# ELEC-E7311 - SDN Fundamentals & Techniques

## Demo 4: OpenFlow Flow Tables

**Md Shawkot Hossain**

*Submission date: March 25, 2024*

### Task 4.1

**Rule 1:** Blocking ICMP traffic from the “Blue" namespace to the “Red" namespace.

// this rule will drop any icmp packet which source is 10.0.0.3 and destination is 10.0.0.2. With priority, we can set which rule should be implemented first.

***sudo ovs-ofctl -O OpenFlow13 add-flow br-3 priority=100,ip,nw_src=10.0.0.3,nw_dst=10.0.0.2,icmp,icmp_type=8,actions=drop***

**Rule 2:** Blocking all traffic to “Red” namespace

// this rule will drop any packet that comes to br-1 and destined for 10.0.0.2 i.e. the red namespace.

***sudo ovs-ofctl -O OpenFlow13 add-flow br-1 priority=100,ip,nw_dst=10.0.0.2,actions=drop***

**Rule 3:** Allow all traffic from “Blue” and “Green” namespaces to “Red” namespace

// the following two rules will accept traffic from 10.0.0.3/4 and redirect the traffic towards port ‘veth-red-br' i.e. to the red namespace

***sudo ovs-ofctl -O OpenFlow13 add-flow br-1 priority=200,ip,nw_dst=10.0.0.2,nw_src=10.0.0.3,actions=output:veth-red-br***

***sudo ovs-ofctl -O OpenFlow13 add-flow br-1 priority=200,ip,nw_dst=10.0.0.2,nw_src=10.0.0.4,actions=output:veth-red-br***

“Rule 3” can be worked without deleting rule 2, because rule is blcoking all traffic in general irrespective of of source ip or port. But with rule 3, it allows traffic from “Blue” and “Green” namespace by using the port.

**Rule 4:** Allow only HTTP and HTTPS traffic to the "Red" namespace:

// the following two rules will allow traffic that is destined to 10.0.0.2 hosts and the destination port should be 80 and 443.

***ovs-ofctl -O OpenFlow13 add-flow br-1 priority=150,ip,nw_dst=10.0.0.2,tcp,tp_dst=80,actions=output:veth-red-br***

***ovs-ofctl -O OpenFlow13 add-flow br-1 priority=150,ip,nw_dst=10.0.0.2,tcp,tp_dst=443,actions=output:veth-red-br***

## Task 4.2

A linear topology with 5 switches and 10 hosts. Each switch is connected to two hosts. The following script is a bash script for creating the topology.

```
#!/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
sudo ovs-vsctl set Bridge $1 protocols=OpenFlow13
}

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 $1-to-$2 type veth peer name $2-to-$1
    ip link set $1-to-$2 up
    ip link set $2-to-$1 up
    ovs-vsctl add-port $1 $1-to-$2 -- set Interface $1-to-$2 ofport_request=$5
    ovs-vsctl add-port $2 $2-to-$1 -- set Interface $2-to-$1 ofport_request=$5
}

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

# Creating namespaces
for i in {1..10}; do
    create_ns "host$i"
done

# Creating bridges
for i in {1..5}; do
    create_ovs_bridge "br$i"
    attach_ovs_to_sdn "br$i"
    for j in {1..2}; do
        k=$((2*(i-1)+j))
        ip=$((k+1))
        attach_ns_to_ovs "host$k" "br$i" "veth-host$k" "veth-host$k-br" $j "10.0.0.$ip"
    done
done

# Connecting bridges
for i in {1..4}; do
    j=$((i + 1))
    attach_ovs_to_ovs "br$i" "br$j" "br-ovs$i" "br-ovs$j" 1
done

# Testing connectivity
for i in {1..4}; do
    j=$((i + 1))
    ip netns exec "host$i" ping -c 1 "10.0.0.$j"
    ip netns exec "host$j" ping -c 1 "10.0.0.$i"
done
```

For creating the network slices, I used two different methods. One is by using flowtables in ovs-ofctl and another one is by using vlan in ovs. With vlan, we can tag a port of a switch under a certain vlan id and hosts under the same vlan id can reach each other but not between two different vlans. For this problem, I used python script. The two scripts for two different methods are given below:

# Network slicing using flowtables:

```
import os

# Defining the IP ranges for the Red and Blue slices
red_ips = ["10.0.0.2", "10.0.0.4", "10.0.0.6", "10.0.0.8", "10.0.0.10"]
blue_ips = ["10.0.0.3", "10.0.0.5", "10.0.0.7", "10.0.0.9", "10.0.0.11"]

# Defining the bridges
bridges = ["br1", "br2", "br3", "br4", "br5"]

# Adding flow rules to isolate the Red and Blue slices
for bridge in bridges:
  for ip in red_ips:
    # Allowing traffic within the Red slice
    os.system(f"ovs-ofctl add-flow {bridge} ip,nw_src={ip},actions=output:1")
    # Dropping traffic from the Red slice to the Blue slice
    for blue_ip in blue_ips:
      os.system(f"ovs-ofctl add-flow {bridge}
ip,nw_src={ip},nw_dst={blue_ip},actions=drop")

  for ip in blue_ips:
    # Allowing traffic within the Blue slice
    os.system(f"ovs-ofctl add-flow {bridge} ip,nw_src={ip},actions=output:2")
    # Dropping traffic from the Blue slice to the Red slice
    for red_ip in red_ips:
      os.system(f"ovs-ofctl add-flow {bridge}
ip,nw_src={ip},nw_dst={red_ip},actions=drop")
```

Network slicing using vlan:

```
import os

# Defining the VLAN IDs for the Red and Blue slices
red_vlan = 100
```

```
blue_vlan = 200

# Defining the bridges
bridges = ["br1", "br2", "br3", "br4", "br5"]

# Defining the ports for the Red and Blue slices
red_ports = [1, 3, 5, 7, 9]
blue_ports = [2, 4, 6, 8, 10]

# Adding VLAN tags to the ports
for bridge, red_port, blue_port in zip(bridges, red_ports, blue_ports):
   # Adding the Red VLAN tag to the Red port
   os.system(f"ovs-vsctl set Port {bridge}-eth{red_port} tag={red_vlan}")
   # Adding the Blue VLAN tag to the Blue port
   os.system(f"ovs-vsctl set Port {bridge}-eth{blue_port} tag={blue_vlan}")

# Isolating the VLANs
for bridge in bridges:
   # Dropping traffic between the Red and Blue VLANs
   os.system(f"ovs-ofctl add-flow {bridge} priority=100,dl_vlan={red_vlan},actions=drop")
   os.system(f"ovs-ofctl add-flow {bridge}
priority=100,dl_vlan={blue_vlan},actions=drop")
```