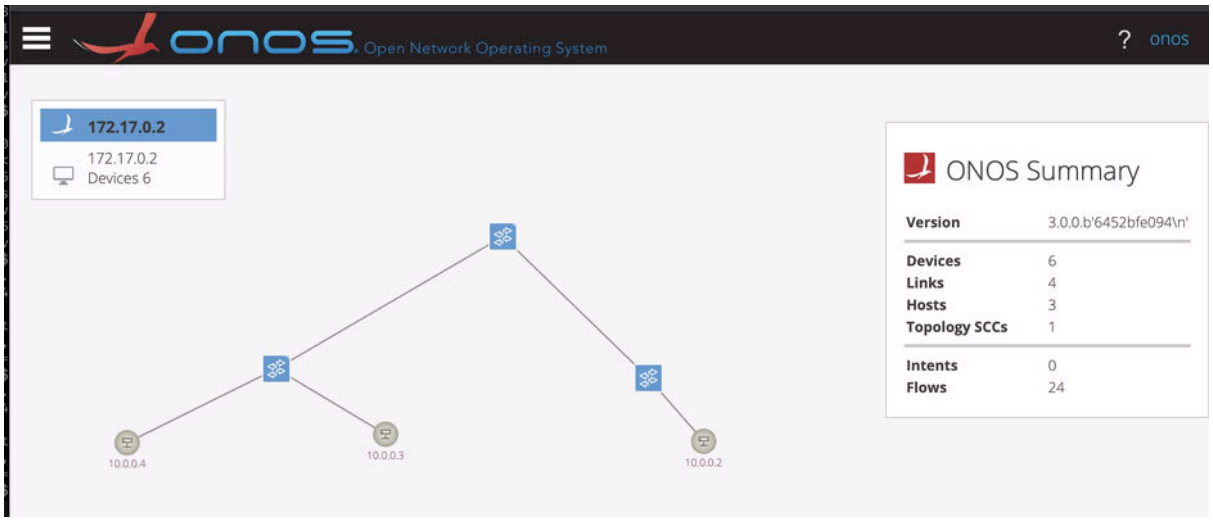
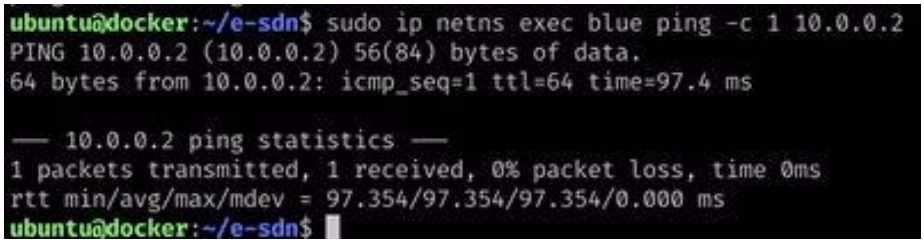
Demo 4 - OpenFlow flow tables



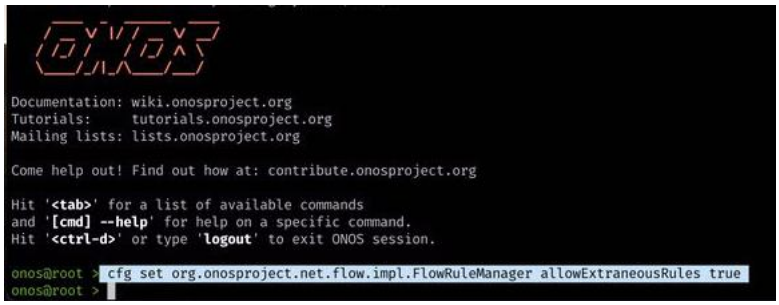
Task 4.1

First try blue to red working. And it tested by using ping and see its working as expected.

Note: Commands for the steps are included in the screenshot.

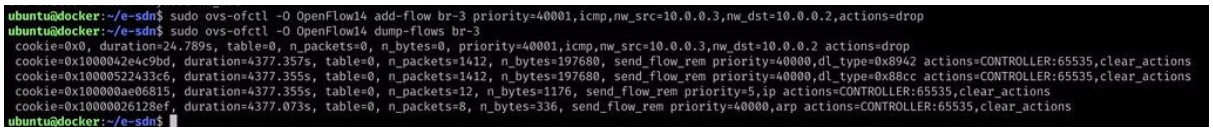


Here “allow external rules on ONOS” set to true using CLI in ONOS server.

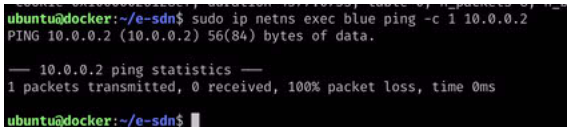


- Created the first flow rule to block the ICMP traffic from the “Blue" namespace to the “Red" namespace using CLI command and dumped all flows and checked it’s there.

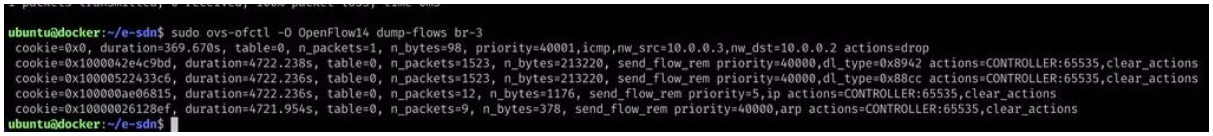
Note: Commands for the steps are included in the screenshot. For example, we used -O flag for specifying which version of OpenFlow it should use and rest are the commands for adding the rule.



Sent a ping to test and it works!

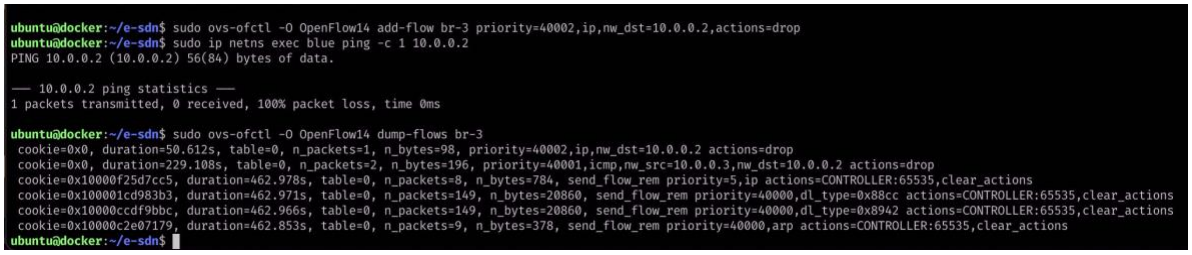


Also checked Packet 1 in the flow rule means its working as expected.



- Created the second flow rule to block the traffic to the “Red" namespace.

Note: Commands for the steps are included in the screenshot. For example, we have added the rule in first command, we tested using ping, and also checked that number of packets are there in the intended rule as expected which was lost by ping previously.



- Created the third flow rule to allow total access to the “Red" namespace from the “Green" and the “Blue" namespaces.

Note: Commands for the steps are included in the screenshot. For example, we have added the rule in first command, we tested using ping, and checked that number of packets are there in the intended rule as expected which was lost by ping previously.

```
ubuntu@docker:~/e-sdn$ sudo ovs-ofctl -O OpenFlow14 add-flow br-3 priority=40003,ip,nw_dst=10.0.0.2,actions=normal
ubuntu@docker:~/e-sdn$ sudo ip netns exec blue ping -c 1 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=71.6 ms

-- 10.0.0.2 ping statistics --
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 71.554/71.554/71.554/0.000 ms
ubuntu@docker:~/e-sdn$ sudo ip netns exec green ping -c 1 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=110 ms

-- 10.0.0.2 ping statistics --
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 110.174/110.174/110.174/0.000 ms
ubuntu@docker:~/e-sdn$ sudo ovs-ofctl -O OpenFlow14 dump-flows br-3
 cookie=0x0, duration=464.384s, table=0, n_packets=1, n_bytes=98, priority=40002,ip,nw_dst=10.0.0.2 actions=drop
 cookie=0x0, duration=45.592s, table=0, n_packets=2, n_bytes=196, priority=40003,ip,nw_dst=10.0.0.2 actions=NORMAL
 cookie=0x0, duration=642.880s, table=0, n_packets=2, n_bytes=196, priority=40001,icmp,nw_src=10.0.0.3,nw_dst=10.0.0.2 actions=drop
 cookie=0x10000f25d7cc5, duration=876.750s, table=0, n_packets=10, n_bytes=980, send_flow_rem priority=5,ip actions=CONTROLLER:65535,clear_actions
 cookie=0x100001cd983b3, duration=876.743s, table=0, n_packets=283, n_bytes=39620, send_flow_rem priority=40000,dl_type=0x88cc actions=CONTROLLER:65535,clear_actions
 cookie=0x10000ccdf9bbc, duration=876.738s, table=0, n_packets=283, n_bytes=39620, send_flow_rem priority=40000,dl_type=0x8942 actions=CONTROLLER:65535,clear_actions
 cookie=0x10000c2e07179, duration=876.625s, table=0, n_packets=11, n_bytes=462, send_flow_rem priority=40000,arp actions=CONTROLLER:65535,clear_actions
ubuntu@docker:~/e-sdn$
```

- Do you need to delete the second flow rule to activate the third flow rule?

We may NOT need to delete the previous rule instead we can set higher priority and HIGHER PRIORITY level rules will be considered first to execute.

- Create a flow rule to allow only Http and Https traffic to the “Red" namespace.

Note: Commands for the steps are included in the screenshot. For example, we have added the rules using these 3 commands.

```
sudo ovs-ofctl -O OpenFlow14 add-flow br-2 priority=40002,ip,nw_dst=10.0.0.2,tcp,tp_dst=80,actions=normal
sudo ovs-ofctl -O OpenFlow14 add-flow br-2 priority=40002,ip,nw_dst=10.0.0.2,tcp,tp_dst=443,actions=normal
sudo ovs-ofctl -O OpenFlow14 add-flow br-2 priority=40001,ip,nw_dst=10.0.0.2,actions=drop
```

Flow rules are implemented on br-2

```
ubuntu@docker:~/e-sdn$ sudo ovs-ofctl -O OpenFlow14 dump-flows br-2
 cookie=0x0, duration=178.731s, table=0, n_packets=0, n_bytes=0, priority=40002,tcp,nw_dst=10.0.0.2,tp_dst=80 actions=NORMAL
 cookie=0x0, duration=170.091s, table=0, n_packets=0, n_bytes=0, priority=40002,tcp,nw_dst=10.0.0.2,tp_dst=443 actions=NORMAL
 cookie=0x0, duration=196.507s, table=0, n_packets=1, n_bytes=98, priority=40001,ip,nw_dst=10.0.0.2 actions=drop
 cookie=0x10000779b8fce, duration=1233.532s, table=0, n_packets=794, n_bytes=111160, send_flow_rem priority=40000,dl_type=0x8942 actions=CONTROLLER:65535,clear_actions
 cookie=0x100004b23652a, duration=1233.532s, table=0, n_packets=11, n_bytes=898, send_flow_rem priority=5,ip actions=CONTROLLER:65535,clear_actions
 cookie=0x1000057ada874, duration=1233.485s, table=0, n_packets=794, n_bytes=111160, send_flow_rem priority=40000,dl_type=0x88cc actions=CONTROLLER:65535,clear_actions
 cookie=0x100003e07fafd, duration=1233.467s, table=0, n_packets=0, n_bytes=0, send_flow_rem priority=40000,arp actions=CONTROLLER:65535,clear_actions
ubuntu@docker:~/e-sdn$
```

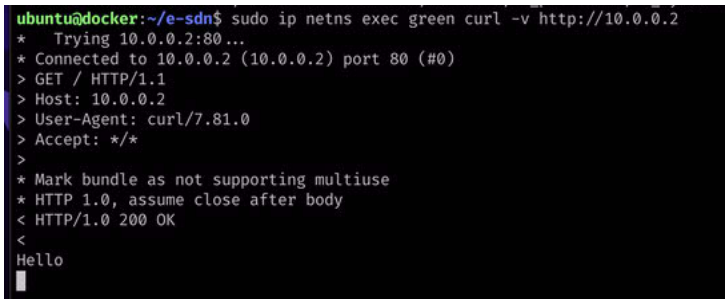
Tested ping which is blocked

```
ubuntu@docker:~$ sudo ip netns exec blue ping -c 1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

-- 10.0.0.2 ping statistics --
1 packets transmitted, 0 received, 100% packet loss, time 0ms

ubuntu@docker:~$
```

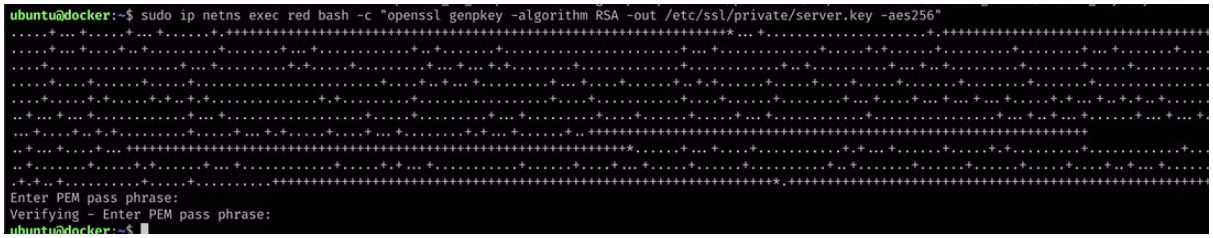
I have started a server on the host red first

```
ubuntu@docker:~$ sudo ip netns exec red bash -c "echo -ne 'HTTP/1.0 200 OK\r\n\r\nHello\r\n' | nc -l 80"
GET / HTTP/1.1
Host: 10.0.0.2
User-Agent: curl/7.81.0
Accept: */*
```

http working as expected



For https I have generated a private key on red host



Generated a self signed certificate

I have run the http server same way as previous and tested this time with https.

https connection established as expected

Task 4.2

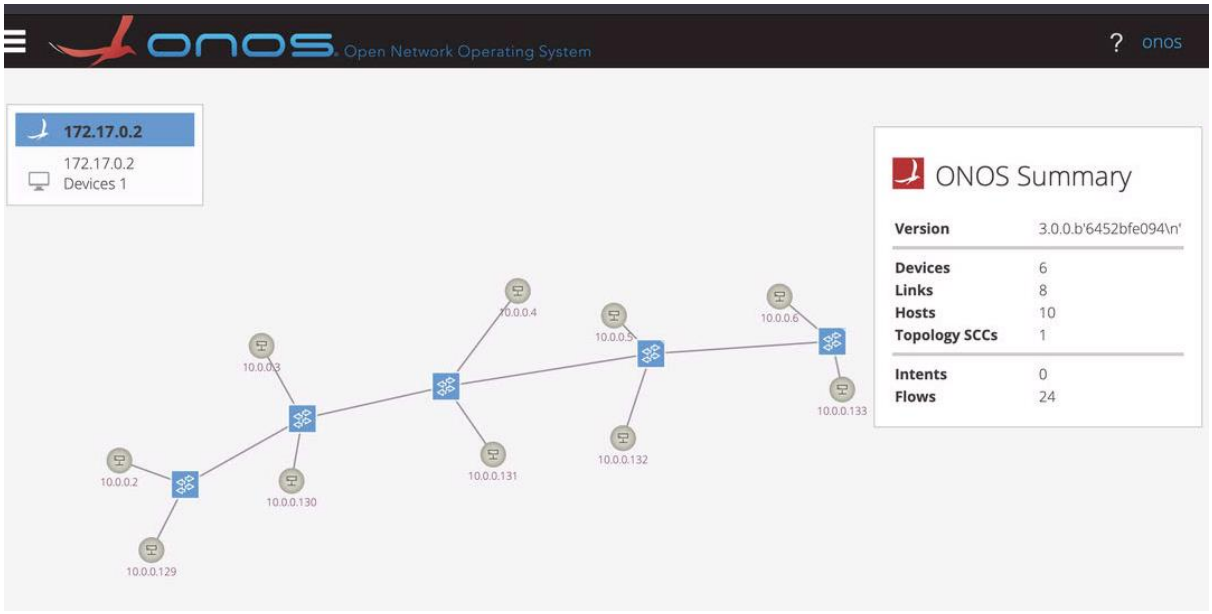
We have the linear topology using below script.

```
#!/usr/bin/env bash


function create_ns() {
echo "Creating the namespace $1"
ip netns add $1
}

function create_ovs_bridge() {
echo "Creating the OVS bridge $1"
ovs-vsctl add-br $1
```

```
ovs-vsctl set bridge $1 protocols=OpenFlow14
}


function attach_ns_to_ovs() {
echo "Attaching the namespace $1 to the OVS $2"
ip link add $3 type veth peer name $4
ip link set $3 netns $1
ovs-vsctl add-port $2 $4 -- set Interface $4 ofport_request=$5
ip netns exec $1 ip addr add $6/24 dev $3
ip netns exec $1 ip link set dev $3 up
ip link set $4 up
}


function attach_ovs_to_ovs() {
echo "Attaching the OVS $1 to the OVS $2"
ip link add name $3 type veth peer name $4
ip link set $3 up
ip link set $4 up
ovs-vsctl add-port $1 $3 -- set Interface $3 ofport_request=$5
ovs-vsctl add-port $2 $4 -- set Interface $4 ofport_request=$5
}

function attach_ovs_to_sdn() {
echo "Attaching the OVS bridge to the ONOS controller"
ovs-vsctl set-controller $1 tcp:$(docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}'
$(docker ps -q  --filter ancestor=onosproject/onos)):6653
}

create_ns red1
create_ns red2
create_ns red3
create_ns red4
create_ns red5
create_ns blue1
create_ns blue2
create_ns blue3
create_ns blue4
create_ns blue5
```

```
create_ovs_bridge br-1
create_ovs_bridge br-2
create_ovs_bridge br-3
create_ovs_bridge br-4
create_ovs_bridge br-5

attach_ovs_to_ovs br-1 br-2 br-ovs11 br-ovs21 1
attach_ovs_to_ovs br-2 br-3 br-ovs22 br-ovs31 2
attach_ovs_to_ovs br-3 br-4 br-ovs32 br-ovs41 3
attach_ovs_to_ovs br-4 br-5 br-ovs42 br-ovs51 4

attach_ns_to_ovs red1 br-1 veth-red1 veth-red1-br 5 10.0.0.2
attach_ns_to_ovs red2 br-2 veth-red2 veth-red2-br 6 10.0.0.3
attach_ns_to_ovs red3 br-3 veth-red3 veth-red3-br 7 10.0.0.4
attach_ns_to_ovs red4 br-4 veth-red4 veth-red4-br 8 10.0.0.5
attach_ns_to_ovs red5 br-5 veth-red5 veth-red5-br 9 10.0.0.6
attach_ns_to_ovs blue1 br-1 veth-blue1 veth-blue1-br 10 10.0.0.129
attach_ns_to_ovs blue2 br-2 veth-blue2 veth-blue2-br 11 10.0.0.130
attach_ns_to_ovs blue3 br-3 veth-blue3 veth-blue3-br 12 10.0.0.131
attach_ns_to_ovs blue4 br-4 veth-blue4 veth-blue4-br 13 10.0.0.132
attach_ns_to_ovs blue5 br-5 veth-blue5 veth-blue5-br 14 10.0.0.133

attach_ovs_to_sdn br-1
attach_ovs_to_sdn br-2
attach_ovs_to_sdn br-3
attach_ovs_to_sdn br-4
attach_ovs_to_sdn br-5
```

Here is the topology on ONOS GUI



Using below script, I have created rules in each switch to slice the red & blue network.

```
Users > eashin > Documents > sdn > $ red-blue.sh
1   #!/usr/bin/env bash
2   #block traffic from red to blue
3   sudo ovs-ofctl -O OpenFlow14 add-flow br-1 priority=40001,ip,nw_src=10.0.0.2/29,nw_dst=10.0.0.129/29,actions=drop
4   sudo ovs-ofctl -O OpenFlow14 add-flow br-2 priority=40001,ip,nw_src=10.0.0.2/29,nw_dst=10.0.0.129/29,actions=drop
5   sudo ovs-ofctl -O OpenFlow14 add-flow br-3 priority=40001,ip,nw_src=10.0.0.2/29,nw_dst=10.0.0.129/29,actions=drop
6   sudo ovs-ofctl -O OpenFlow14 add-flow br-4 priority=40001,ip,nw_src=10.0.0.2/29,nw_dst=10.0.0.129/29,actions=drop
7   sudo ovs-ofctl -O OpenFlow14 add-flow br-5 priority=40001,ip,nw_src=10.0.0.2/29,nw_dst=10.0.0.129/29,actions=drop
8   #block traffic from blue to red
9   sudo ovs-ofctl -O OpenFlow14 add-flow br-1 priority=40001,ip,nw_src=10.0.0.129/29,nw_dst=10.0.0.2/29,actions=drop
10  sudo ovs-ofctl -O OpenFlow14 add-flow br-2 priority=40001,ip,nw_src=10.0.0.129/29,nw_dst=10.0.0.2/29,actions=drop
11  sudo ovs-ofctl -O OpenFlow14 add-flow br-3 priority=40001,ip,nw_src=10.0.0.129/29,nw_dst=10.0.0.2/29,actions=drop
12  sudo ovs-ofctl -O OpenFlow14 add-flow br-4 priority=40001,ip,nw_src=10.0.0.129/29,nw_dst=10.0.0.2/29,actions=drop
13  sudo ovs-ofctl -O OpenFlow14 add-flow br-5 priority=40001,ip,nw_src=10.0.0.129/29,nw_dst=10.0.0.2/29,actions=drop
```

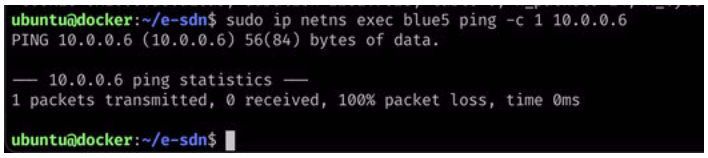
Br-1 flow dumps are below, same way it is implemented in all br-1 to br-5

```
ubuntu@docker:~/e-sdn$ sudo ovs-ofctl -O OpenFlow14 dump-flows br-1
 cookie=0x0, duration=380.807s, table=0, n_packets=1, n_bytes=98, priority=40001,ip,nw_src=10.0.0.0/29,nw_dst=10.0.0.128/29 actions=drop
 cookie=0x0, duration=380.384s, table=0, n_packets=0, n_bytes=0, priority=40001,ip,nw_src=10.0.0.128/29,nw_dst=10.0.0.0/29 actions=drop
 cookie=0x1000015fefc1e, duration=2282.120s, table=0, n_packets=20, n_bytes=1960, send_flow_rem priority=5,ip actions=CONTROLLER:65535,clear_actions
 cookie=0x1000074590665, duration=2282.120s, table=0, n_packets=735, n_bytes=102900, send_flow_rem priority=40000,dl_type=0x88cc actions=CONTROLLER:65535,clear_actions
 cookie=0x100005eb3b72d, duration=2282.120s, table=0, n_packets=735, n_bytes=102900, send_flow_rem priority=40000,dl_type=0x8942 actions=CONTROLLER:65535,clear_actions
 cookie=0x10000d80a1ea3, duration=2281.952s, table=0, n_packets=14, n_bytes=588, send_flow_rem priority=40000,arp actions=CONTROLLER:65535,clear_actions
ubuntu@docker:~/e-sdn$
```

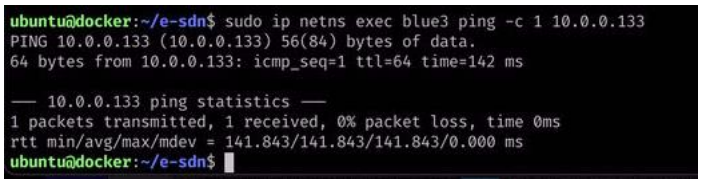
Random tests from red to blue – blocked!

```
ubuntu@docker:~/e-sdn$ sudo ip netns exec red1 ping -c 1 10.0.0.129
PING 10.0.0.129 (10.0.0.129) 56(84) bytes of data.

— 10.0.0.129 ping statistics —
1 packets transmitted, 0 received, 100% packet loss, time 0ms

ubuntu@docker:~/e-sdn$
buntu@docker:~/e-sdn$ sudo ip netns exec red5 ping -c 1 10.0.0.130
ING 10.0.0.130 (10.0.0.130) 56(84) bytes of data.

— 10.0.0.130 ping statistics —
 packets transmitted, 0 received, 100% packet loss, time 0ms

buntu@docker:~/e-sdn$
```

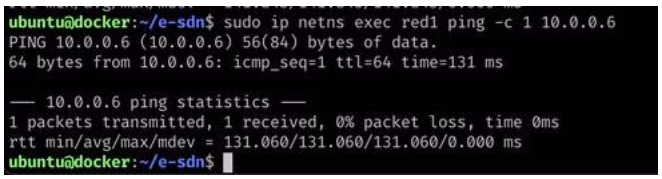
Random tests from blue to red - blocked!



Random test from blue to blue – not blocked!



Random test from red to red – not blocked!



Note: I have tested each host to other host to check that slicing works, I just added some here (but not all).