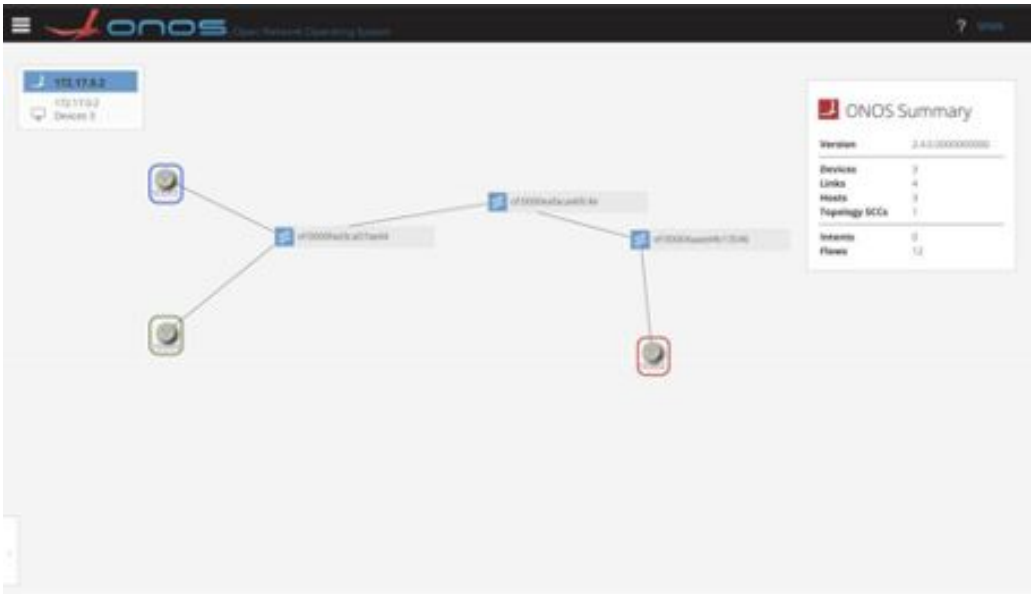
SDN Fundamentals & Techniques

Chapter 4 - Demo 3 - OpenFlow flow tables

Jing Yan (yanj3, jing.yan@aalto.fi) Feb 18th, 2020

Task 3.1

The file related to Task 3.1 is stored in "task1" at http://www.mosaic-lab.org/SDN_Demos.aspx under "Demo 3 - ONOS and Linux Containers." This shell script creates a topology as depicted in Fig. 5. A host in the ONOS's GUI is a namespace, however, LXC or LXD containers can be used. In this task you are asked to:



3.1.1 Create the first flow rule to block the ICMP traffic from the "Blue" namespace to the "Red" namespace

CMD to set OVS rules

```
# ovs-ofctl add-flow br-3
"priority=11,icmp,nw_src=10.0.0.3,nw_dst=10.0.0.2,action=drop"
```

Results

```
# ip netns exec blue ping -c1 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=5.28 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 5.280/5.280/5.280/0.000 ms
```

```
# ovs-ofctl add-flow br-3
"priority=11,icmp,nw_src=10.0.0.3,nw_dst=10.0.0.2,action=drop"
root@elxa9698s73:/home/ejinyna# ovs-ofctl dump-flows br-3 | grep drop
cookie=0x0, duration=9.208s, table=0, n_packets=0, n_bytes=0, idle_age=9,
icmp,nw_src=10.0.0.3,nw_dst=10.0.0.2 actions=drop

# ip netns exec blue ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^C
--- 10.0.0.2 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

# ovs-ofctl dump-flows br-3 | grep drop
cookie=0x0, duration=56.158s, table=0, n_packets=1, n_bytes=98,
priority=11,icmp,nw_src=10.0.0.3,nw_dst=10.0.0.2 actions=drop
```

Explanation

In the “ovs-ofctl” cmd, “icmp” means “icmp” packages, and “nw_src=10.0.0.3” means the source IP, namely “blue” namespace’s IP, and “nw_dst=10.0.0.2” means the destination IP, namely “red” namespace’s IP, and configure the matching rules on “br-3” OVS switch. Before configuring this cmd, “blue” namespace can ping “red” namespace successfully, after configuring this cmd, ping failed and we can even see the blocked packets and blocked bytes by this rule.

3.1.2 Create the second flow rule to block the traffic to the “Red" namespace

CMD to set OVS rules

```
# ovs-ofctl add-flow br-1 "priority=11,arp,nw_dst=10.0.0.2 actions=drop"
# ovs-ofctl add-flow br-1 "priority=11,ip,nw_dst=10.0.0.2 actions=drop"
```

Results

```
# ip netns exec green ping -c1 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=5.15 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 5.155/5.155/5.155/0.000 ms


# ovs-ofctl add-flow br-1 "priority=11,arp,nw_dst=10.0.0.2 actions=drop"
# ovs-ofctl add-flow br-1 "priority=11,ip,nw_dst=10.0.0.2 actions=drop"

# ip netns exec green ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^C
--- 10.0.0.2 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms


# ovs-ofctl dump-flows br-1 | grep drop
cookie=0x0, duration=24.271s, table=0, n_packets=0, n_bytes=0, idle_age=24,
priority=11,arp,arp_tpa=10.0.0.2 actions=drop
cookie=0x0, duration=17.343s, table=0, n_packets=1, n_bytes=98, idle_age=12,
priority=11,ip,nw_dst=10.0.0.2 actions=drop
```

Explanation

I added two rules here to block both L2 packets and L3 packets with destination IP equal “10.0.0.2”, namely red namespace’s IP on br-1 OVS switch.
And ping from green namespace to red namespace can succeed before this rule and then fail after this rule, we can also see the blocked packets and bytes by this cmd.

3.1.3 Create the third flow rule to allow total access to the “Red" namespace from the “Green" and the “Blue" namespaces

CMD to set OVS rules

```
# ovs-ofctl add-flow br-1 "priority=12,arp,nw_dst=10.0.0.2,actions=normal"
# ovs-ofctl add-flow br-1 "priority=12,ip,nw_dst=10.0.0.2,actions=normal"
# ovs-ofctl add-flow br-3
"priority=12,icmp,nw_src=10.0.0.3,nw_dst=10.0.0.2,action=normal"
```

Results

```
# ip netns exec green ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^C
--- 10.0.0.2 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

# ip netns exec blue ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^C
--- 10.0.0.2 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

# ovs-ofctl add-flow br-1 "priority=12,arp,nw_dst=10.0.0.2,actions=normal"
# ovs-ofctl add-flow br-1 "priority=12,ip,nw_dst=10.0.0.2,actions=normal"
# ovs-ofctl add-flow br-3
"priority=12,icmp,nw_src=10.0.0.3,nw_dst=10.0.0.2,action=normal"

# ip netns exec blue ping -c1 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=9.50 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 9.507/9.507/9.507/0.000 ms

# ip netns exec green ping -c1 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=11.3 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 11.302/11.302/11.302/0.000 ms

# ovs-ofctl dump-flows br-3
cookie=0x0, duration=11.957s, table=0, n_packets=1, n_bytes=98,
priority=12,icmp,nw_src=10.0.0.3,nw_dst=10.0.0.2 actions=NORMAL
# ovs-ofctl dump-flows br-1
cookie=0x0, duration=28.085s, table=0, n_packets=1, n_bytes=98,
priority=12,ip,nw_dst=10.0.0.2 actions=NORMAL
```

Explanation

Normally, all the packets should be forward. And because of the previous rules in 3.1.1 and 3.1.2, all the packets no matter from blue namespace or green namespace to red namespace are blocked. After adding the rule with “action=normal” and with higher priority, then all the packets can be forward to red namespace normally now.
And before configuring these three rules, either green namespace or blue namespace can not ping red namespace, after configuring these three rules, ping from green namespace and blue namespace to red namespace can succeed now and we can also see the matching packets and bytes by the rules.

3.1.4 Do you need to delete the second flow rule to activate the third flow rule

No need.
Just set higher priority for the “action=normal” rules than the “action=drop” rules as 3.1.3 shows.

3.1.5 Create a flow rule to allow only HTTP and HTTPS traffic to the “Red" namespace

CMD to set OVS rules

In this question, I deleted all the rules before and only added the following 2 rules to make the flows to red namespace blocked.

```
# ovs-ofctl add-flow br-1 "priority=11,arp,nw_dst=10.0.0.2 actions=drop"
# ovs-ofctl add-flow br-1 "priority=11,ip,nw_dst=10.0.0.2 actions=drop"
```

And adding the following two rules to allow http/https flows

```
# ovs-ofctl add-flow br-1 "priority=12,tcp,nw_dst=10.0.0.2
actions=mod_tp_dst:443,normal"
# ovs-ofctl add-flow br-1 "priority=12,tcp,nw_dst=10.0.0.2
actions=mod_tp_dst:80,normal"
```

Results

Before configuring the rules to allow http/https,

```
# ip netns exec blue telnet 10.0.0.2 80
Trying 10.0.0.2...
^C

# ovs-ofctl dump-flows br-1
cookie=0x0, duration=10.159s, table=0, n_packets=1, n_bytes=74,
priority=11,ip,nw_dst=10.0.0.2 actions=drop

# ip netns exec blue telnet 10.0.0.2 443
Trying 10.0.0.2...
^C

# ovs-ofctl dump-flows br-1
cookie=0x0, duration=16.135s, table=0, n_packets=2, n_bytes=148,
priority=11,ip,nw_dst=10.0.0.2 actions=drop
```

After configuring the rules to allow http/https,

```
# ip netns exec blue telnet 10.0.0.2 443
```

```
Trying 10.0.0.2...
telnet: Unable to connect to remote host: Connection refused

# ovs-ofctl dump-flows br-1
cookie=0x0, duration=5s, table=0, n_packets=1, n_bytes=74,
priority=12,tcp,nw_dst=10.0.0.2 actions=mod_tp_dst:443,NORMAL

# ip netns exec blue telnet 10.0.0.2 80
Trying 10.0.0.2...
telnet: Unable to connect to remote host: Connection refused

# ovs-ofctl dump-flows br-1
cookie=0x0, duration=80.515s, table=0, n_packets=1, n_bytes=74,
priority=12,tcp,nw_dst=10.0.0.2 actions=mod_tp_dst:80,NORMAL
```

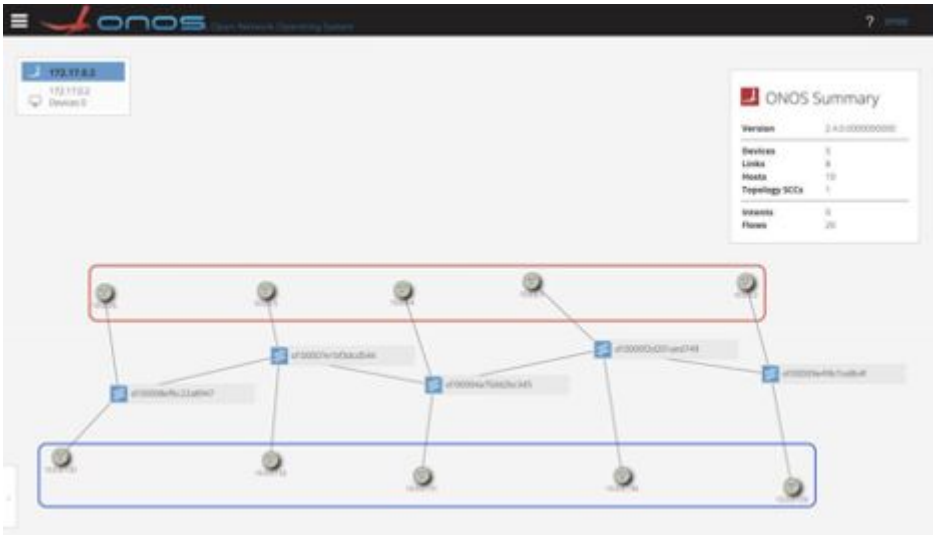
Explanation

In this case, all the flows including L2 and L3 are blocked by the first two rules with priority equals 11, and then I configured another rules to allow the flows with destination port equals 80 and 443, and destination ip equals 10.0.0.2, with action equals normal and with high priority(priority equals 12).

Note that for each flow rule, the student is asked to provide details on the entered rule and explanations. Use only the OVS command line to enter the flow rules.

Task 3.2

The objective of this task is to create a pseudo Network Slice based on namespaces and OVSs.



3.2.1 Create a linear topology of 5 switches and 10 hosts (as in Fig. 6). In a linear topology, each switch has two connections with other switches except the first and the last ones. Each switch contains two hosts, one for the “Red" slice and one for the “Blue" slice

Shell Scripts

```
#!/usr/bin/env bash
create_ovs_bridge() {
   echo "Creating the OVS bridge $1"
   ovs-vsctl add-br $1
}
create_ns() {
   echo "Creating the namespace $1"
   ip netns add $1
}
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
}
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
}
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_ovs_bridge ovs1
create_ovs_bridge ovs2
create_ovs_bridge ovs3
create_ovs_bridge ovs4
create_ovs_bridge ovs5
create_ns ovs1_red
create_ns ovs2_red
create_ns ovs3_red
create_ns ovs4_red
create_ns ovs5_red
create_ns ovs1_blue
create_ns ovs2_blue
create_ns ovs3_blue
create_ns ovs4_blue
create_ns ovs5_blue
attach_ovs_to_ovs ovs1 ovs2 br_ovs12 br_ovs21 1
attach_ovs_to_ovs ovs2 ovs3 br_ovs23 br_ovs32 2
attach_ovs_to_ovs ovs3 ovs4 br_ovs34 br_ovs43 3
attach_ovs_to_ovs ovs4 ovs5 br_ovs45 br_ovs54 4
attach_ns_to_ovs ovs1_red ovs1 v-ovs1-red v-ovs1-red-br 5 10.0.0.10
attach_ns_to_ovs ovs2_red ovs2 v-ovs2-red v-ovs2-red-br 6 10.0.0.11
attach_ns_to_ovs ovs3_red ovs3 v-ovs3-red v-ovs3-red-br 7 10.0.0.12
```

```
attach_ns_to_ovs ovs4_red ovs4 v-ovs4-red v-ovs4-red-br 8 10.0.0.13
attach_ns_to_ovs ovs5_red ovs5 v-ovs5-red v-ovs5-red-br 9 10.0.0.14
attach_ns_to_ovs ovs1_blue ovs1 v-ovs1-blue v-ovs1-blue-br 10 10.0.0.20
attach_ns_to_ovs ovs2_blue ovs2 v-ovs2-blue v-ovs2-blue-br 11 10.0.0.21
attach_ns_to_ovs ovs3_blue ovs3 v-ovs3-blue v-ovs3-blue-br 12 10.0.0.22
attach_ns_to_ovs ovs4_blue ovs4 v-ovs4-blue v-ovs4-blue-br 13 10.0.0.23
attach_ns_to_ovs ovs5_blue ovs5 v-ovs5-blue v-ovs5-blue-br 14 10.0.0.24
attach_ovs_to_sdn ovs1
attach_ovs_to_sdn ovs2
attach_ovs_to_sdn ovs3
attach_ovs_to_sdn ovs4
attach_ovs_to_sdn ovs5
```

Result for network test

```
# for i in 10.0.0.11 10.0.0.12 10.0.0.13 10.0.0.14; do ip netns exec ovs1_red
ping -c1 $i; done
PING 10.0.0.11 (10.0.0.11) 56(84) bytes of data.
64 bytes from 10.0.0.11: icmp_seq=1 ttl=64 time=0.884 ms

--- 10.0.0.11 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.884/0.884/0.884/0.000 ms
PING 10.0.0.12 (10.0.0.12) 56(84) bytes of data.
64 bytes from 10.0.0.12: icmp_seq=1 ttl=64 time=1.35 ms

--- 10.0.0.12 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.351/1.351/1.351/0.000 ms
PING 10.0.0.13 (10.0.0.13) 56(84) bytes of data.
64 bytes from 10.0.0.13: icmp_seq=1 ttl=64 time=1.06 ms

--- 10.0.0.13 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.060/1.060/1.060/0.000 ms
PING 10.0.0.14 (10.0.0.14) 56(84) bytes of data.
64 bytes from 10.0.0.14: icmp_seq=1 ttl=64 time=0.579 ms

--- 10.0.0.14 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.579/0.579/0.579/0.000 ms

# for i in 10.0.0.21 10.0.0.22 10.0.0.23 10.0.0.24; do ip netns exec ovs1_red
ping -c1 $i; done
PING 10.0.0.21 (10.0.0.21) 56(84) bytes of data.
64 bytes from 10.0.0.21: icmp_seq=1 ttl=64 time=0.794 ms

--- 10.0.0.21 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.794/0.794/0.794/0.000 ms
PING 10.0.0.22 (10.0.0.22) 56(84) bytes of data.
64 bytes from 10.0.0.22: icmp_seq=1 ttl=64 time=0.890 ms

--- 10.0.0.22 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.890/0.890/0.890/0.000 ms
PING 10.0.0.23 (10.0.0.23) 56(84) bytes of data.
64 bytes from 10.0.0.23: icmp_seq=1 ttl=64 time=0.531 ms

--- 10.0.0.23 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.531/0.531/0.531/0.000 ms
PING 10.0.0.24 (10.0.0.24) 56(84) bytes of data.
64 bytes from 10.0.0.24: icmp_seq=1 ttl=64 time=0.522 ms
```

```
--- 10.0.0.24 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.522/0.522/0.522/0.000 ms
```

3.2.2 After creating this topology, you are asked to create two slices, “Red" and “Blue", each slice is totally separated from the other without any toleration for shared access between them. To do that, you may write a shell script or a python-based code, your code must be well-commented and follow coding principles and naming conventions. Your solution may leverage OVS’s CLI to create the necessary flow rules. Note that all hosts are in the same network range

CMD to set OVS rules

```
# for j in ovs1 ovs2 ovs3 ovs4 ovs5; do for i in 10.0.0.20 10.0.0.21
10.0.0.22 10.0.0.23 10.0.0.24; do ovs-ofctl add-flow $j
"arp,nw_src=10.0.0.10,nw_dst=$i,actions=drop"; ovs-ofctl add-flow $j
"ip,nw_src=10.0.0.10,nw_dst=10.0.0.20,actions=drop"; ovs-ofctl add-flow $j
"arp,nw_src=$i,nw_dst=10.0.0.10,actions=drop"; ovs-ofctl add-flow $j
"ip,nw_src=$i,nw_dst=10.0.0.10,actions=drop"; done; done
```

```
root@elxa9698s73:/home/ejinyna/Desktop/SDN# for j in ovs1 ovs2 ovs3 ovs4 ovs5
> do
> for i in 10.0.0.20 10.0.0.21 10.0.0.22 10.0.0.23 10.0.0.24
> do
> ovs-ofctl add-flow $j "arp,nw_src=10.0.0.10,nw_dst=$i,actions=drop"
> ovs-ofctl add-flow $j "ip,nw_src=10.0.0.10,nw_dst=10.0.0.20,actions=drop"
> ovs-ofctl add-flow $j "arp,nw_src=$i,nw_dst=10.0.0.10,actions=drop"
> ovs-ofctl add-flow $j "ip,nw_src=$i,nw_dst=10.0.0.10,actions=drop"
> done
> done
```

Results

Before the configuration,

```
root@elxa9698s73:/home/ejinyna/Desktop/SDN# for i in 10.0.0.20 10.0.0.21 10.0.0.22 10.0.0.23 10.0.0.24
> do
> ip netns exec ovs1_red ping -c1 $i
> done
PING 10.0.0.20 (10.0.0.20) 56(84) bytes of data.
64 bytes from 10.0.0.20: icmp_seq=1 ttl=64 time=3.03 ms

--- 10.0.0.20 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 3.039/3.039/3.039/0.000 ms
PING 10.0.0.21 (10.0.0.21) 56(84) bytes of data.
64 bytes from 10.0.0.21: icmp_seq=1 ttl=64 time=6.29 ms

--- 10.0.0.21 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 6.293/6.293/6.293/0.000 ms
PING 10.0.0.22 (10.0.0.22) 56(84) bytes of data.
64 bytes from 10.0.0.22: icmp_seq=1 ttl=64 time=7.12 ms

--- 10.0.0.22 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 7.120/7.120/7.120/0.000 ms
PING 10.0.0.23 (10.0.0.23) 56(84) bytes of data.
64 bytes from 10.0.0.23: icmp_seq=1 ttl=64 time=8.18 ms

--- 10.0.0.23 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 8.187/8.187/8.187/0.000 ms
PING 10.0.0.24 (10.0.0.24) 56(84) bytes of data.
64 bytes from 10.0.0.24: icmp_seq=1 ttl=64 time=7.65 ms

--- 10.0.0.24 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 7.659/7.659/7.659/0.000 ms
```

```
root@elxa9698s73:/home/ejinyna/Desktop/SDN# for i in 10.0.0.11 10.0.0.12 10.0.0.13 10.0.0.14
> do
> ip netns exec ovs1_red ping -c1 $i
> done
PING 10.0.0.11 (10.0.0.11) 56(84) bytes of data.
64 bytes from 10.0.0.11: icmp_seq=1 ttl=64 time=14.9 ms

--- 10.0.0.11 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 14.977/14.977/14.977/0.000 ms
PING 10.0.0.12 (10.0.0.12) 56(84) bytes of data.
64 bytes from 10.0.0.12: icmp_seq=1 ttl=64 time=19.1 ms

--- 10.0.0.12 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 19.178/19.178/19.178/0.000 ms
PING 10.0.0.13 (10.0.0.13) 56(84) bytes of data.
64 bytes from 10.0.0.13: icmp_seq=1 ttl=64 time=10.3 ms

--- 10.0.0.13 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 10.369/10.369/10.369/0.000 ms
PING 10.0.0.14 (10.0.0.14) 56(84) bytes of data.
64 bytes from 10.0.0.14: icmp_seq=1 ttl=64 time=6.78 ms

--- 10.0.0.14 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 6.781/6.781/6.781/0.000 ms
```

After the configuration,

```
root@elxa9698s73:/home/ejinyna/Desktop/SDN# for i in 10.0.0.20 10.0.0.21 10.0.0.22 10.0.0.23 10.0.0.24
> do
> ip netns exec ovs1_red ping -c1 $i
> done
PING 10.0.0.20 (10.0.0.20) 56(84) bytes of data.
^C
--- 10.0.0.20 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

PING 10.0.0.21 (10.0.0.21) 56(84) bytes of data.
^C
--- 10.0.0.21 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

PING 10.0.0.22 (10.0.0.22) 56(84) bytes of data.
^C
--- 10.0.0.22 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

PING 10.0.0.23 (10.0.0.23) 56(84) bytes of data.
^C
--- 10.0.0.23 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

PING 10.0.0.24 (10.0.0.24) 56(84) bytes of data.
^C
--- 10.0.0.24 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

```
# ovs-ofctl dump-flows ovs1 | grep "n_packets=1" | grep drop
cookie=0x0, duration=160.570s, table=0, n_packets=1, n_bytes=98, idle_age=58,
ip,nw_src=10.0.0.10,nw_dst=10.0.0.20 actions=drop
# ovs-ofctl dump-flows ovs2 | grep "n_packets=1" | grep drop
cookie=0x0, duration=163.711s, table=0, n_packets=1, n_bytes=98, idle_age=60,
ip,nw_src=10.0.0.21,nw_dst=10.0.0.10 actions=drop
# ovs-ofctl dump-flows ovs3 | grep "n_packets=1" | grep drop
cookie=0x0, duration=168.034s, table=0, n_packets=1, n_bytes=98, idle_age=64,
ip,nw_src=10.0.0.22,nw_dst=10.0.0.10 actions=drop
# ovs-ofctl dump-flows ovs4 | grep "n_packets=1" | grep drop
cookie=0x0, duration=170.875s, table=0, n_packets=1, n_bytes=98, idle_age=67,
ip,nw_src=10.0.0.23,nw_dst=10.0.0.10 actions=drop
# ovs-ofctl dump-flows ovs5 | grep "n_packets=1" | grep drop
cookie=0x0, duration=173.814s, table=0, n_packets=1, n_bytes=98, idle_age=70,
ip,nw_src=10.0.0.24,nw_dst=10.0.0.10 actions=drop
```

Explanation

Set “10.0.0.10-14” for the 5 host namespaces in the red slice
Set “10.0.0.20-24” for the 5 host namespaces in the blue slice
For the configuration, I configured on every switch, to block both IP(L2) and arp (L3) packets when the src is any of “10.0.0.10-14” and the destination is any of “10.0.0.20-24”, and when the destination is any of “10.0.0.10-14” and the src is any of “10.0.0.20-24”.
As we can see from the result, before configuration, the ping from ovs1_red to any other namespace can succeed, after the configuration, ovs1_red can only ping hosts in the red slice, and when pinging any host in the blue slice, it will be blocked and shown the detailed blocked packets and bytes in the rules.
Test for any other case also succeeds in the way “ovs1_red” behaves.

3.2.3 Redo the isolation using another approach that differs from the one you have used in the previous question

Solution

For this part, I re-write the deployment script for OVS and namespaces as follows to add vlan tags.

```
#!/usr/bin/env bash
create_ovs_bridge() {
   echo "Creating the OVS bridge $1"
   ovs-vsctl add-br $1
}
create_ns() {
   echo "Creating the namespace $1"
   ip netns add $1
}
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
}
attach_ns_to_ovs_red() {
   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 tag=10 -- 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
}
attach_ns_to_ovs_blue() {
   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 tag=20 -- 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
}
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_ovs_bridge ovs1
create_ovs_bridge ovs2
create_ovs_bridge ovs3
create_ovs_bridge ovs4
create_ovs_bridge ovs5
create_ns ovs1_red
create_ns ovs2_red
create_ns ovs3_red
create_ns ovs4_red
create_ns ovs5_red
create_ns ovs1_blue
create_ns ovs2_blue
create_ns ovs3_blue
create_ns ovs4_blue
```

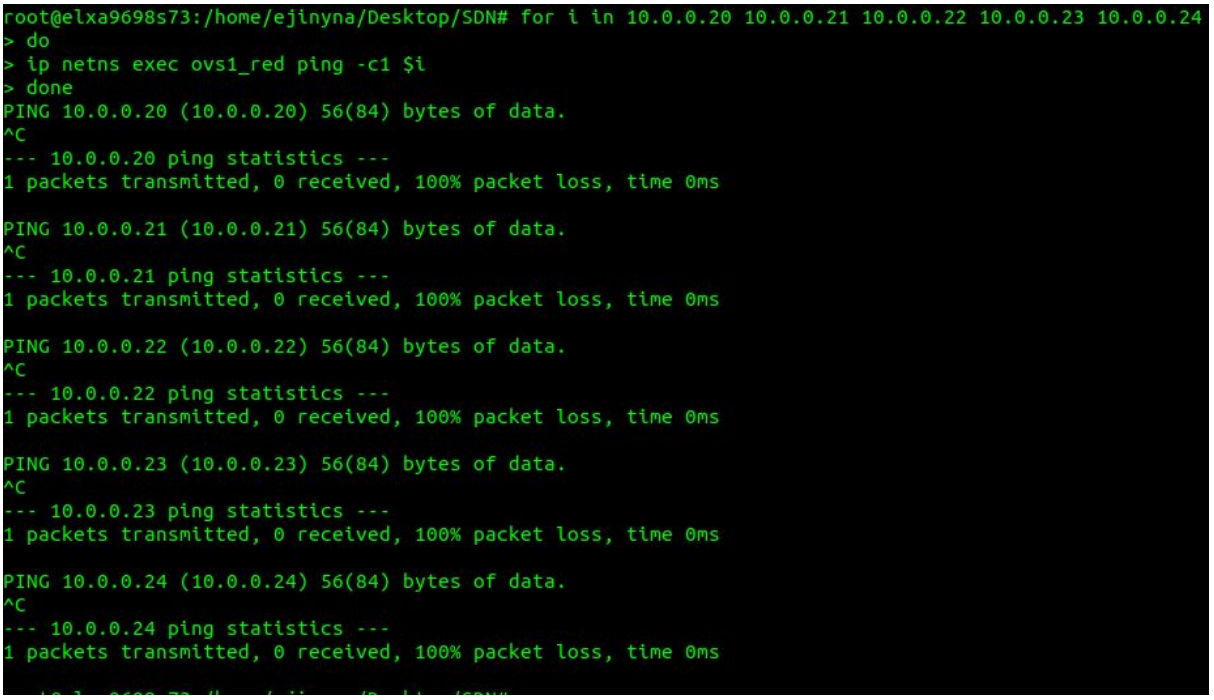
```
create_ns ovs5_blue
attach_ovs_to_ovs ovs1 ovs2 br_ovs12 br_ovs21 1
attach_ovs_to_ovs ovs2 ovs3 br_ovs23 br_ovs32 2
attach_ovs_to_ovs ovs3 ovs4 br_ovs34 br_ovs43 3
attach_ovs_to_ovs ovs4 ovs5 br_ovs45 br_ovs54 4
attach_ns_to_ovs_red ovs1_red ovs1 v-ovs1-red v-ovs1-red-br 5 10.0.0.10
attach_ns_to_ovs_red ovs2_red ovs2 v-ovs2-red v-ovs2-red-br 6 10.0.0.11
attach_ns_to_ovs_red ovs3_red ovs3 v-ovs3-red v-ovs3-red-br 7 10.0.0.12
attach_ns_to_ovs_red ovs4_red ovs4 v-ovs4-red v-ovs4-red-br 8 10.0.0.13
attach_ns_to_ovs_red ovs5_red ovs5 v-ovs5-red v-ovs5-red-br 9 10.0.0.14
attach_ns_to_ovs_blue ovs1_blue ovs1 v-ovs1-blue v-ovs1-blue-br 10 10.0.0.20
attach_ns_to_ovs_blue ovs2_blue ovs2 v-ovs2-blue v-ovs2-blue-br 11 10.0.0.21
attach_ns_to_ovs_blue ovs3_blue ovs3 v-ovs3-blue v-ovs3-blue-br 12 10.0.0.22
attach_ns_to_ovs_blue ovs4_blue ovs4 v-ovs4-blue v-ovs4-blue-br 13 10.0.0.23
attach_ns_to_ovs_blue ovs5_blue ovs5 v-ovs5-blue v-ovs5-blue-br 14 10.0.0.24
attach_ovs_to_sdn ovs1
attach_ovs_to_sdn ovs2
attach_ovs_to_sdn ovs3
attach_ovs_to_sdn ovs4
attach_ovs_to_sdn ovs5
```

Results

ovs1-red can ping other namespaces in red slice successfully. Also tried other four namespaces in red slice, all works well.

```
root@elxa9698s73:/home/ejinyna/Desktop/SDN# for i in 10.0.0.11 10.0.0.12 10.0.0.13 10.0.0.14
> do
> ip netns exec ovs1_red ping -c1 $i
> done
PING 10.0.0.11 (10.0.0.11) 56(84) bytes of data.
64 bytes from 10.0.0.11: icmp_seq=1 ttl=64 time=0.741 ms

--- 10.0.0.11 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.741/0.741/0.741/0.000 ms
PING 10.0.0.12 (10.0.0.12) 56(84) bytes of data.
64 bytes from 10.0.0.12: icmp_seq=1 ttl=64 time=2.54 ms

--- 10.0.0.12 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2.544/2.544/2.544/0.000 ms
PING 10.0.0.13 (10.0.0.13) 56(84) bytes of data.
64 bytes from 10.0.0.13: icmp_seq=1 ttl=64 time=0.914 ms

--- 10.0.0.13 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.914/0.914/0.914/0.000 ms
PING 10.0.0.14 (10.0.0.14) 56(84) bytes of data.
64 bytes from 10.0.0.14: icmp_seq=1 ttl=64 time=0.998 ms

--- 10.0.0.14 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.998/0.998/0.998/0.000 ms
```

ovs1-red can not ping other namespaces in red slice. Also tried other four namespaces in red slice, all works as expected.



Explanation

For all the ports of OVS switches connected to namespaces in red slice, they are all tagged with vlanid equals 10. For all the ports of OVS switches connected to namespaces in blue slice, they are all tagged with vlanid equals 20. So all 10 namespaces are separated by vlan into 2 subnetworks.

Explore the usage of VLANs, this may help to solve one of the questions.