CSC7078 / Aaron McCullagh / 40036410

Evaluation

"The success of the solution will be determined by:

- Efficient detection and mitigation of the SYN flood attack, based on the packet count threshold that is set in the application, dropping packets when threshold is exceeded, and blocking future traffic from malicious source.
- Effectively distinguishing between SYN flood traffic and benign TCP packets, allowing legitimate traffic to pass through the network while application is running.
- Limited impact on network performance caused by the application running at the controller. Traffic monitoring using 'iperf' will be used to determine if latency is introduced, comparing network traffic bandwidth when application is, and is not running. " (*project proposal*; *measuring success*)

To evaluate the success of the solution, in line with the evaluation methods discussed in the project proposal, the following steps are carried out:

1. Launch "ddos_application.py" at Ryu controller:

```
vagrant@ubuntu-focal:/vagrant_data$ python3 -m py_compile local/apps/src/ddos_application.py && ryu-manag
er --use-stderr --nouse-syslog --log-conf "" local/apps/src/ddos_application.py
loading app local/apps/src/ddos_application.py
loading app ryu.controller.ofp_handler
instantiating app local/apps/src/ddos_application.py of Syn_Flood_Detection
instantiating app ryu.controller.ofp_handler of OFPHandler
```

2. Launch "ddos_scenario.yaml" in mininet:

```
vagrant@ubuntu-focal:/vagrant_data$ sudo -E python3 remote/script_run_mininet.py /vagrant_data/local/app
s/scenarios/ddos_scenario.yaml
```

3. Check network topology:

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
h4 h4-eth0:s1-eth4
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0 s1-eth4:h4-eth0
c0
```

4. Test network connectivity:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet>
```

5. Send excess packets per second that's exceeds threshold value, from h1 to h2:

```
mininet> h1 hping3 -i u1000 -S 10.0.0.2
```

6. Observe result:

```
ALERT: Blocking TCP SYN source in_port=1 : SYN packet speed 246 pps exceeds threshold
```

```
--- 10.0.0.2 hping statistic ---
2764 packets transmitted, 1020 packets received, 64% packet loss
round-trip min/avg/max = 0.2/88.6/1005.2 ms
```

7. Check flow rules in flow table:

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=82.306s, table=0, n_packets=1751, n_bytes=94534, priority=100,in_port="s1-eth1" ac
tions=drop
 cookie=0x0, duration=83.457s, table=0, n_packets=0, n_bytes=0, idle_timeout=180, priority=10,tcp,in_por
t="s1-eth2",dl_src=2a:6f:17:36:cf:2a,dl_dst=ee:d5:6d:65:f4:8f,tcp_flags=syn actions=output:"s1-eth1"
 cookie=0x0, duration=82.810s, table=0, n_packets=811, n_bytes=43794, idle_timeout=180, priority=10,tcp,
in_port="s1-eth1",dl_src=ee:d5:6d:65:f4:8f,dl_dst=2a:6f:17:36:cf:2a,tcp_flags=syn actions=output:"s1-eth
2"
 cookie=0x0, duration=83.457s, table=0, n_packets=1008, n_bytes=54396, idle_timeout=180, priority=1,in_p
ort="s1-eth2",dl_src=2a:6f:17:36:cf:2a,dl_dst=ee:d5:6d:65:f4:8f actions=output:"s1-eth1"
 cookie=0x0, duration=82.810s, table=0, n_packets=0, n_bytes=0, idle_timeout=180, priority=1,in_port="s1
-eth1",dl_src=ee:d5:6d:65:f4:8f,dl_dst=2a:6f:17:36:cf:2a actions=output:"s1-eth2"
 cookie=0x0, duration=88.108s, table=0, n_packets=224, n_bytes=12144, priority=0 actions=CONTROLLER:6553
5
```

8. Test connectivity between hosts:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X
h2 -> X h3 h4
h3 -> X h2 h4
h4 -> X h2 h3
*** Results: 50% dropped (6/12 received)
```

Steps 1 to 8, demonstrate that when the TCP SYN packet per second rate exceeds the threshold value of 100 PPS, a drop rule is added to the flow table, to protect the network from traffic incoming on the port on which the malicious source has been detected.

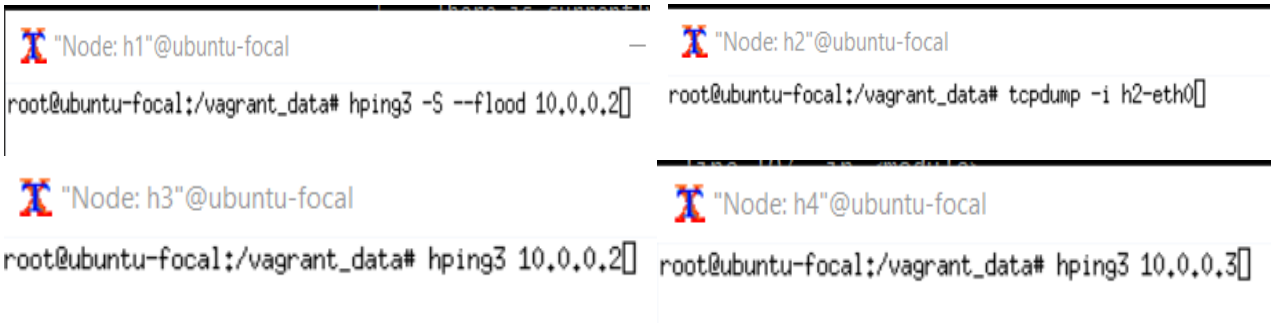
9. Re-launch application at controller to clear flow table:

```
^Cvagrant@ubuntu-focal:/vagrant_data$ python3 -m py_compile local/apps/src/ddos_application.py && ryu-man
ager --use-stderr --nouse-syslog --log-conf "" local/apps/src/ddos_application.py
loading app local/apps/src/ddos_application.py
loading app ryu.controller.ofp_handler
instantiating app local/apps/src/ddos_application.py of Syn_Flood_Detection
instantiating app ryu.controller.ofp_handler of OFPHandler
```

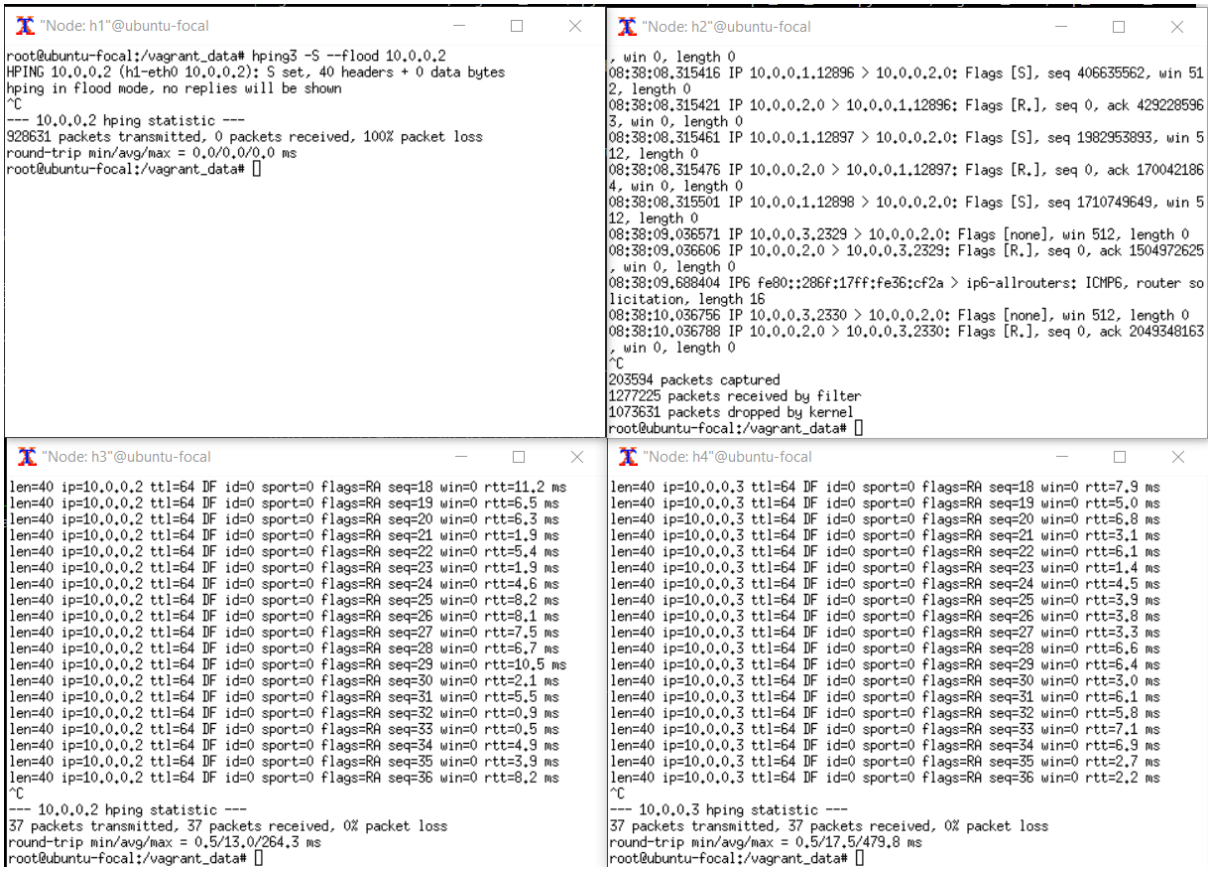
10. Launch xterms for h1, h2, h3 & h4:

```
mininet> xterm h1
mininet> xterm h2
mininet> xterm h3
mininet> xterm h4
```

11. Launch; packet in listening on h2 - TCP SYN flood attack from h1 to h2 - legitimate traffic from h3 to h2 - legitimate traffic from h4 to h3:



12. Observe results:




```
packet in S: 1, src: 2a:6f:17:36:cf:2a, dst: ee:d5:6d:65:f4:8f, in_port: 2
packet in S: 1, src: 2a:6f:17:36:cf:2a, dst: ee:d5:6d:65:f4:8f, in_port: 2
packet in S: 1, src: 2a:6f:17:36:cf:2a, dst: ee:d5:6d:65:f4:8f, in_port: 2
ALERT: Blocking TCP SYN source in_port=1 : SYN packet speed 20866 pps exceeds threshold
```

13. Check flow rules in flow table:

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=178.234s, table=0, n_packets=663737, n_bytes=35841742, priority=100,in_port="s1-eth1" actions=drop
 cookie=0x0, duration=238.565s, table=0, n_packets=20866, n_bytes=1126764, idle_timeout=180, priority=10,tcp,in_port="s1-eth1",dl_src=ee:d5:6d:65:f4:8f,dl_dst=2a:6f:17:36:cf:2a,tcp_flags=syn actions=output:"s1-eth2"
 cookie=0x0, duration=15.896s, table=0, n_packets=0, n_bytes=0, idle_timeout=180, priority=10,tcp,in_port="s1-eth3",dl_src=9e:55:fb:af:f1:56,dl_dst=2a:6f:17:36:cf:2a,tcp_flags=syn actions=output:"s1-eth2"
 cookie=0x0, duration=15.640s, table=0, n_packets=0, n_bytes=0, idle_timeout=180, priority=10,tcp,in_port="s1-eth2",dl_src=2a:6f:17:36:cf:2a,dl_dst=9e:55:fb:af:f1:56,tcp_flags=syn actions=output:"s1-eth3"
 cookie=0x0, duration=6.540s, table=0, n_packets=0, n_bytes=0, idle_timeout=180, priority=10,tcp,in_port="s1-eth4",dl_src=da:bf:97:19:05:5a,dl_dst=9e:55:fb:af:f1:56,tcp_flags=syn actions=output:"s1-eth3"
 cookie=0x0, duration=6.286s, table=0, n_packets=0, n_bytes=0, idle_timeout=180, priority=10,tcp,in_port="s1-eth3",dl_src=9e:55:fb:af:f1:56,dl_dst=da:bf:97:19:05:5a,tcp_flags=syn actions=output:"s1-eth4"
 cookie=0x0, duration=234.506s, table=0, n_packets=49613, n_bytes=2679042, idle_timeout=180, priority=1,in_port="s1-eth2",dl_src=2a:6f:17:36:cf:2a,dl_dst=ee:d5:6d:65:f4:8f actions=output:"s1-eth1"
 cookie=0x0, duration=15.896s, table=0, n_packets=14, n_bytes=732, idle_timeout=180, priority=1,in_port="s1-eth3",dl_src=9e:55:fb:af:f1:56,dl_dst=2a:6f:17:36:cf:2a actions=output:"s1-eth2"
 cookie=0x0, duration=15.640s, table=0, n_packets=14, n_bytes=732, idle_timeout=180, priority=1,in_port="s1-eth2",dl_src=2a:6f:17:36:cf:2a,dl_dst=9e:55:fb:af:f1:56 actions=output:"s1-eth3"
 cookie=0x0, duration=6.540s, table=0, n_packets=4, n_bytes=192, idle_timeout=180, priority=1,in_port="s1-eth4",dl_src=da:bf:97:19:05:5a,dl_dst=9e:55:fb:af:f1:56 actions=output:"s1-eth3"
 cookie=0x0, duration=6.286s, table=0, n_packets=4, n_bytes=192, idle_timeout=180, priority=1,in_port="s1-eth3",dl_src=9e:55:fb:af:f1:56,dl_dst=da:bf:97:19:05:5a actions=output:"s1-eth4"
 cookie=0x0, duration=322.413s, table=0, n_packets=52175, n_bytes=2817482, priority=0 actions=CONTROLLER:65535
```

14. Check connectivity between hosts:

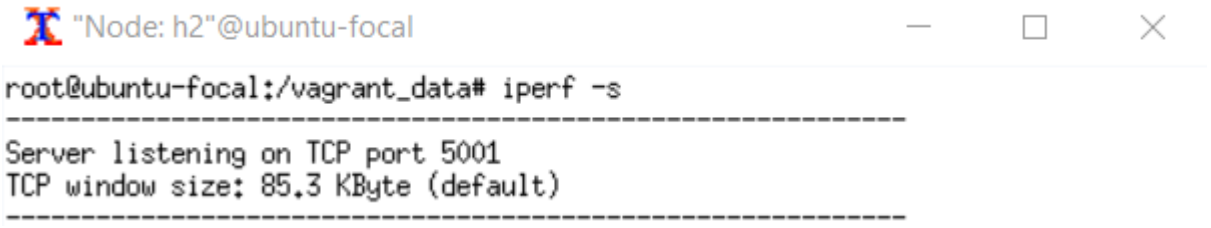
```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X
h2 -> X h3 h4
h3 -> X h2 h4
h4 -> X h2 h3
*** Results: 50% dropped (6/12 received)
```

Steps 9 – 14, demonstrate how the solution application effectively detects and mitigates a TCP SYN flood attack based on a high volume of traffic within a short time period, while allowing benign TCP traffic to continue to flow on the network, and therefore achieving the security goal of availability.

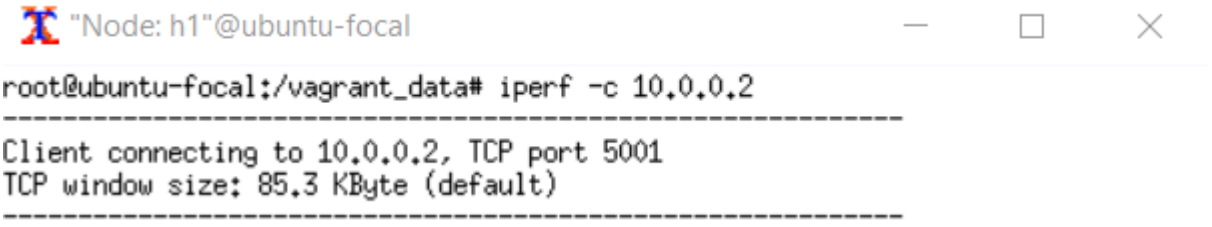
15. Launch basic application with simple learning switch at the controller & relaunch “ddos_scenario.yaml” in mininet:

```
^Cvagrant@ubuntu-focal:/vagrant_datapython3 -m py_compile local/apps/src/learning.py && ryu-manager --use
-stderr --nouse-syslog --log-conf "" local/apps/src/learning.py
loading app local/apps/src/learning.py
loading app ryu.controller.ofp_handler
instantiating app local/apps/src/learning.py of SimpleSwitch
instantiating app ryu.controller.ofp_handler of OFPHandler
```

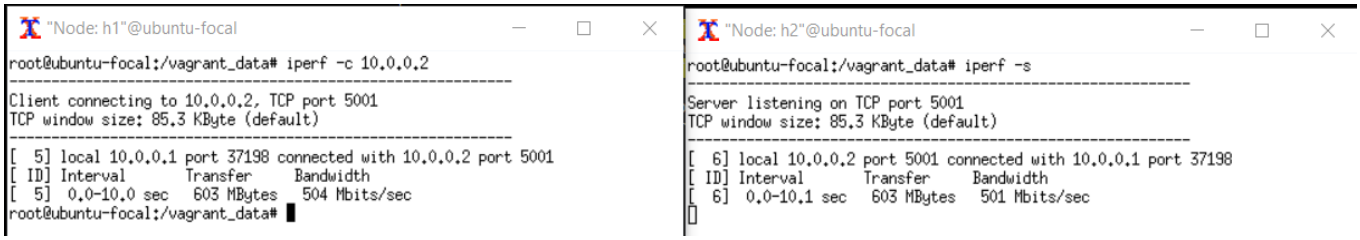
16. Setup server on h2, using iperf:



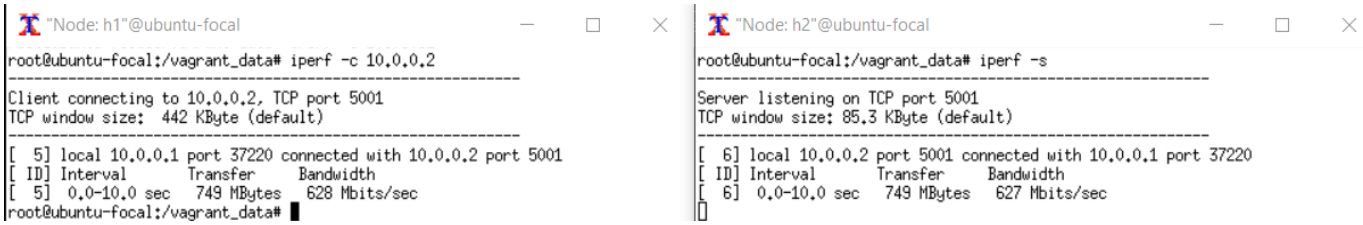
17. Setup client on h1, to communicate with h2:



18. Observe results:



19. Repeat steps 16 - 18, with solution application “ddos_application.py” running at controller & observe results:



Steps 15 – 19, demonstrate that the solution application does not negatively affect the performance of the network. Therefore, all measures to determine the success of the solution, set out in the project proposal have been achieved.

The solution of detecting packets per second and dropping packets on the in_port of the malicious host is suitable for the network topology for this project, of 4 hosts and 1 switch, as hosts sending benign TCP traffic are not prevented from accessing the network. However, in a network environment where a wireless site with multiple users are connected to the in_port on the switch, adding a drop rule to the flow table for all traffic on the in_port would not be suitable, as multiple users would lose access to the network, resulting in failure to achieve the network security goal of availability. Therefore, a solution, where the controller periodically detects flow counters, and if the set threshold is exceeded, a meter is used to limit the speed, would be more suitable to a network with a wireless site connected to an in_port on the switch.