

1 An Introduction to OpenFlow with POX Controllers

1.1 Experiment Set-up

Exercise 1.

Exercise 1.1: For your lab report: In Step 2 (Configure the Open vSwitch), Step 2a (Configure the Software Switch (OVS Window)), part v, take a screen shot for your report that shows that your software switch is configured. Explain by referencing your screen shot.

What is the output on PC A when the ping commands are issued?

◇ **Solution:** The software switch is set up with an ethernet bridge br0 and is attempting to connect to the controller at 130.127.215.151.

```
asarabi3@switch:~$ sudo ovs-vsctl show
7f96f552-dc1f-49d2-8d23-449b84165a2c
    Bridge "br0"
        Controller "tcp:130.127.215.151:6653"
        fail_mode: secure
        Port "eth3"
            Interface "eth3"
        Port "br0"
            Interface "br0"
                type: internal
        Port "eth2"
            Interface "eth2"
        Port "eth1"
            Interface "eth1"
    ovs version: "2.9.8"
```

Exercise 1.2: In Step 2, Step 2b (Point your Switch to a Controller), part v, take a screen shot for your lab report that shows that your switch is pointed to a controller. Explain by referencing your screen shot.

◇ **Solution:** The software switch is attempting to connect to the controller at 130.127.215.151.

The switch is setup since it lists the three interfaces that are specified as switch ports.

Exercise 1.3: For your lab report:

1. In Step 3b (Use a Learning Switch Controller) step 3, take a screen shot that illustrates the operation of the learning switch controller, i.e., shows the ping output between host1 and host2.
2. In Step 3b step 4, take a screen shot of the flows installed on the switch by the controller.

For both screen shots, provide a brief explanation of what you observe in your screen shot.

◇ **Solution:**

1. Screenshot of the ping
between host 1 and host 2:

```
asarabi3@host1:~$ ping 10.0.0.2 -c 10
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
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=862 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=1.16 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.918 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.906 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.803 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.704 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.714 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 7 received, +3 errors, 30% packet
rtt min/avg/max/mdev = 0.704/123.921/862.237/301.416 ms,
```

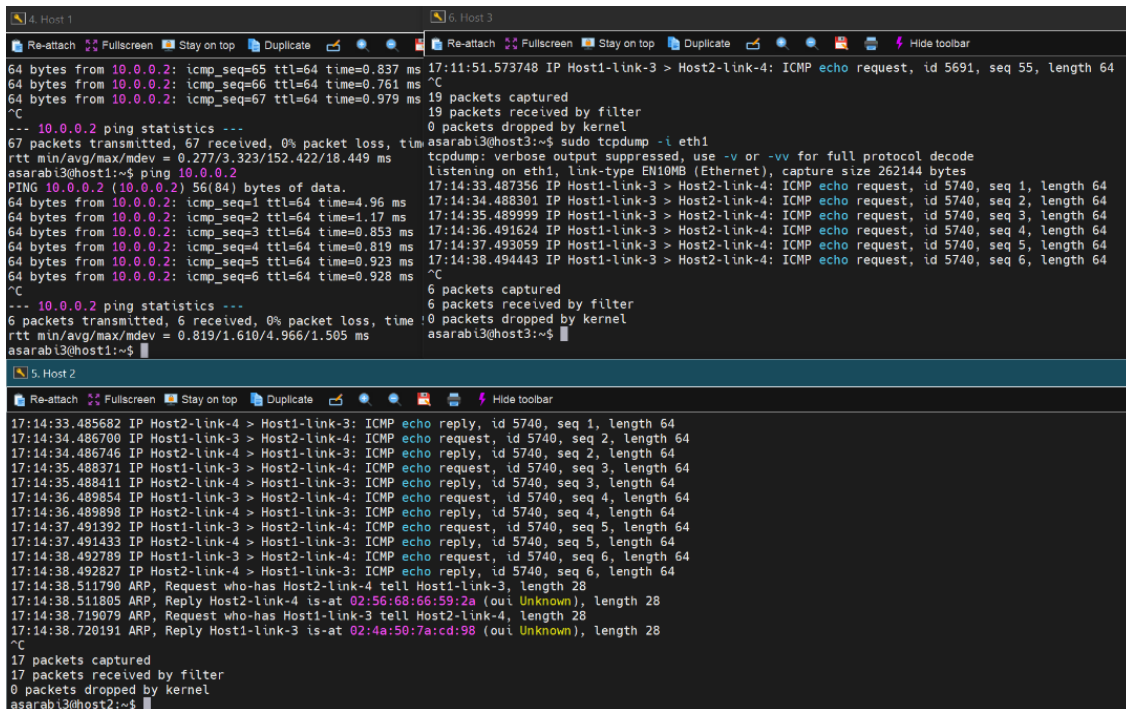
2. You should see that your controller installed flows based on the mac addresses of your packets.

```
asarabi3@switch:~$ sudo ovs-ofctl dump-flows br0
cookie=0x2000000050000000, duration=3.545s, table=0, n_packets=3, n_bytes=294, idle_timeout=5, priority=1,ip,in_port
=eth1,d_l_src=02:4a:50:7a:cd:98,d_l_dst=02:56:68:66:59:2a,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=output:eth2
cookie=0x2000000060000000, duration=3.539s, table=0, n_packets=3, n_bytes=294, idle_timeout=5, priority=1,ip,in_port
=eth2,d_l_src=02:56:68:66:59:2a,d_l_dst=02:4a:50:7a:cd:98,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=output:eth1
cookie=0x0, duration=833.818s, table=0, n_packets=13, n_bytes=910, priority=0 actions=CONTROLLER:65535
```

Because the Open vSwitch contacts the controller the first time a packet-in event happens, the first ICMP message took substantially longer than future requests. The controller then inserts the flow into the Open vSwitch flow table.

Exercise 1.4:

1. In Step 3e, after executing the `curl` command, make a note of the details of your topology such as the Data Path IDentifier (DPID) of the Open vSwitch, the MAC addresses of the hosts, and the port numbers on which the hosts are connected to the Open vSwitch, etc. These are highlighted in yellow in the instructions.
2. Use the information collected in Step 3e in Step 3f (Run a Traffic Duplication Controller), to insert a flow to duplicate traffic. Take a screen shot for your lab report that illustrates the operation of the traffic duplication controller, i.e., the `tcpdump` output that shows duplication is happening. Provide a brief explanation of what is observed in your screen shot.



```
4. Host 1
64 bytes from 10.0.0.2: icmp_seq=65 ttl=64 time=0.837 ms
64 bytes from 10.0.0.2: icmp_seq=66 ttl=64 time=0.761 ms
64 bytes from 10.0.0.2: icmp_seq=67 ttl=64 time=0.979 ms
^C
--- 10.0.0.2 ping statistics ---
67 packets transmitted, 67 received, 0% packet loss, time
rtt min/avg/max/mdev = 0.277/3.323/152.422/18.449 ms
asarabi3@host1:~$ ping 10.0.0.2
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=4.96 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=1.17 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.853 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.819 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.923 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.928 ms
^C
--- 10.0.0.2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time
rtt min/avg/max/mdev = 0.819/1.610/4.966/1.505 ms
asarabi3@host1:~$

5. Host 2
17:14:33.485682 IP Host2-link-4 > Host1-link-3: ICMP echo reply, id 5740, seq 1, length 64
17:14:34.486700 IP Host1-link-3 > Host2-link-4: ICMP echo request, id 5740, seq 2, length 64
17:14:34.486746 IP Host2-link-4 > Host1-link-3: ICMP echo reply, id 5740, seq 2, length 64
17:14:35.488371 IP Host1-link-3 > Host2-link-4: ICMP echo request, id 5740, seq 3, length 64
17:14:35.488411 IP Host2-link-4 > Host1-link-3: ICMP echo reply, id 5740, seq 3, length 64
17:14:36.489854 IP Host1-link-3 > Host2-link-4: ICMP echo request, id 5740, seq 4, length 64
17:14:36.489898 IP Host2-link-4 > Host1-link-3: ICMP echo reply, id 5740, seq 4, length 64
17:14:37.491392 IP Host1-link-3 > Host2-link-4: ICMP echo request, id 5740, seq 5, length 64
17:14:37.491433 IP Host2-link-4 > Host1-link-3: ICMP echo reply, id 5740, seq 5, length 64
17:14:38.492789 IP Host1-link-3 > Host2-link-4: ICMP echo request, id 5740, seq 6, length 64
17:14:38.492827 IP Host2-link-4 > Host1-link-3: ICMP echo reply, id 5740, seq 6, length 64
17:14:38.511790 ARP, Request who-has Host2-link-4 tell Host1-link-3, length 28
17:14:38.511805 ARP, Reply Host2-link-4 is-at 02:56:68:66:59:2a (out Unknown), length 28
17:14:38.719079 ARP, Request who-has Host1-link-3 tell Host2-link-4, length 28
17:14:38.720191 ARP, Reply Host1-link-3 is-at 02:4a:50:7a:cd:98 (out Unknown), length 28
^C
17 packets captured
17 packets received by filter
0 packets dropped by kernel
asarabi3@host2:~$

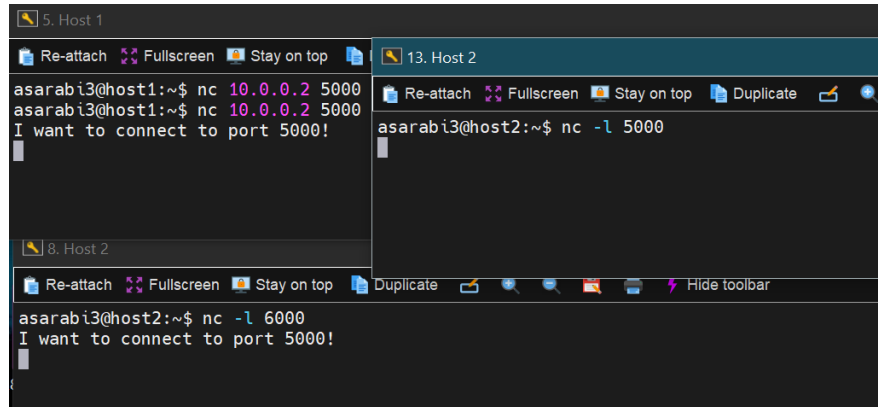
6. Host 3
17:11:51.573748 IP Host1-link-3 > Host2-link-4: ICMP echo request, id 5691, seq 55, length 64
^C
19 packets captured
19 packets received by filter
0 packets dropped by kernel
asarabi3@host3:~$ sudo tcpdump -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
17:14:33.487356 IP Host1-link-3 > Host2-link-4: ICMP echo request, id 5740, seq 1, length 64
17:14:34.488301 IP Host1-link-3 > Host2-link-4: ICMP echo request, id 5740, seq 2, length 64
17:14:35.489999 IP Host1-link-3 > Host2-link-4: ICMP echo request, id 5740, seq 3, length 64
17:14:36.491624 IP Host1-link-3 > Host2-link-4: ICMP echo request, id 5740, seq 4, length 64
17:14:37.493059 IP Host1-link-3 > Host2-link-4: ICMP echo request, id 5740, seq 5, length 64
17:14:38.494443 IP Host1-link-3 > Host2-link-4: ICMP echo request, id 5740, seq 6, length 64
^C
6 packets captured
6 packets received by filter
0 packets dropped by kernel
asarabi3@host3:~$
```

Host 1 pings Host 2 but we can see the duplicated traffic on host 3 as well.

Exercise 1.5:

1. Use the information collected in Step 3e in Step 3g (Run a Port Forward Controller).
2. Take a screen shots for your lab report in step 3 showing the **netcat** output before rule for the port forwarding is inserted.
3. Take another screen shot for your lab report in step 6, that illustrates the operation of the port forwarding controller, i.e., showing the **netcat** output after the port forwarding is inserted.

Provide a brief explanation of what is observed in each screen shot.



```
curl -X POST -d '{"switch":"00:00:a2:bf:07:5b:d6:41","name":"flow-2","priority":"32768","in_port":"1","active":"true", "eth_type":"0x0800", "ip_proto":"0x06", "eth_src":"02:4a:50:7a:cd:98", "eth_dst":"02:56:68:66:59:2a", "tcp_dst":"5000", "ipv4_src":"10.0.0.1", "ipv4_dst":"10.0.0.2", "actions":{"set_field=tcp_dst->6000,output=2}}' http://localhost:8080/wm/staticflowpusher/json
```

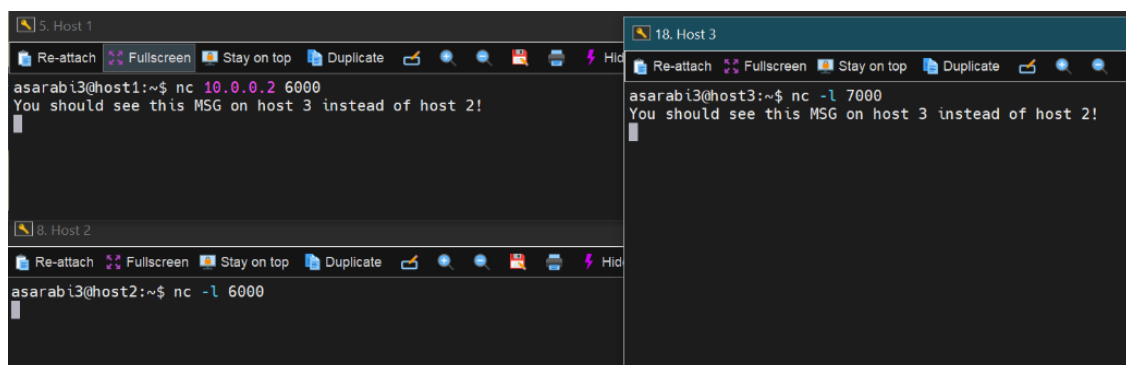
```
curl -X POST -d '{"switch":"00:00:a2:bf:07:5b:d6:41","name":"flow-3","priority":"32768","in_port":"2","active":"true", "eth_type":"0x0800", "ip_proto":"0x06", "eth_src":"02:56:68:66:59:2a", "eth_dst":"02:4a:50:7a:cd:98", "tcp_src":"6000", "ipv4_src":"10.0.0.2", "ipv4_dst":"10.0.0.1", "actions":{"set_field=tcp_src->5000,output=1}}' http://localhost:8080/wm/staticflowpusher/json
```

Exercise 1.6:

1. Use the information collected in Step 3e in Step 3h (Run a Server Proxy Controller).
2. Take a screen shots for your lab report that illustrate the operation of the proxy controller, i.e., before the traffic is diverted, and in step 6, showing the diverted traffic.

Provide a brief explanation of what is observed in the screen shots.

The traffic is directed to host3 on port 7000 instead of host2 on port 6000. Like before, each request is rewritten to direct to the new, changed port, and in this case, the destination/source IP address is also changed to make it appear to host1 that host3 is host2.

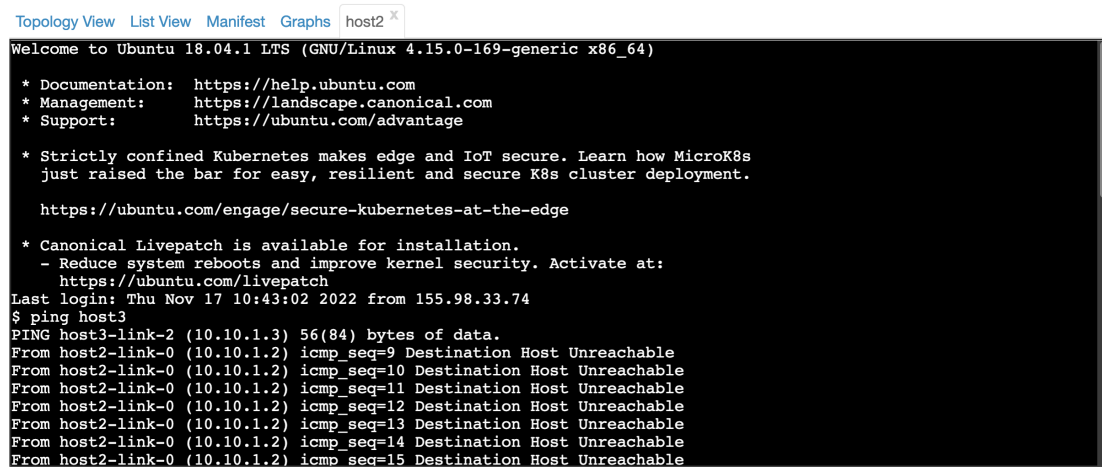


Exercise 1.7: Implement your own Denial-of-Service Controller.

1. Select one of the hosts to block, say Host2.
2. Implement a *denial-of-service controller* that drops all traffic coming from the source IP address of the host you selected. (The source IP address can be hard coded.) You should make use of the utilities provided; see step **3d. Download the pox apps in Step 3. Execute Experiment.**
3. Upload and test your controller program. It should allow traffic from Host1 and Host3 and block the traffic from Host 2.
4. Design an experiment to demonstrate the functionality of your controller, taking a screen shots for your lab report that illustrate its operation, i.e., both blocked and unblocked traffic.

Include the code for your controller program, as well as an explanation of the experiment you designed to demonstrate its correctness. Also provide a brief explanation of what is observed in the screen shots that are part of your demonstration.

Modify or create new Pox app which tells the switch to not route any packets from Host2. Codes could be different.



```
Topology View List View Manifest Graphs host2 x
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-169-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch
Last login: Thu Nov 17 10:43:02 2022 from 155.98.33.74
$ ping host3
PING host3-link-2 (10.10.1.3) 56(84) bytes of data.
From host2-link-0 (10.10.1.2) icmp_seq=9 Destination Host Unreachable
From host2-link-0 (10.10.1.2) icmp_seq=10 Destination Host Unreachable
From host2-link-0 (10.10.1.2) icmp_seq=11 Destination Host Unreachable
From host2-link-0 (10.10.1.2) icmp_seq=12 Destination Host Unreachable
From host2-link-0 (10.10.1.2) icmp_seq=13 Destination Host Unreachable
From host2-link-0 (10.10.1.2) icmp_seq=14 Destination Host Unreachable
From host2-link-0 (10.10.1.2) icmp_seq=15 Destination Host Unreachable
```

Similar to the following code is acceptable:

```
def _handle_PacketIn(self, event):
    self.packet = event.parsed
    self.event = event
    self.macLearningHandle()
    out_port = self.get_out_port()

    if packetSrcIp(self.packet, '10.10.1.2', log):
        log.debug("Got a packet from 10.10.1.2, dropping it.")
    else:
        log.debug("Forwarding packet")
        self.forward_packet([out_port])
```