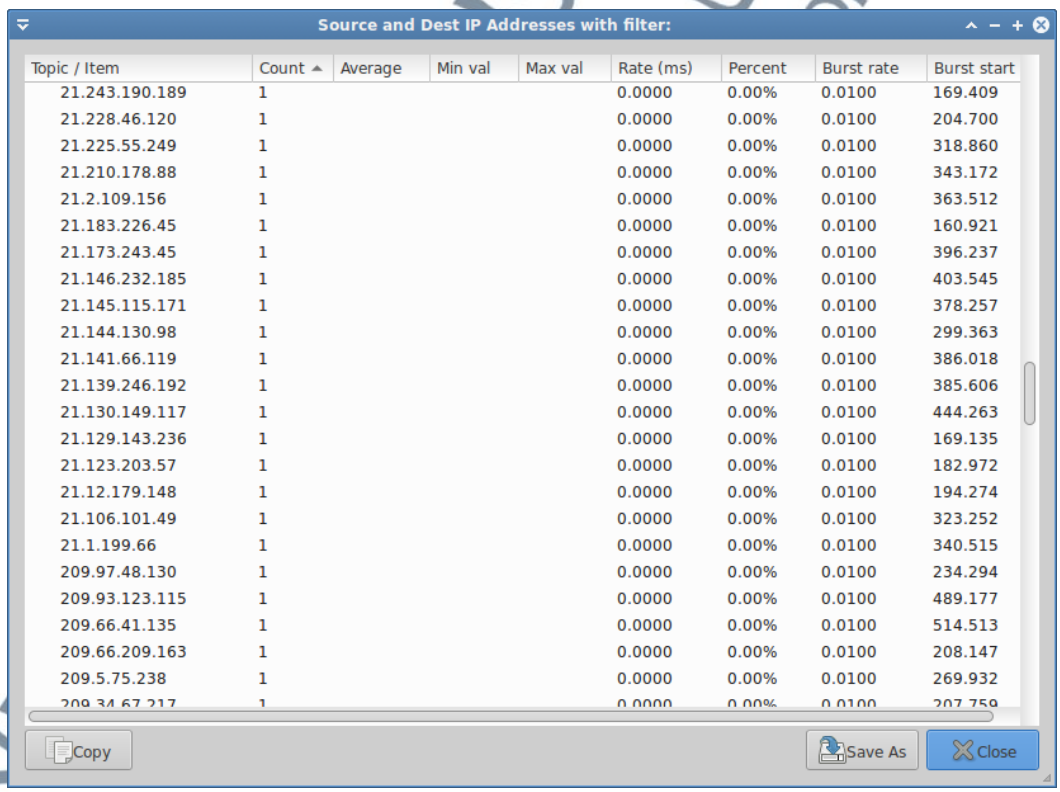
Figure 5.59: Last results

As shown in the Figures from 5.53 to 5.59, the scheme detects and mitigates both the low-rate attack packets and flooding attack packets. Although the normal traffic is generated in the network, the IP addresses of the hosts h1 and h3 (10.0.0.1 and 10.0.0.3) did not appear in the list provided by the scheme during all the time of traffic generation at all. This means that no false alarms have been produced at any second of the traffic generation. The host h4 is the attack machine responsible for launching the low-rate attack packets but, it did not appear on any figure of the mentioned figures. The reason is that host h4 launched the low-rate attack packets separately at a low rate to increase the number of attack flows. For this, we notice lots of attack flows have been detected and removed as shown on the terminals.

The list of the source IP addresses appeared in Figure 5.60 show that the packets that have sent in the network have generated from different spoofed source IP addresses.




Source and Dest IP Addresses with filter:
Topic / Item
Count
Average
Min val
Max val
Rate (ms)
Percent
Burst rate
Burst start
48.228.143.25 1 0.0000 0.00% 0.0100 329.291
48.20.184.47 1 0.0000 0.00% 0.0100 455.358
48.191.9.149 1 0.0000 0.00% 0.0100 245.223
48.18.117.153 1 0.0000 0.00% 0.0100 406.454
48.164.26.6 1 0.0000 0.00% 0.0100 423.879
48.161.153.112 1 0.0000 0.00% 0.0100 214.649
48.151.72.144 1 0.0000 0.00% 0.0100 511.704
48.151.166.235 1 0.0000 0.00% 0.0100 371.911
48.140.201.239 1 0.0000 0.00% 0.0100 231.165
48.139.157.44 1 0.0000 0.00% 0.0100 484.063
48.125.224.40 1 0.0000 0.00% 0.0100 433.093
48.114.76.238 1 0.0000 0.00% 0.0100 433.837
48.111.165.31 1 0.0000 0.00% 0.0100 386.509
48.105.34.32 1 0.0000 0.00% 0.0100 314.891
48.105.193.130 1 0.0000 0.00% 0.0100 15.206
48.0.75.119 1 0.0000 0.00% 0.0100 268.823
48.0.119.100 1 0.0000 0.00% 0.0100 144.394
47.64.193.24 1 0.0000 0.00% 0.0100 290.637
47.55.251.174 1 0.0000 0.00% 0.0100 403.240
47.44.48.42 1 0.0000 0.00% 0.0100 421.803
47.43.67.119 1 0.0000 0.00% 0.0100 469.843
47.32.164.30 1 0.0000 0.00% 0.0100 172.923
47.245.205.250 1 0.0000 0.00% 0.0100 386.012
Copy
Save As
Close


Source and Dest IP Addresses with filter:
Topic / Item
Count
Average
Min val
Max val
Rate (ms)
Percent
Burst rate
Burst start
21.243.190.189 1 0.0000 0.00% 0.0100 169.409
21.228.46.120 1 0.0000 0.00% 0.0100 204.700
21.225.55.249 1 0.0000 0.00% 0.0100 318.860
21.210.178.88 1 0.0000 0.00% 0.0100 343.172
21.2.109.156 1 0.0000 0.00% 0.0100 363.512
21.183.226.45 1 0.0000 0.00% 0.0100 160.921
21.173.243.45 1 0.0000 0.00% 0.0100 396.237
21.146.232.185 1 0.0000 0.00% 0.0100 403.545
21.145.115.171 1 0.0000 0.00% 0.0100 378.257
21.144.130.98 1 0.0000 0.00% 0.0100 299.363
21.141.66.119 1 0.0000 0.00% 0.0100 386.018
21.139.246.192 1 0.0000 0.00% 0.0100 385.606
21.130.149.117 1 0.0000 0.00% 0.0100 444.263
21.129.143.236 1 0.0000 0.00% 0.0100 169.135
21.123.203.57 1 0.0000 0.00% 0.0100 182.972
21.12.179.148 1 0.0000 0.00% 0.0100 194.274
21.106.101.49 1 0.0000 0.00% 0.0100 323.252
21.1.199.66 1 0.0000 0.00% 0.0100 340.515
209.97.48.130 1 0.0000 0.00% 0.0100 234.294
209.93.123.115 1 0.0000 0.00% 0.0100 489.177
209.66.41.135 1 0.0000 0.00% 0.0100 514.513
209.66.209.163 1 0.0000 0.00% 0.0100 208.147
209.5.75.238 1 0.0000 0.00% 0.0100 269.932
209.34.67.217 1 0.0000 0.00% 0.0100 207.759
Copy
Save As
Close

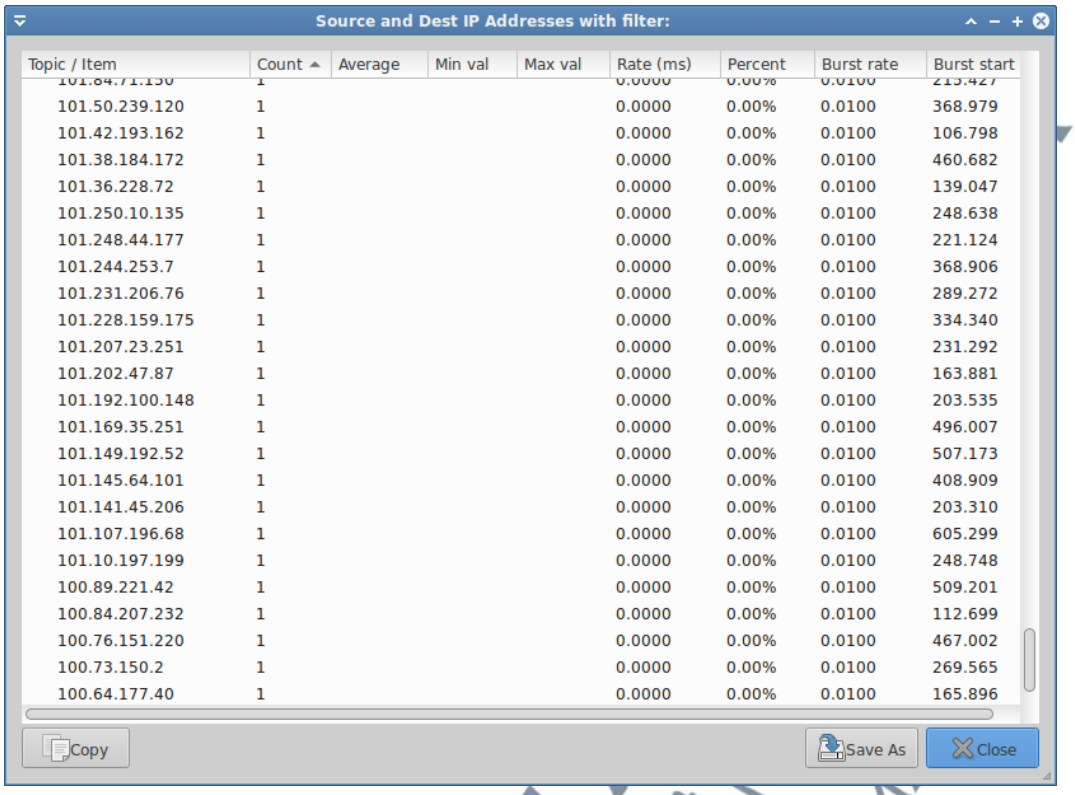
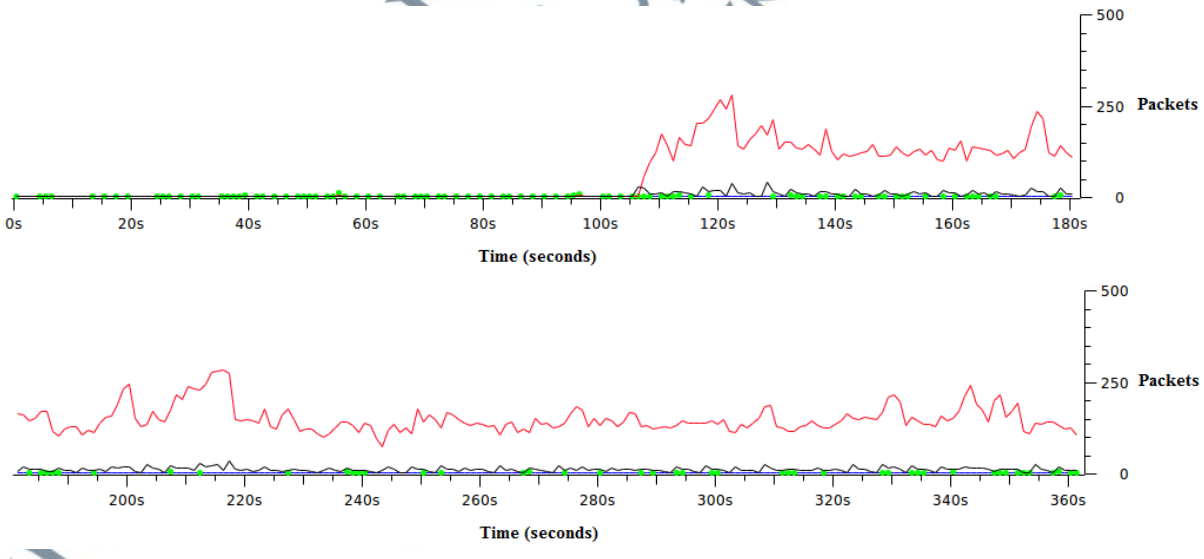


Figure 5.60: List of the source IP addresses

The two source IP addresses (10.0.0.6 and 10.0.0.5) that sent a large number of packets were the IP addresses of hosts that generated the flooding attack traffic while the next two source IP addresses in the list (10.0.0.2 and 10.0.0.1), the IP addresses that established normal TCP handshake. It is important to mention that IP address 10.0.0.2 is the victim's IP address but it appears in the list as a source IP address because it is one of the two sides in the TCP connection establishment.

The IP address 10.0.0.1 is the IP address generates normal TCP traffic and the next source IP addresses until the IP address 133.212.85.31 are the IP addresses generate normal UDP traffic. The rest of the spoofed source IP addresses that appeared in the list are the IP addresses that generate the low-rate SYN attack packets. These packets sent separately to increase the number of attack flows.

In all graphs appeared in Figure 5.61, the red line represents the UDP flood attack and the black line represents the low-rate SYN attack. In these graphs, we also used another two different colours to represent the normal traffic. The blue line represents the normal TCP traffic and the green represents the normal UDP traffic. In all graphs, we can notice that no sharp increase in the green dots or the blue line during the time of traffic generation due to generating the normal packets at an average of not greater than ten packets per second from true source IP addresses. We can also notice that sharp increase presented by the graphs clearly appeared in the red lines more than the black lines due to the basic difference between the flooding attacks and the low-rate attacks in the attack rate. The flooding attacks send the attack packets in a large number of packets at a high average of rate whereas the low-rate attacks send the attack packets at a low average of rate. In addition, the low-rate attack used in the experimental scenarios is designed to send the attack packets separately to increase the number of flows.



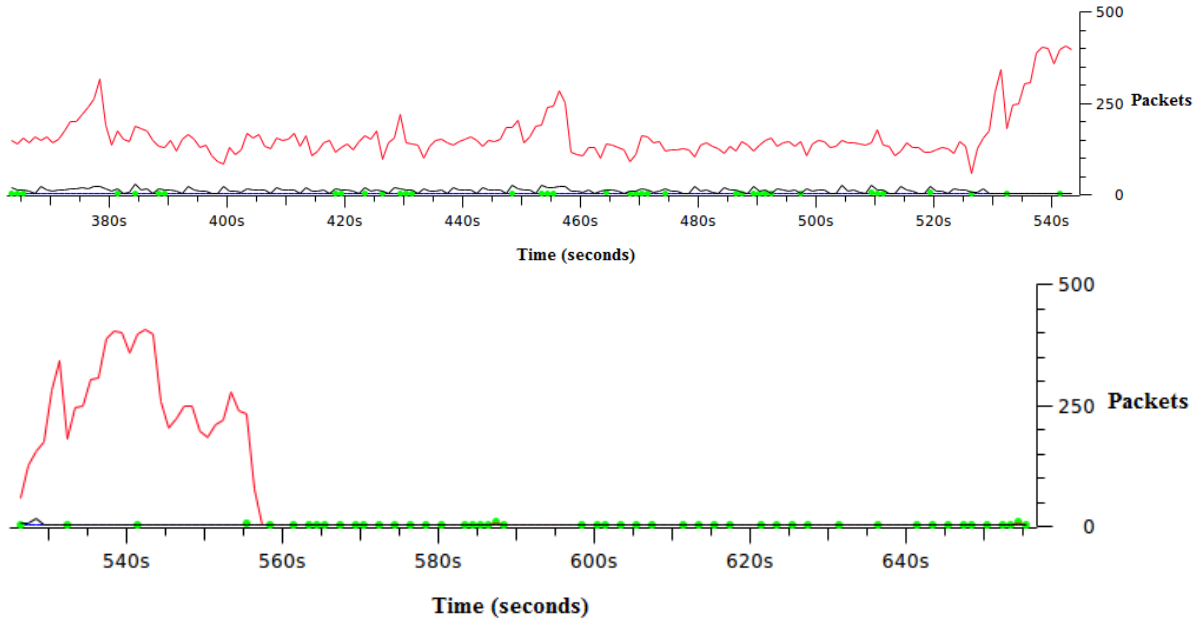
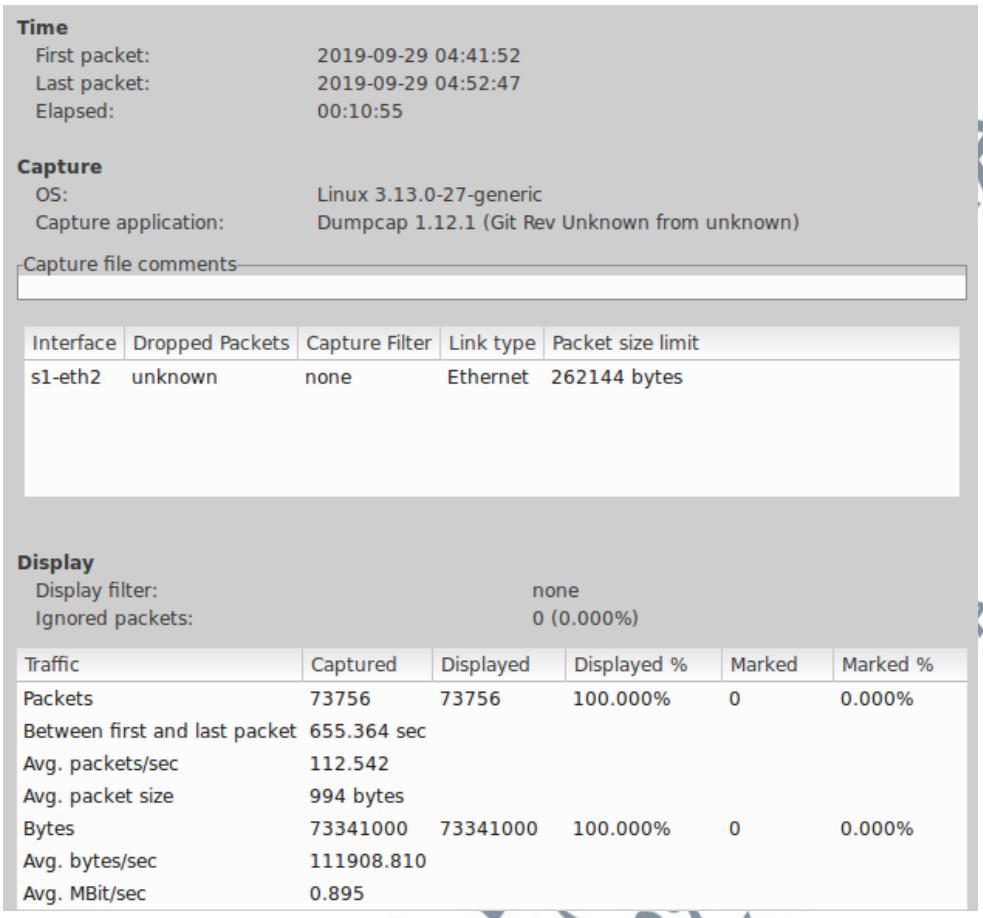


Figure 5.61: The performance under the mixture traffic (short term)

We can observe from all graphs appeared in Figure 5.61 that the scheme succeeded to protect the OpenFlow switch and the SDN controller from different types of DDoS attacks each of which follows a different strategy of attack during the time of traffic generation. Keeping the SDN controller and the OpenFlow switch running smoothly under different attack characteristics proves that the scheme works efficiently under different attack situations. The scheme stops taking actions against incoming packets once the DDoS scripts stop.

Figure 5.62 shows that the mixture of traffic generation has last for ten minutes and fifty-five seconds. Also, the summary provides details about packets that have sent in the mixture traffic scenario.



Time

First packet:	2019-09-29 04:41:52
Last packet:	2019-09-29 04:52:47
Elapsed:	00:10:55

Capture

OS:	Linux 3.13.0-27-generic
Capture application:	Dumpcap 1.12.1 (Git Rev Unknown from unknown)

Capture file comments

Interface	Dropped Packets	Capture Filter	Link type	Packet size limit
s1-eth2	unknown	none	Ethernet	262144 bytes

Display

Display filter:	none
Ignored packets:	0 (0.000%)

Traffic	Captured	Displayed	Displayed %	Marked	Marked %
Packets	73756	73756	100.000%	0	0.000%
Between first and last packet	655.364 sec				
Avg. packets/sec	112.542				
Avg. packet size	994 bytes				
Bytes	73341000	73341000	100.000%	0	0.000%
Avg. bytes/sec	111908.810				
Avg. MBit/sec	0.895				

Figure 5.62: Summary of the mixture traffic (short term)

5.2.4.1 Scheme Performance Evaluation (UDP flood, Low-rate SYN and Normal traffic scenario – short term)

5.2.4.1.1 Overhead (CPU Usage)

To examine the effect of the proposed scheme on the controller's CPU, we left the simulation running with all the running processes on a Linux PC (running Ubuntu 14.04).

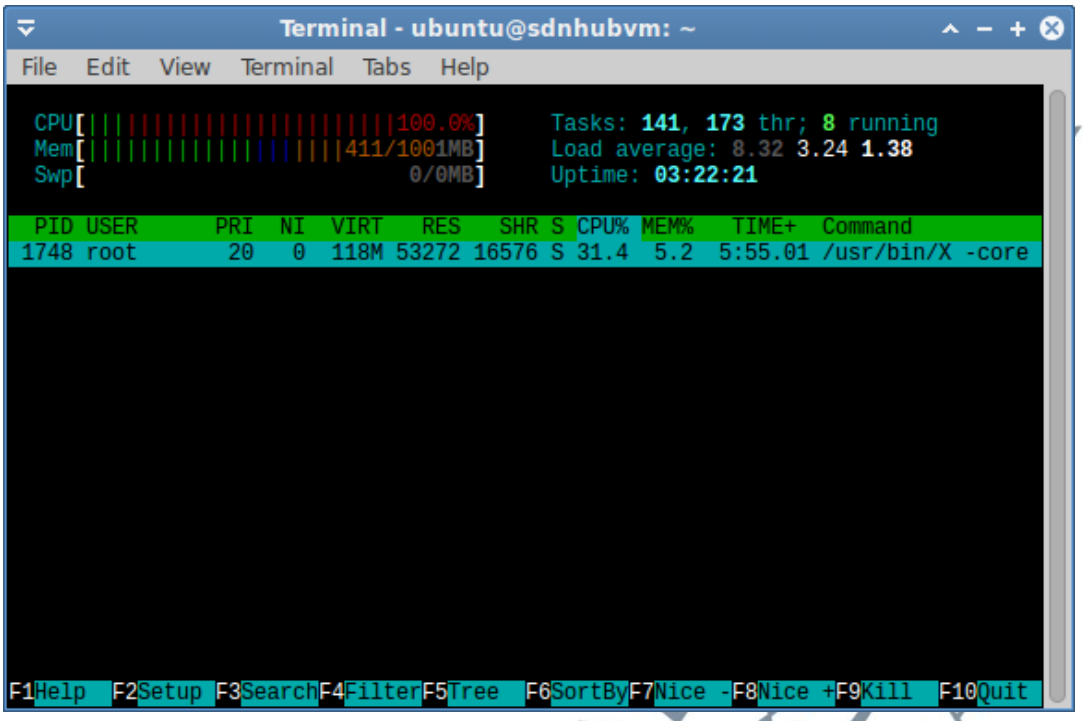


Figure 5.63: CPU usage under the mixture traffic

We left the system monitoring application of Linux (htop) running during all the time of the experiment. Figure 5.63 shows the CPU usage while the proposed scheme is in use at the time of mixture traffic (UDP flood, low-rate SYN and Normal traffic) generation. The overhead measured is only 31.4%. The low overhead measured shows the scheme has a positive reflection on the CPU usage of the SDN controller even if the attack characteristics have changed.

5.2.4.1.2 Accuracy

The accuracy achieved by the proposed scheme under the experimental scenario (UDP flood, low-rate SYN, and Normal traffic) is 98.13%. The accuracy is calculated as in chapter three, section 3.5.3.2. By considering that the last number of attack packets appeared in the last terminal was dropped by the scheme to be less than the threshold, the accuracy achieved in this scenario will be 100%. This result

shows that this scheme can, easily, detect both the attack packets and attack flows when it is destined to a victim in the network.

Since this scheme has been tested under this experimental scenario many times, no false positives or false negatives have appeared on the POX terminal at any time of the traffic generation. So, false alarms produced by the scheme under this experimental scenario are 0.

5.3 DISCUSSION ON THE RESULTS

In order to validate our results, we must compare it to solutions have implemented and tested in the SDN. Results of each mixture traffic experimental scenario are compared to previous works of machine learning and entropy solutions in separate tables shown in sections 5.2.1.4, 5.2.2.4 and, 5.2.3.1.

Compared to machine learning based solutions that achieve high accuracy but with high resource usage, the proposed scheme produces low measured CPU usage in the SDN with achieving a high accuracy percentage at the same time. The proposed scheme achieved 99.27% of accuracy with 34.3% of CPU usage under the mixture of UDP flood and normal traffic scenario, 99.47% of accuracy with 2.5% of CPU usage under low-rate SYN and normal traffic scenario and 99.85% of accuracy with 29.9% of CPU usage under the mixture of UDP flood and low-rate SYN and normal traffic scenario. Although machine learning based solutions give high accuracy with low false alarms rates, they produce high levels of overhead when they operate as presented in the comparisons tables.

The overhead is high in the machine learning based solutions due to, extracting lots of features from the network traffic and training the sets that contain

these features for a long time to learn the traffic behaviour which limits the controller's performance. The reason for that is the complexity of the machine learning technique (Bawany et al., 2017) compared to the proposed scheme and the reflection of the selected parameters for attack detection on the performance (Cvitić et al., 2017). While machine learning techniques use several calculations with large matrices to learn the network behaviour at a very high cost of resource usage, the proposed scheme used in this research does all the functionalities without the need for any of the complicated measures used in them.

Also, the results of the proposed scheme have compared to the results of entropy-based solutions. In this research, the proposed scheme achieved 99.27% of accuracy under the mixture of UDP flood and normal traffic scenario, 99.47% of accuracy under SYN flood and normal traffic scenario and 99.85% of accuracy under the mixture of UDP flood and SYN flood, and normal traffic scenario. Compared to up to 70% false alarms rate produced by entropy-based solutions appeared in table 4.1 chapter two, the proposed scheme produced 0% false alarms. Although, entropy-based solutions showed high accuracy in detecting DDoS; they produced high false alarms rates compared to the machine learning based solutions (XU et al., 2017). However, entropy based solutions produce lower overhead than machine learning solutions, the overhead still considered high.

In terms of parameters used for DDoS detection in SDN, the parameters used in the proposed scheme were less and different than the parameters used by the existing solutions. Parameters numbers and types are one of the reasons for positive or negative effects on the detection solution. The number of used parameters must be as low as possible and it is necessary to use parameters that have the greatest impact

when detecting network traffic anomaly (Bhattacharyya & Kalita, 2016). According to (Cvitić et al., 2017), the selection of parameters to be used is a key component for achieving an effective system of DDoS attacks detection.

In contrast to both machine learning based solutions and entropy-based solutions, the scheme proposed in this research produced a very low overhead and very high accuracy with no false alarms.

According to (Behal & Kumar, 2016), there is a failure of existing DDoS detection and defense solutions to detect the changing in the volume of attack packets as attack moves from the high volume of attack to the low volume and vice versa. As appeared in the first and third experiments, the proposed scheme has detected the high volume of attack packets as well as the low volume that was generated at the same time in the same traffic with 99.27% of accuracy and 34.3% of CPU usage in the first experiment and, 99.85% of accuracy and 29.9% of CPU usage in the third experiment.

According to (Behal & Kumar, 2016; Sahoo et al., 2018), the low-rate attacks are difficult to detect and few works have considered the low-rate DDoS attacks detection in SDN. The proposed scheme detected the low-rate SYN attack packets that sent to create a vast number of flows with 99.47% of accuracy and 2.5% of CPU usage as appeared in the experimental scenario (Low-rate SYN attack and normal traffic).

The results obtained in the third experiment have been compared to the results of the fourth experiment to show the correctness of the results and robustness of the scheme in a confrontation of different attack scenarios.

Table 5.5: Comparison of results of the elapsed-time based scheme obtained in experiment three and four

Attack traffic scenario	Thresholds values	Overhead	Accuracy	False alarms
UDP Flood, Low-rate SYN and Normal traffic Scenario **(long term)**	1500 for packets and 10 for flows	29.9%	99.85%	0%
UDP Flood, Low-rate SYN and Normal traffic Scenario **(short term)**	5000 for packets and 30 for flows	31.4%	98.13%	0%

The comparison shows that results obtained by the scheme are almost the same. The reason is the selection of parameters used in the scheme. As appear in table 5.5, the difference in the results is insignificant. According to (Cvitić et al., 2017), as the selection of parameters has an impact on the performance, it is also one of the reasons to provide convergent performance. The results in table 5.5 confirm that the scheme produced low overhead and high accuracy with low false alarms no matter how the confrontation scenarios change.

5.4 SUMMARY

In this chapter, we have conducted four extensive experiments for evaluating the performance and confirming the validity of the results. Firstly, we have conducted an experiment had three test cases to evaluate the proposed scheme under UDP flood DDoS attack. The first test case was used to test the detection function of the scheme under normal traffic generation (normal UDP packets) to see if it produced any false positives. The second test case was used to test the detection and mitigation functions of the scheme under attack traffic generation to make sure the detection function

produced no false negatives and, the mitigation function dropped the attack packets successfully. The third test case was aimed to evaluate the detection and mitigation functions under a mixture of traffic generation where the attack packets and legitimate packets are mixed and, generated in the same traffic simultaneously.

Secondly, we conducted an experiment had three test cases to evaluate the scheme under low-rate SYN DDoS attack. This experiment aimed to evaluate the scheme under a low-rate attack type that increases the number of packet-in messages. This attack sends the packets at a low rate to look normal and sends them separately as single packets to increase the number of flows instead of packets. The first test case was used to test the detection function of the scheme under normal traffic generation (normal TCP handshakes) to see if it produced any false positives. The second test case was used to test the detection and mitigation functions of the scheme under attack traffic generation (low-rate SYN packets) to make sure the detection function produced no false negatives and, the mitigation function removed the attack flows successfully. The third test case was aimed to evaluate the detection and mitigation functions under a mixture of traffic generation where the attack packets (low-rate SYN packets) and legitimate packets (normal TCP packets) are mixed and, generated in the same traffic simultaneously.

Thirdly, we conducted an experiment to evaluate the scheme under a mixture of a low-rate attack and, flooding attack in the same traffic generation. This experiment aimed to evaluate the detection and mitigation functions of the scheme under attacks that generate traffic at high and low traffic rates to put all SDN components under a dangerous situation at the same time.

Finally, we conducted an experiment to confirm the correctness of the results and robustness of the scheme under different attack traffic characteristics.

The evaluation and validation of the presented scheme were conducted using Mininet emulator. For each experiment, we compared the overhead, accuracy and, false alarms to all previous works presented in chapter two, table 4.1. As a result, the new scheme shows better performance compared to the existing solutions and convergent results under different attack situation compared to the proposed scheme itself.