Ravi Nayyar
3/21/2022
Security in Networked Systems

Lab 3: Dive into SDN Security, Part II: The Murky Depths

Section 1: Attacking the SDN Framework

This section of the lab calls for executing the three attacks created in lab 1: a direct ICMP flooding attack, an obfuscated ICMP flooding attack, and a TCP syn flood. Prior to testing the attacks, the SDN topology (Topology 1) needed to be initialized by running the “ovs-init.sh” script on switches sw1 - sw4 and the “init_learn_rules.py” script on the controller server. Lab 1 first had us manually construct ICMP and TCP packets before moving on to attack creation. Thus, the initial task0a and task0c scripts were run on topology 1 to ensure that the manually created packets would still be viable. It should be noted, that while the attacks are being tested on topology 1, the firewall rules created in lab 3 part 1 will be turned off. However, in section 2, keeping in accordance with the firewall rules, all attacks will be started (or look like they started) from the 10.0.0.0 subnet. Figures 1 and 2 show the terminal output and Wireshark captures proving valid packet creation.

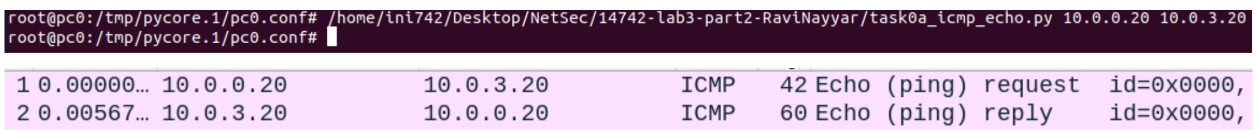


Figure 1: Created ICMP packet from pc0 to pc6

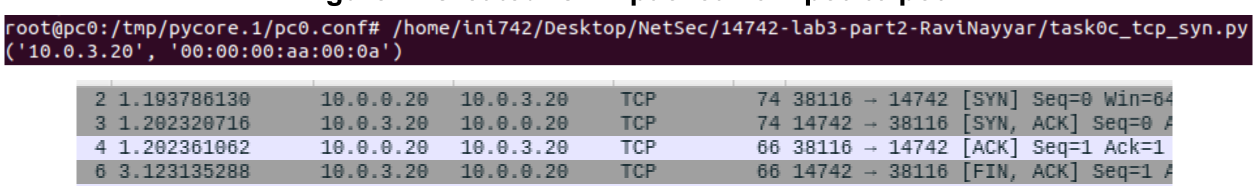


Figure 2: Created TCP packet from pc0 to pc6

Attack 1: Flooding Attacks with ICMP

With manual ICMP packets able to be sent, a standard ICMP flooding attack could then be started. In lab 1, the attack’s parameters were that the hacker’s host would send an ICMP echo request message to the target at a rate of around 10 messages/second, and with this implementation, the attacker decided not to spoof its own identifying features. Figure 4 shows 10 ICMP request/reply pairs being sent to and from pc0 (10.0.0.20) and pc6 (10.0.3.20) in the span of 1 second. The ICMP smurf attack, while sending packets out at the same rate, sends request messages to all available devices except for the attack target - but it replaces all the relevant source identifiers with the target device’s identifiers - i.e. pc0 will send ICMP request messages on behalf of the target pc6 to every other node in topology 1. The smurf attack amplifies the standard ICMP request flood because every host the attacker pings will send an ICMP reply back to the target. Figure 5 shows the Wireshark capture of the smurf attack - the only requests are shown to be coming from pc6 (10.0.3.20) and the replies are originating from every host and are also all destined for pc6. To run the standard ICMP attack, open the pc0 host and enter the full path to the “task2_icmp_flooding.py” script. A single command-line argument needs to be set. If the variable is set to “False”, then a normal ICMP flooding attack will start, but if “True” is set, then the smurf attack would start (Figure 5).

```
root@pc0:/tmp/pycore.1/pc0.conf# /home/ini742/Desktop/NetSec/14742-lab3-part2-RaviNayyar/task2_icmp_flooding.py False
Starting Standard Attack
```

```
root@pc0:/tmp/pycore.1/pc0.conf# /home/ini742/Desktop/NetSec/14742-lab3-part2-RaviNayyar/task2_icmp_flooding.py True
Starting Smurf Attack
```

Figure 3: Starting both kinds of standard ICMP flooding attacks

1	0.000000…	10.0.0.20	10.0.3.20	ICMP	60	Echo (ping) request id=0x0000, seq=1/256,…
2	0.000034…	10.0.3.20	10.0.0.20	ICMP	42	Echo (ping) reply id=0x0000, seq=1/256,…
3	0.098096…	10.0.0.20	10.0.3.20	ICMP	60	Echo (ping) request id=0x0000, seq=1/256,…
4	0.098119…	10.0.3.20	10.0.0.20	ICMP	42	Echo (ping) reply id=0x0000, seq=1/256,…
5	0.197745…	10.0.0.20	10.0.3.20	ICMP	60	Echo (ping) request id=0x0000, seq=1/256,…
6	0.197766…	10.0.3.20	10.0.0.20	ICMP	42	Echo (ping) reply id=0x0000, seq=1/256,…
7	0.298401…	10.0.0.20	10.0.3.20	ICMP	60	Echo (ping) request id=0x0000, seq=1/256,…
8	0.298432…	10.0.3.20	10.0.0.20	ICMP	42	Echo (ping) reply id=0x0000, seq=1/256,…
9	0.398632…	10.0.0.20	10.0.3.20	ICMP	60	Echo (ping) request id=0x0000, seq=1/256,…
10	0.398656…	10.0.3.20	10.0.0.20	ICMP	42	Echo (ping) reply id=0x0000, seq=1/256,…
11	0.498726…	10.0.0.20	10.0.3.20	ICMP	60	Echo (ping) request id=0x0000, seq=1/256,…
12	0.498757…	10.0.3.20	10.0.0.20	ICMP	42	Echo (ping) reply id=0x0000, seq=1/256,…
13	0.600513…	10.0.0.20	10.0.3.20	ICMP	60	Echo (ping) request id=0x0000, seq=1/256,…
14	0.600536…	10.0.3.20	10.0.0.20	ICMP	42	Echo (ping) reply id=0x0000, seq=1/256,…
15	0.700889…	10.0.0.20	10.0.3.20	ICMP	60	Echo (ping) request id=0x0000, seq=1/256,…
16	0.700912…	10.0.3.20	10.0.0.20	ICMP	42	Echo (ping) reply id=0x0000, seq=1/256,…
17	0.801439…	10.0.0.20	10.0.3.20	ICMP	60	Echo (ping) request id=0x0000, seq=1/256,…
18	0.801463…	10.0.3.20	10.0.0.20	ICMP	42	Echo (ping) reply id=0x0000, seq=1/256,…
19	0.902188…	10.0.0.20	10.0.3.20	ICMP	60	Echo (ping) request id=0x0000, seq=1/256,…
20	0.902215…	10.0.3.20	10.0.0.20	ICMP	42	Echo (ping) reply id=0x0000, seq=1/256,…
21	1.001958…	10.0.0.20	10.0.3.20	ICMP	60	Echo (ping) request id=0x0000, seq=1/256,…

Figure 3: Standard ICMP flooding attack from pc0 to pc6

1	0.000000…	10.0.3.20	10.0.3.21	ICMP	60	Echo (ping) request id=0x0000, seq=1/256,…
2	0.000023…	10.0.3.21	10.0.3.20	ICMP	42	Echo (ping) reply id=0x0000, seq=1/256,…
3	0.001852…	10.0.2.20	10.0.3.20	ICMP	60	Echo (ping) reply id=0x0000, seq=1/256,…
4	0.002009…	10.0.2.21	10.0.3.20	ICMP	60	Echo (ping) reply id=0x0000, seq=1/256,…
5	0.002077…	10.0.1.21	10.0.3.20	ICMP	60	Echo (ping) reply id=0x0000, seq=1/256,…
6	0.002141…	10.0.1.20	10.0.3.20	ICMP	60	Echo (ping) reply id=0x0000, seq=1/256,…
7	0.097472…	10.0.3.20	10.0.3.21	ICMP	60	Echo (ping) request id=0x0000, seq=1/256,…
8	0.097505…	10.0.3.21	10.0.3.20	ICMP	42	Echo (ping) reply id=0x0000, seq=1/256,…
9	0.098678…	10.0.1.20	10.0.3.20	ICMP	60	Echo (ping) reply id=0x0000, seq=1/256,…
10	0.098755…	10.0.1.21	10.0.3.20	ICMP	60	Echo (ping) reply id=0x0000, seq=1/256,…
11	0.099554…	10.0.2.20	10.0.3.20	ICMP	60	Echo (ping) reply id=0x0000, seq=1/256,…
12	0.099652…	10.0.2.21	10.0.3.20	ICMP	60	Echo (ping) reply id=0x0000, seq=1/256,…
13	0.198512…	10.0.3.20	10.0.3.21	ICMP	60	Echo (ping) request id=0x0000, seq=1/256,…
14	0.198550…	10.0.3.21	10.0.3.20	ICMP	42	Echo (ping) reply id=0x0000, seq=1/256,…
15	0.199631…	10.0.1.20	10.0.3.20	ICMP	60	Echo (ping) reply id=0x0000, seq=1/256,…
16	0.199712…	10.0.1.21	10.0.3.20	ICMP	60	Echo (ping) reply id=0x0000, seq=1/256,…
17	0.200451…	10.0.2.20	10.0.3.20	ICMP	60	Echo (ping) reply id=0x0000, seq=1/256,…
18	0.200526…	10.0.2.21	10.0.3.20	ICMP	60	Echo (ping) reply id=0x0000, seq=1/256,…

Figure 4: Smurf ICMP flooding attack from pc0 to pc6

Attack 2: Smarter ICMP Attacks

The smarter ICMP attack calls for modifying the attacker's MAC address that appears in the ethernet header. Apart from that, however, the attack is identical to the ICMP smurf flooding attack. The methodology for this attack was changed from lab 1; previously, the following three commands were used to programmatically change the actual mac address of the interface.

ip link set dev eth0 down
ip link set dev eth0 address <new mac address>
ip link set dev eth0 up

However, this solution was not compatible with the topology and thus the ethernet header was changed to allow for a randomized mac address. The following code allows for a valid mac address with the last byte randomized. Figure 5 shows the Wireshark capture for the smarter ICMP attack.

src_mac = "000000AA00"+hex(random.randint(50,255))[2:]

1	0.000000…	10.0.3.20	10.0.3.21	ICMP	60	Echo (ping) request	id=0x0000,	seq=1/256,…
2	0.000021…	10.0.3.21	10.0.3.20	ICMP	42	Echo (ping) reply	id=0x0000,	seq=1/256,…
3	0.000092…	10.0.1.20	10.0.3.20	ICMP	60	Echo (ping) reply	id=0x0000,	seq=1/256,…
4	0.000103…	10.0.1.21	10.0.3.20	ICMP	60	Echo (ping) reply	id=0x0000,	seq=1/256,…
5	0.001030…	10.0.2.20	10.0.3.20	ICMP	60	Echo (ping) reply	id=0x0000,	seq=1/256,…
6	0.001054…	10.0.2.21	10.0.3.20	ICMP	60	Echo (ping) reply	id=0x0000,	seq=1/256,…
7	0.104234…	10.0.3.20	10.0.3.21	ICMP	60	Echo (ping) request	id=0x0000,	seq=1/256,…
8	0.104275…	10.0.3.21	10.0.3.20	ICMP	42	Echo (ping) reply	id=0x0000,	seq=1/256,…
9	0.109377…	10.0.2.20	10.0.3.20	ICMP	60	Echo (ping) reply	id=0x0000,	seq=1/256,…
10	0.109406…	10.0.1.20	10.0.3.20	ICMP	60	Echo (ping) reply	id=0x0000,	seq=1/256,…
11	0.109414…	10.0.2.21	10.0.3.20	ICMP	60	Echo (ping) reply	id=0x0000,	seq=1/256,…
12	0.109432…	10.0.1.21	10.0.3.20	ICMP	60	Echo (ping) reply	id=0x0000,	seq=1/256,…
13	3.024714…	10.0.1.21	10.0.3.20	ICMP	60	Echo (ping) reply	id=0x0000,	seq=1/256,…
14	3.024751…	10.0.1.20	10.0.3.20	ICMP	60	Echo (ping) reply	id=0x0000,	seq=1/256,…
15	3.024759…	10.0.3.20	10.0.3.21	ICMP	60	Echo (ping) request	id=0x0000,	seq=1/256,…

Figure 5: Smarter ICMP Flooding Attack

Attack 3: TCP SYN Flooding

The TCP SYN flooding attack was taken directly from lab 1 and was changed slightly to match topology 1 (figure 6). In this case, the victim IP was pc6 (10.0.3.20) and the attacker was pc0 (10.0.0.20). In figure 7, one can see the Wireshark capture showing the SYN flood. One can see that all other hosts on the network are sending TCP SYN packets to pc6 which is attempting to send back ACK packets. However, because there was no listener on pc6, all of the ACK packets are connection reset or RST ACK packets.

```
source_host_list = [
    ("10.0.0.21", "00:00:00:aa:00:05"), ("10.0.1.20", "00:00:00:aa:00:06"),
    ("10.0.1.21", "00:00:00:aa:00:07"), ("10.0.2.20", "00:00:00:aa:00:08"),
    ("10.0.2.21", "00:00:00:aa:00:09"), ("10.0.3.20", "00:00:00:aa:00:0a"),
    ("10.0.3.21", "00:00:00:aa:00:0b")]

#Victim
dst_addr = "10.0.3.20"
dst_mac  = "00:00:00:aa:00:0a"
```

Figure 6: TCP SYN flooding attack changes

1996	91.83269…	10.0.1.21	10.0.3.20	TCP	74	13820 → 14742 [SYN] Seq=0 Win=64240 Len=0 …
1997	91.83270…	10.0.3.20	10.0.1.21	TCP	54	14742 → 13820 [RST, ACK] Seq=1 Ack=1 Win=0…
1998	91.83274…	10.0.2.20	10.0.3.20	TCP	74	12497 → 14742 [SYN] Seq=0 Win=64240 Len=0 …
1999	91.83275…	10.0.3.20	10.0.2.20	TCP	54	14742 → 12497 [RST, ACK] Seq=1 Ack=1 Win=0…
2000	91.83279…	10.0.2.21	10.0.3.20	TCP	74	57325 → 14742 [SYN] Seq=0 Win=64240 Len=0 …
2001	91.83279…	10.0.3.20	10.0.2.21	TCP	54	14742 → 57325 [RST, ACK] Seq=1 Ack=1 Win=0…
2002	91.83283…	10.0.3.20	10.0.3.20	TCP	74	62934 → 14742 [SYN] Seq=0 Win=64240 Len=0 …
2003	91.83285…	10.0.3.21	10.0.3.20	TCP	74	65022 → 14742 [SYN] Seq=0 Win=64240 Len=0 …
2004	91.83285…	10.0.3.20	10.0.3.21	TCP	54	14742 → 65022 [RST, ACK] Seq=1 Ack=1 Win=0…
2005	92.33539…	10.0.0.21	10.0.3.20	TCP	74	18838 → 14742 [SYN] Seq=0 Win=64240 Len=0 …
2006	92.33540…	10.0.3.20	10.0.0.21	TCP	54	14742 → 18838 [RST, ACK] Seq=1 Ack=1 Win=0…
2007	92.33546…	10.0.1.21	10.0.3.20	TCP	74	20237 → 14742 [SYN] Seq=0 Win=64240 Len=0 …
2008	92.33547…	10.0.3.20	10.0.1.21	TCP	54	14742 → 20237 [RST, ACK] Seq=1 Ack=1 Win=0…
2009	92.33551…	10.0.1.20	10.0.3.20	TCP	74	45509 → 14742 [SYN] Seq=0 Win=64240 Len=0 …
2010	92.33551…	10.0.3.20	10.0.1.20	TCP	54	14742 → 45509 [RST, ACK] Seq=1 Ack=1 Win=0…
2011	92.33556…	10.0.2.20	10.0.3.20	TCP	74	10539 → 14742 [SYN] Seq=0 Win=64240 Len=0 …
2012	92.33556…	10.0.3.20	10.0.2.20	TCP	54	14742 → 10539 [RST, ACK] Seq=1 Ack=1 Win=0…
2013	92.33560…	10.0.2.21	10.0.3.20	TCP	74	27752 → 14742 [SYN] Seq=0 Win=64240 Len=0 …
2014	92.33561…	10.0.3.20	10.0.3.20	TCP	74	12660 → 14742 [SYN] Seq=0 Win=64240 Len=0 …
2015	92.33562…	10.0.3.21	10.0.3.20	TCP	74	26368 → 14742 [SYN] Seq=0 Win=64240 Len=0 …
2016	92.33563…	10.0.3.20	10.0.3.21	TCP	54	14742 → 26368 [RST, ACK] Seq=1 Ack=1 Win=0…
2017	92.83841…	10.0.0.21	10.0.3.20	TCP	74	63061 → 14742 [SYN] Seq=0 Win=64240 Len=0 …
2018	92.83844…	10.0.3.20	10.0.0.21	TCP	54	14742 → 63061 [RST, ACK] Seq=1 Ack=1 Win=0…
2019	92.83851…	10.0.1.21	10.0.3.20	TCP	74	53742 → 14742 [SYN] Seq=0 Win=64240 Len=0 …

Figure 7: Wireshark capture of TCP SYN Flood

Section 2: Detection of attacks using controller and switch statistics

Detection of attacks revolves around two data structures: the connection list and the controller statistics dictionary. The connection list dictionary uses a normal integer index as its primary key. Each corresponding value is a list containing the dpid of the switch and the source and destination IP address (figure 8). The controller statistics data structure is a nested dictionary that also uses an integer index as its primary key. Each integer key has two sub-dictionaries with “icmp” and “tcp” being the keys for each. The “icmp” dictionary value contains 8 fields (4 relating to ICMP request and 4 for ICMP response). For both types of ICMP packets, there are fields for the time the first one showed arrived in the current interval, how many have arrived since as well as if and for how long has rate limiting occurred (figure 9). The “tcp” dictionary is exactly the same as the “icmp” dictionary, but it contains half the number of fields.

The connection list and controller statistics data structures are also linked by corresponding integer indexes i.e if the first index in the connection list maps to sw0 with a packet going from pc0 to pc6, then the data in the first index of the controller statistics structure would also map to that connection.

Detection of an attack works as follows. First, the controller checks to see if the current interval has elapsed. The interval is preset (currently set to 2 seconds) and starts when an ICMP request or response packet is detected. The packet’s arrival time is updated in the controller statistics data structure and “1” is placed in the packet count. At any time in the next two seconds, another ICMP request or reply packet occurs the packet count field is incremented. After two seconds, the statistics are then analyzed. If the packet count field is greater than the preset ICMP packet threshold (currently set to 10), then an attack has been detected, the log file is updated (section 4), and rate-limiting can begin (section 3).

```
    connection_list[len(connection_list)] = [dp.id, pkt_ipv4.src, pkt_ipv4.dst]
```

Figure 8: Connection list data structure

```
controller_statistics[host] = {
    "icmp" : [None,  # Time of first recorded ICMP request packet in this interval
              None,  # Number of ICMP request packets in this interval
              None,  # Time since rate limiting started for ICMP request packets for this connection
              False, # Boolean flag for if rate limiting has been started for this connection

              None,  # Time of first recorded ICMP reply packet in this interval
              None,  # Number of ICMP reply packets in this interval
              None,  # Time since rate limitign started for ICMP reply packets for this connection
              False],# Boolean flag for if rate limiting has been started for this connection

    "tcp"  : [None,  # Time of first recorded TCP packet in this interval
              None,  # Number of TCP packets in this interval
              None,  # Time since rate limiting started for ICMP request packets for this connection
              False, # Boolean flag for if rate limiting has been started for this connection
              ]
}
```

Figure 9: Controller statistics data structure

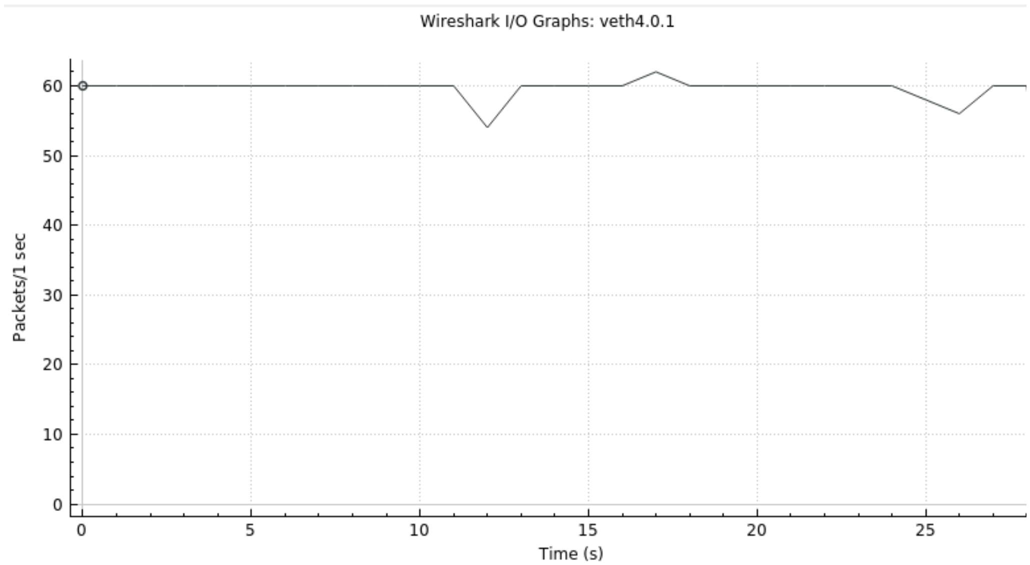
Section 3: Attack prevention via rate limiting

Rate limiting for both ICMP and TCP packets is controlled by the controller statistics data structure. Once an attack has been detected, the rate limit boolean flag is set to true, and its corresponding date-time entry is set to the current date-time. When rate-limiting is enabled, the controller, for that specific connection, will monitor the number of detected packets for a preset limiting period (in this case, 2 seconds). If the number of packets that have elapsed is greater than the preset rate (0.5 packets per second), then all further packets will be dropped. Wireshark's auto-generated I/O graphs were used to prove that for each attack, rate-limiting was enabled and helped to mitigate its effects.

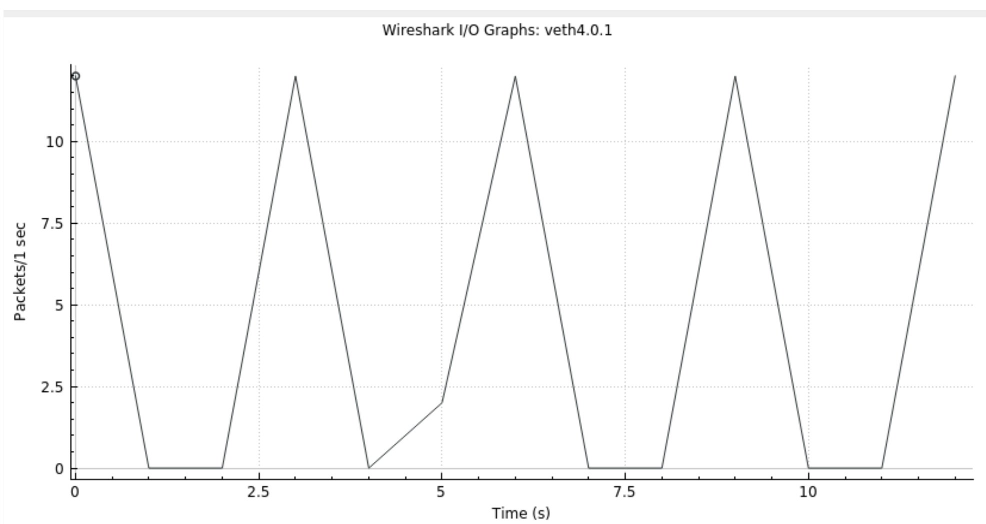
Graphs 1 and 2 were generated during the execution of the standard ICMP flood attack. Without rate limiting, the average packet/second rate was around 60 (graph 1). However, with rate-limiting enabled (graph 2), the packet/second rate never goes higher than 12. After the attack is detected, there is a period of time where no packets are able to be sent as the threshold was reached, before the pattern starts again. This results in the sawtooth line graph. As the standard ICMP attack and standard/smarter smurf ICMP attacks result in the same amount of packets being sent, their graphs both with (graph 4) and without (graph 3) rate-limiting look very similar.

Graphs 5 and 6 were generated during the execution of the TCP SYN flood attack. Both graphs have a black line which represents the TCP SYN packets being sent to the victim and the green line represents the number of TCP RST ACK packets being sent from the victim. Before rate-limiting, the SYN and RST ACT packet rates were 50 and 20 packets/second respectively, but after rate-limiting, their numbers dropped down to 10 and 4 packets/second respectively.

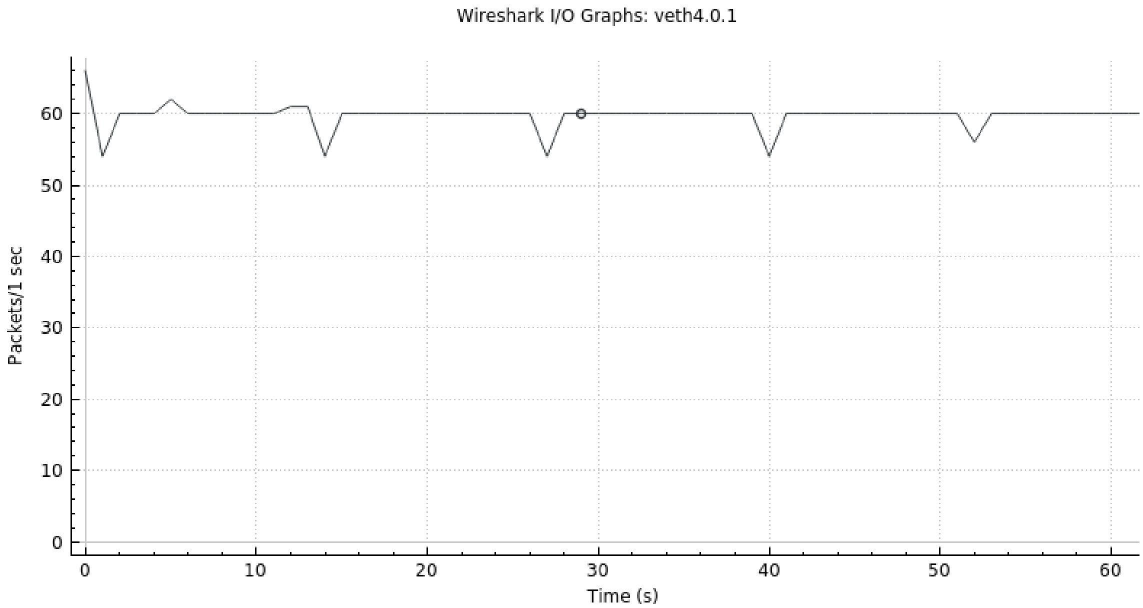
Graphs



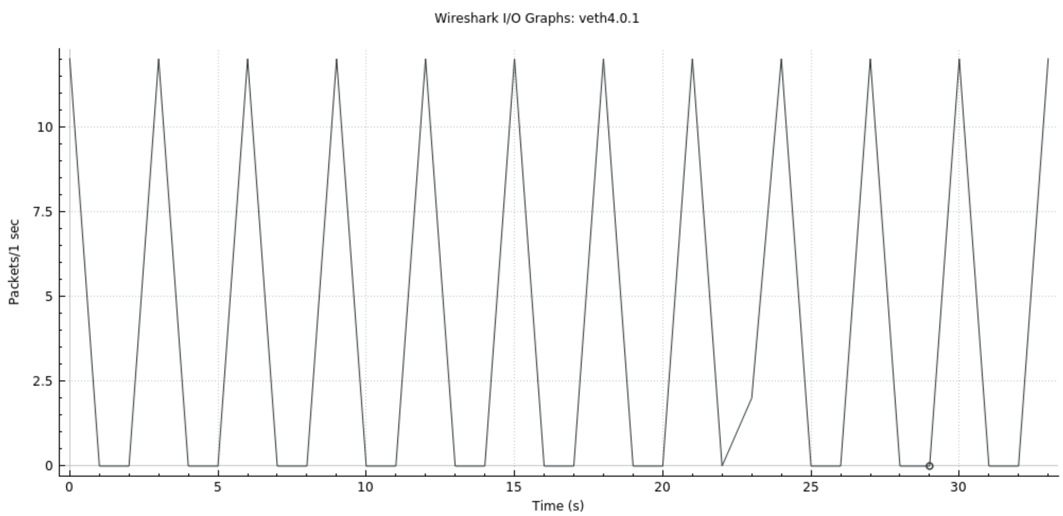
Graph 1: ICMP Standard Flood Attack I/O graph without Rate Limiting



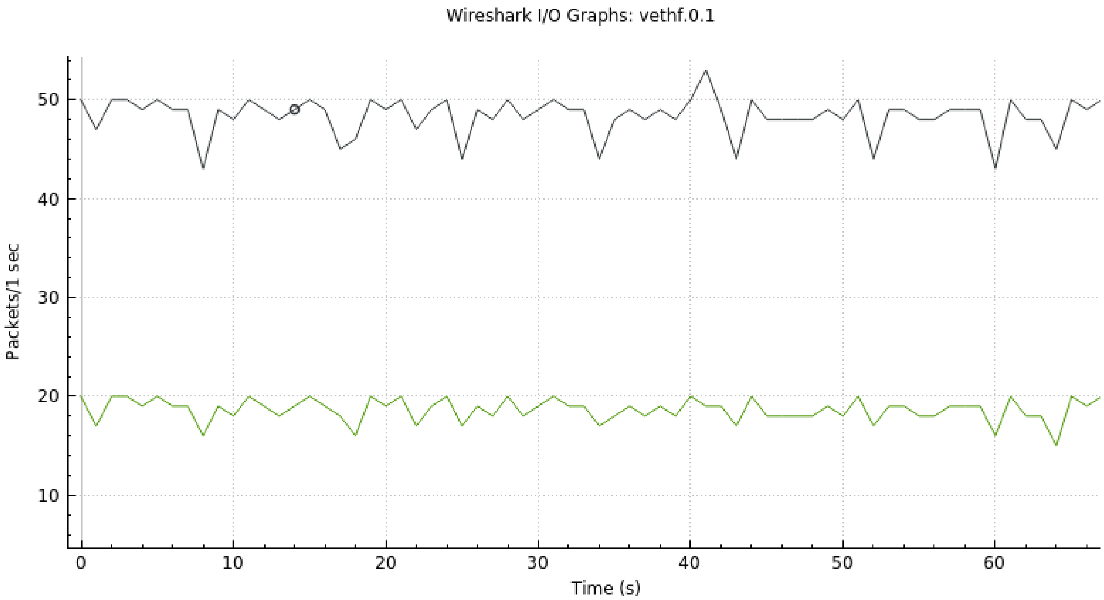
Graph 2: ICMP Standard Flood Attack I/O graph with Rate Limiting



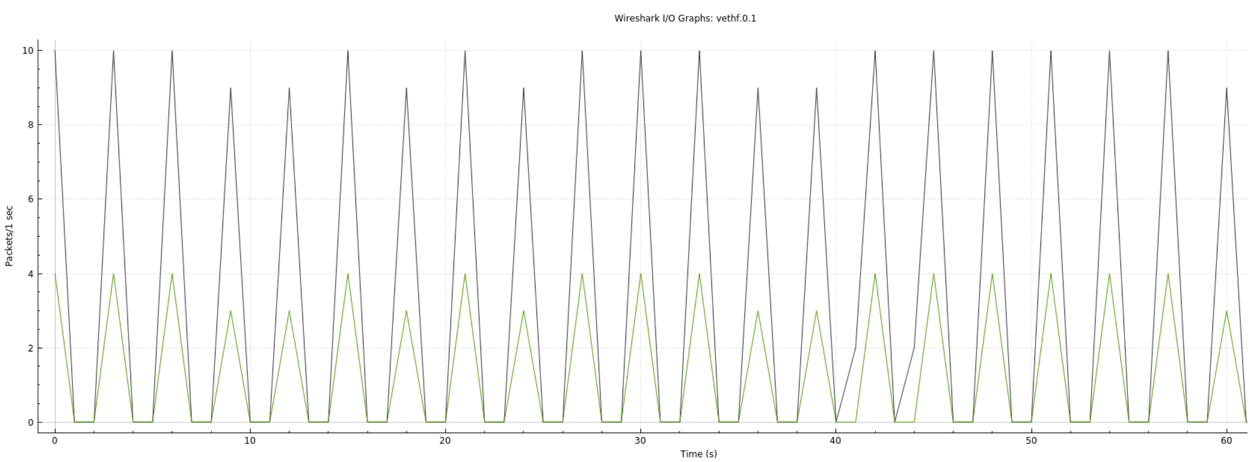
Graph 3: ICMP Smarter Smurf Flood Attack I/O graph without Rate Limiting



Graph 4: ICMP Smarter Smurf Flood Attack I/O graph with Rate Limiting



Graph 5: TCP SYN Flood Attack I/O graph without Rate Limiting



Graph 6: TCP SYN Flood Attack I/O graph with Rate Limiting

Section 4: Packet log creation

The final part of the lab was the setup of a logging and alerting system that would tell the user what type of attack was starting, where it was coming from, and what steps were being taken to mitigate it. The file is named “cntrl_log_file.csv” and contains headers for the date-time that the packet was detected, the packet_type, dpid, source mac, and IP address, destination mac and IP address, and lastly, a comment. The log file is appended every time a new packet is detected. However, whenever an attack occurs, a new message also appears that describes the type of attack as well as the fact that rate limiting has been started. The comment field serves several purposes - it tells the user if the packet has reached its final destination, if the packet is being sent while being rate limited, or if the packet was dropped as a result of rate-limiting. Figures 10 and 11 show the regular sending and receiving of ICMP and TCP packets while figures 12 and 13 show the detection of the attack as well as packets being sent and dropped while rate-limiting. The controller terminal is also updated whenever an attack takes place (figure 14).

```
date_time, packet_type, dpid, src_mac, src_ip, dst_mac, dst_ip, comment
2022-03-30 19:53:56.307614, TCP Packet, 11141120, 00:00:00:aa:00:0a, 10.0.0.21, 00:00:00:aa:00:05, 10.0.3.20,
2022-03-30 19:53:56.308082, TCP Packet, 11141120, 00:00:00:aa:00:0a, 10.0.3.20, 00:00:00:aa:00:0a, 10.0.3.20,
2022-03-30 19:53:56.308300, TCP Packet, 11141120, 00:00:00:aa:00:0a, 10.0.3.21, 00:00:00:aa:00:0b, 10.0.3.20,
2022-03-30 19:53:56.309595, TCP Packet, 11141122, ff:ff:ff:ff:ff:ff, 10.0.0.21, 00:00:00:aa:00:10, 10.0.3.20,
2022-03-30 19:53:56.309863, TCP Packet, 11141122, ff:ff:ff:ff:ff:ff, 10.0.3.20, 00:00:00:aa:00:10, 10.0.3.20,
2022-03-30 19:53:56.310056, TCP Packet, 11141122, ff:ff:ff:ff:ff:ff, 10.0.3.21, 00:00:00:aa:00:10, 10.0.3.20,
2022-03-30 19:53:56.310914, TCP Packet, 11141123, ff:ff:ff:ff:ff:ff, 10.0.0.21, 00:00:00:aa:00:0e, 10.0.3.20, Packet has reached its destination
2022-03-30 19:53:56.311115, TCP Packet, 11141123, ff:ff:ff:ff:ff:ff, 10.0.3.20, 00:00:00:aa:00:0e, 10.0.3.20, Packet has reached its destination
2022-03-30 19:53:56.311322, TCP Packet, 11141123, ff:ff:ff:ff:ff:ff, 10.0.3.21, 00:00:00:aa:00:0e, 10.0.3.20, Packet has reached its destination
2022-03-30 19:53:56.408483, TCP Packet, 11141120, 00:00:00:aa:00:0a, 10.0.0.21, 00:00:00:aa:00:05, 10.0.3.20,
```

Figure 10: Regular TCP Logs

```
2022-03-30 19:56:37.932162, ICMP ECHO REQUEST, 11141120, 00:00:00:aa:00:00, 10.0.3.20, 00:00:00:aa:00:7b, 10.0.1.20,
2022-03-30 19:56:37.932433, ICMP ECHO REQUEST, 11141120, 00:00:00:aa:00:00, 10.0.3.20, 00:00:00:aa:00:72, 10.0.1.21,
2022-03-30 19:56:37.932616, ICMP ECHO REQUEST, 11141120, 00:00:00:aa:00:00, 10.0.3.20, 00:00:00:aa:00:d0, 10.0.2.20,
2022-03-30 19:56:37.932809, ICMP ECHO REQUEST, 11141120, 00:00:00:aa:00:00, 10.0.3.20, 00:00:00:aa:00:b0, 10.0.2.21,
2022-03-30 19:56:37.932987, ICMP ECHO REQUEST, 11141120, 00:00:00:aa:00:00, 10.0.3.20, 00:00:00:aa:00:c3, 10.0.3.21,
2022-03-30 19:56:37.933804, ICMP ECHO REQUEST, 11141121, ff:ff:ff:ff:ff:ff, 10.0.3.20, 00:00:00:aa:00:0c, 10.0.1.20, Packet has reached its destination
2022-03-30 19:56:37.933994, ICMP ECHO REQUEST, 11141121, ff:ff:ff:ff:ff:ff, 10.0.3.20, 00:00:00:aa:00:0c, 10.0.1.21, Packet has reached its destination
2022-03-30 19:56:37.934584, ICMP ECHO REQUEST, 11141122, ff:ff:ff:ff:ff:ff, 10.0.3.20, 00:00:00:aa:00:10, 10.0.2.20, Packet has reached its destination
2022-03-30 19:56:37.934788, ICMP ECHO REQUEST, 11141122, ff:ff:ff:ff:ff:ff, 10.0.3.20, 00:00:00:aa:00:10, 10.0.2.21, Packet has reached its destination
2022-03-30 19:56:37.934967, ICMP ECHO REQUEST, 11141122, ff:ff:ff:ff:ff:ff, 10.0.3.20, 00:00:00:aa:00:10, 10.0.3.21,
2022-03-30 19:56:37.935471, ICMP ECHO REPLY  , 11141121, 00:00:00:aa:00:01, 10.0.1.20, 00:00:00:aa:00:06, 10.0.3.20,
2022-03-30 19:56:37.935658, ICMP ECHO REPLY  , 11141121, 00:00:00:aa:00:01, 10.0.1.21, 00:00:00:aa:00:07, 10.0.3.20,
2022-03-30 19:56:37.936458, ICMP ECHO REPLY  , 11141122, 00:00:00:aa:00:02, 10.0.2.20, 00:00:00:aa:00:08, 10.0.3.20,
2022-03-30 19:56:37.936647, ICMP ECHO REPLY  , 11141122, 00:00:00:aa:00:02, 10.0.2.21, 00:00:00:aa:00:09, 10.0.3.20,
2022-03-30 19:56:37.936841, ICMP ECHO REQUEST, 11141123, ff:ff:ff:ff:ff:ff, 10.0.3.20, 00:00:00:aa:00:0e, 10.0.3.21, Packet has reached its destination
2022-03-30 19:56:37.937018, ICMP ECHO REPLY  , 11141123, ff:ff:ff:ff:ff:ff, 10.0.1.20, 00:00:00:aa:00:12, 10.0.3.20, Packet has reached its destination
2022-03-30 19:56:37.937193, ICMP ECHO REPLY  , 11141123, ff:ff:ff:ff:ff:ff, 10.0.1.21, 00:00:00:aa:00:12, 10.0.3.20, Packet has reached its destination
2022-03-30 19:56:37.938416, ICMP ECHO REPLY  , 11141123, ff:ff:ff:ff:ff:ff, 10.0.2.20, 00:00:00:aa:00:0e, 10.0.3.20, Packet has reached its destination
2022-03-30 19:56:37.938682, ICMP ECHO REPLY  , 11141123, ff:ff:ff:ff:ff:ff, 10.0.2.21, 00:00:00:aa:00:0e, 10.0.3.20, Packet has reached its destination
```

Figure 11: Regular ICMP Logs

```
==================================================
ATTACK DETECTED: TCP SYN Flood Detected
Starting rate limiting
2022-03-30 20:05:44.336992, TCP Packet, 11141120, 00:00:00:aa:00:0a, 10.0.0.21, 00:00:00:aa:00:05, 10.0.3.20,
==================================================
2022-03-30 20:05:44.337129, TCP Packet, 11141120, 00:00:00:aa:00:0a, 10.0.0.21, 00:00:00:aa:00:05, 10.0.3.20, Sent During Rate Limiting
2022-03-30 20:05:44.337860, TCP Packet, 11141120, 00:00:00:aa:00:0a, 10.0.3.20, 00:00:00:aa:00:0a, 10.0.3.20, Sent During Rate Limiting
2022-03-30 20:05:44.338082, TCP Packet, 11141120, 00:00:00:aa:00:0a, 10.0.3.21, 00:00:00:aa:00:0b, 10.0.3.20, Sent During Rate Limiting
2022-03-30 20:05:44.339342, TCP Packet, 11141122, ff:ff:ff:ff:ff:ff, 10.0.0.21, 00:00:00:aa:00:10, 10.0.3.20,
2022-03-30 20:05:44.339612, TCP Packet, 11141122, ff:ff:ff:ff:ff:ff, 10.0.3.20, 00:00:00:aa:00:10, 10.0.3.20,
2022-03-30 20:05:44.339809, TCP Packet, 11141122, ff:ff:ff:ff:ff:ff, 10.0.3.21, 00:00:00:aa:00:10, 10.0.3.20,
2022-03-30 20:05:44.437866, TCP Packet, 11141120, 00:00:00:aa:00:0a, 10.0.0.21, 00:00:00:aa:00:05, 10.0.3.20, Sent During Rate Limiting
2022-03-30 20:05:44.438395, TCP Packet, 11141120, 00:00:00:aa:00:0a, 10.0.3.20, 00:00:00:aa:00:0a, 10.0.3.20, Sent During Rate Limiting
2022-03-30 20:05:44.438593, TCP Packet, 11141120, 00:00:00:aa:00:0a, 10.0.3.21, 00:00:00:aa:00:0b, 10.0.3.20, Sent During Rate Limiting
2022-03-30 20:05:44.439870, TCP Packet, 11141122, ff:ff:ff:ff:ff:ff, 10.0.0.21, 00:00:00:aa:00:10, 10.0.3.20,
2022-03-30 20:05:44.440158, TCP Packet, 11141122, ff:ff:ff:ff:ff:ff, 10.0.3.20, 00:00:00:aa:00:10, 10.0.3.20,
2022-03-30 20:05:44.440381, TCP Packet, 11141122, ff:ff:ff:ff:ff:ff, 10.0.3.21, 00:00:00:aa:00:10, 10.0.3.20,
2022-03-30 20:05:44.539027, TCP Packet, 11141120, 00:00:00:aa:00:0a, 10.0.0.21, 00:00:00:aa:00:05, 10.0.3.20, Dropped During Rate Limiting
2022-03-30 20:05:44.539472, TCP Packet, 11141120, 00:00:00:aa:00:0a, 10.0.3.20, 00:00:00:aa:00:0a, 10.0.3.20, Dropped During Rate Limiting
2022-03-30 20:05:44.539610, TCP Packet, 11141120, 00:00:00:aa:00:0a, 10.0.3.21, 00:00:00:aa:00:0b, 10.0.3.20, Dropped During Rate Limiting
2022-03-30 20:05:44.639764, TCP Packet, 11141120, 00:00:00:aa:00:0a, 10.0.0.21, 00:00:00:aa:00:05, 10.0.3.20, Dropped During Rate Limiting
2022-03-30 20:05:44.640233, TCP Packet, 11141120, 00:00:00:aa:00:0a, 10.0.3.20, 00:00:00:aa:00:0a, 10.0.3.20, Dropped During Rate Limiting
```

Figure 12: TCP Flood Attack Detected

```
==================================================
ATTACK DETECTED: ICMP FLood Attack Detected
Starting rate limiting
2022-03-30 20:09:16.508172, ICMP ECHO REQUEST, 11141120, 00:00:00:aa:00:00, 10.0.3.20, 00:00:00:aa:00:73, 10.0.1.20,
==================================================
2022-03-30 20:09:16.508295, ICMP ECHO REQUEST, 11141120, 00:00:00:aa:00:00, 10.0.3.20, 00:00:00:aa:00:73, 10.0.1.20, Sent During Rate Limiting
2022-03-30 20:09:16.508499, ICMP ECHO REQUEST, 11141120, 00:00:00:aa:00:00, 10.0.3.20, 00:00:00:aa:00:ec, 10.0.1.21, Sent During Rate Limiting
2022-03-30 20:09:16.508679, ICMP ECHO REQUEST, 11141120, 00:00:00:aa:00:00, 10.0.3.20, 00:00:00:aa:00:d4, 10.0.2.20, Sent During Rate Limiting
2022-03-30 20:09:16.508866, ICMP ECHO REQUEST, 11141120, 00:00:00:aa:00:00, 10.0.3.20, 00:00:00:aa:00:dc, 10.0.2.21, Sent During Rate Limiting
2022-03-30 20:09:16.509044, ICMP ECHO REQUEST, 11141120, 00:00:00:aa:00:00, 10.0.3.20, 00:00:00:aa:00:e0, 10.0.3.21, Sent During Rate Limiting
2022-03-30 20:09:16.510253, ICMP ECHO REQUEST, 11141121, ff:ff:ff:ff:ff:ff, 10.0.3.20, 00:00:00:aa:00:0c, 10.0.1.20, Packet has reached its destination
2022-03-30 20:09:16.510525, ICMP ECHO REQUEST, 11141121, ff:ff:ff:ff:ff:ff, 10.0.3.20, 00:00:00:aa:00:0c, 10.0.1.21, Packet has reached its destination
2022-03-30 20:09:16.511174, ICMP ECHO REQUEST, 11141122, ff:ff:ff:ff:ff:ff, 10.0.3.20, 00:00:00:aa:00:10, 10.0.2.21, Packet has reached its destination
2022-03-30 20:09:16.511377, ICMP ECHO REQUEST, 11141122, ff:ff:ff:ff:ff:ff, 10.0.3.20, 00:00:00:aa:00:10, 10.0.2.20, Packet has reached its destination
2022-03-30 20:09:16.511558, ICMP ECHO REQUEST, 11141122, ff:ff:ff:ff:ff:ff, 10.0.3.20, 00:00:00:aa:00:10, 10.0.3.21,
2022-03-30 20:09:16.710422, ICMP ECHO REQUEST, 11141120, 00:00:00:aa:00:00, 10.0.3.20, 00:00:00:aa:00:73, 10.0.1.20, Dropped During Rate Limiting
2022-03-30 20:09:16.710601, ICMP ECHO REQUEST, 11141120, 00:00:00:aa:00:00, 10.0.3.20, 00:00:00:aa:00:ec, 10.0.1.21, Dropped During Rate Limiting
2022-03-30 20:09:16.710725, ICMP ECHO REQUEST, 11141120, 00:00:00:aa:00:00, 10.0.3.20, 00:00:00:aa:00:d4, 10.0.2.20, Dropped During Rate Limiting
2022-03-30 20:09:16.710849, ICMP ECHO REQUEST, 11141120, 00:00:00:aa:00:00, 10.0.3.20, 00:00:00:aa:00:dc, 10.0.2.21, Dropped During Rate Limiting
2022-03-30 20:09:16.710973, ICMP ECHO REQUEST, 11141120, 00:00:00:aa:00:00, 10.0.3.20, 00:00:00:aa:00:e0, 10.0.3.21, Dropped During Rate Limiting
2022-03-30 20:09:16.811516, ICMP ECHO REQUEST, 11141120, 00:00:00:aa:00:00, 10.0.3.20, 00:00:00:aa:00:73, 10.0.1.20, Dropped During Rate Limiting
2022-03-30 20:09:16.811719, ICMP ECHO REQUEST, 11141120, 00:00:00:aa:00:00, 10.0.3.20, 00:00:00:aa:00:ec, 10.0.1.21, Dropped During Rate Limiting
2022-03-30 20:09:16.811847, ICMP ECHO REQUEST, 11141120, 00:00:00:aa:00:00, 10.0.3.20, 00:00:00:aa:00:dc, 10.0.2.21, Dropped During Rate Limiting
```

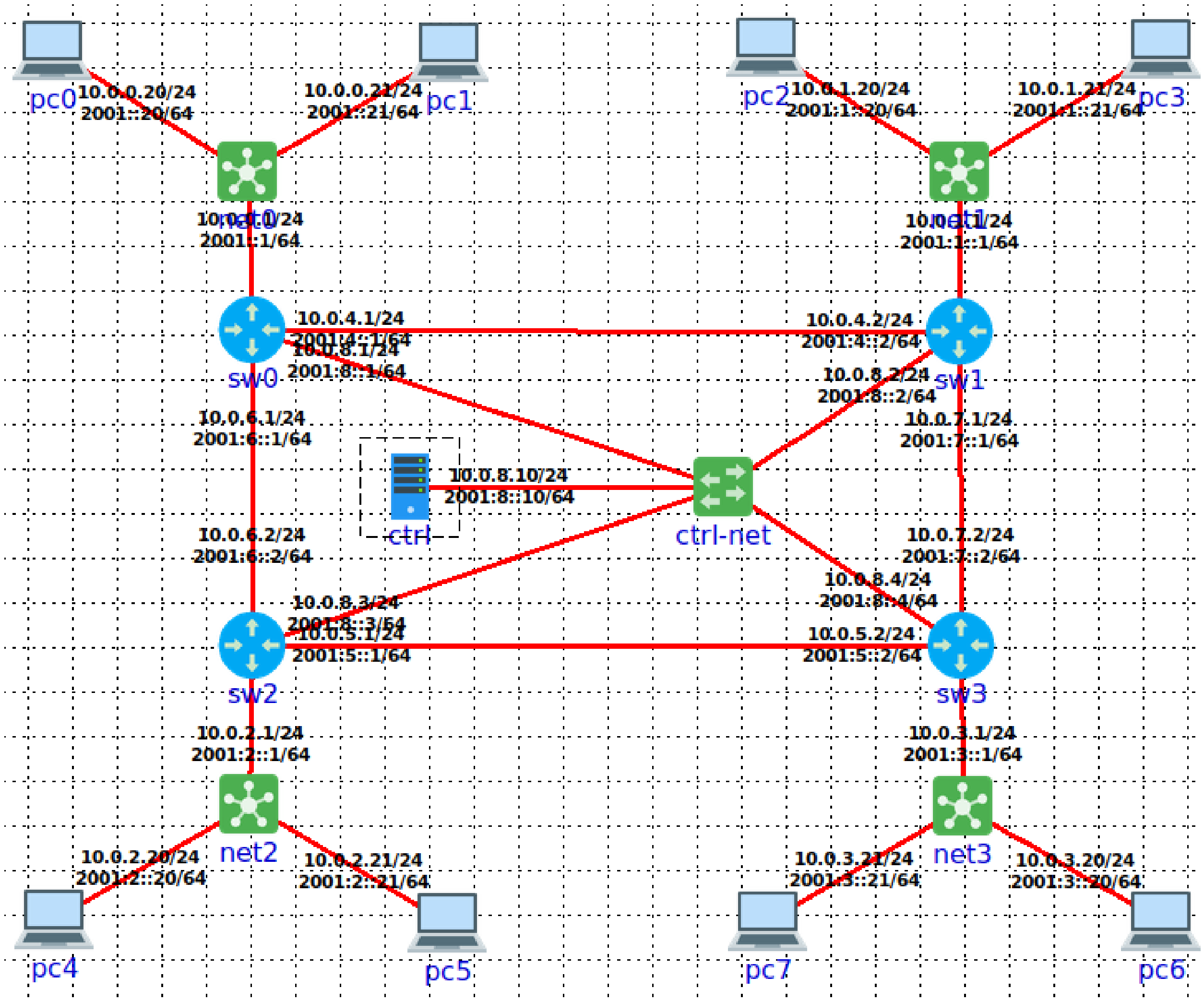
Figure 13: ICMP Flood Attack Detected

```
ATTACK DETECTED: ICMP FLood Attack Detected

2022-03-30 20:12:44.896827, ICMP ECHO REQUEST, 11141120, 00:00:00:aa:00:00, 10.0.3.20, 00:00:00:aa:00:47, 10.0.1.20,
```

```
ATTACK DETECTED: TCP SYN Flood Detected

2022-03-30 20:13:46.720459, TCP Packet, 11141120, 00:00:00:aa:00:0a, 10.0.0.21, 00:00:00:aa:00:05, 10.0.3.20,
```

Figure 14: Controller terminal output

Topology



Topology 1