Research Project

Digital Twin with SDN

Student: Jing Yan (jing.yan@studenti.unitn.it)
Supervisor: Fabrizio Granelli (fabrizio.granelli@unitn.it)
Department of Information Engineering and Computer Science (DISI)
The University of Trento, 2021-2022

Updated: Jan 11, 2021

1. Introduction

1.1 Basic introduction to SDN and Digital Twin

A digital twin is a virtual representation that serves as the real-time digital counterpart of a physical object or process [1]. Indeed, digital twin is a very popular concept in the industrial 4.0 era, through which, companies could simulate the physical object or process through software, and collect data from the simulation platform for further analysis. Recently, machine learning has been introduced to the digital twin platform too. With the huge amount of simulation results generated by digital twin platforms, machine learning could help to make a better decision for the companies whether expanding manufacturing line is feasible or not.

Software-defined networking, namely SDN, has been widely used in core networks in recent years, as compared with traditional routers and switches, SDN enables dynamic, programmatically efficient network configuration in order to improve network performance and monitoring. With the popularity of cloud computing, SDN has been playing more and more important roles in network management in cloud computing environments.

1.2 Main design and implementation of the research project

Here, my research project will combine the two concepts I mentioned in the previous chapter, namely digital twin and SDN. To be more precise, what I have done as follows is more like designing and implementing a tool to create the digital twin for an SDN-based network.

The first major function of this research project is to mirror SDN-based network topology. Given any SDN-based network topology, for example, linear network topology, tree network topology, or star network topology, I need to implement a program to mirror the network topology. In this process, two sub-tasks are included, detecting all the devices in this network topology, and connecting all the devices in the same way in the original network topology.

As for how to implement this function, basically, both the original SDN-based network topology and the mirroring SDN-based network topology consist of ONOS SDN controller and the network topology simulated by Mininet. So, I could detect all the devices, and the connections among all the devices in the original network topology through the RestAPI exposed by the ONOS SDN controller.

The second major function for this research project is to mirror the network traffic in the given SDN-based network topology. Two items are needed to be considered here, capturing the network traffic in the original network topology, and replaying the network traffic in the mirroring network topology.

As for how to implement this function, as I have mentioned above, the original SDN-based network topology is simulated by Mininet, to be more detailed, the hosts in the original network topology are Linux namespaces, which are created by Linux command (“ip netns”). Thus, I could access the namespaces and capture all the network traffic with Linux command (“tcpdump”) from the simulated ethernet interfaces. Finally, with the help of an open-source tool, namely “tcpreplay”, I could replay all the network traffic in the mirroring SDN-based network topology.

The final major function is to automate all the processes I mentioned above through “docker-compose” function, namely, integrating all the programs to the docker images.

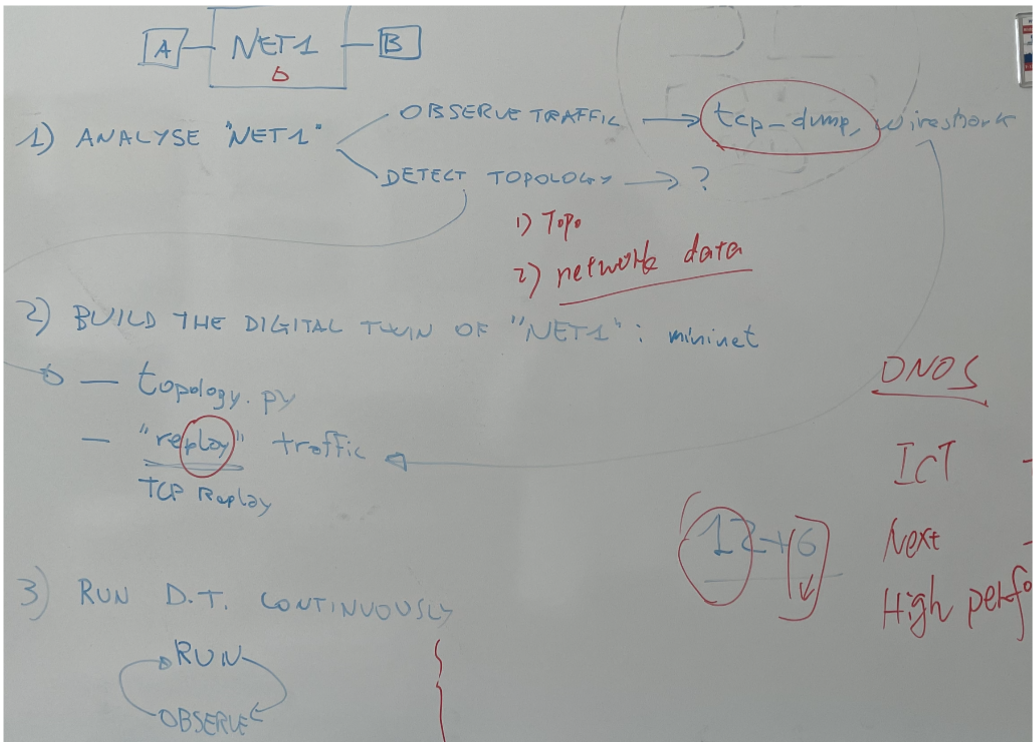


Figure 1: Original idea for this research project provided by my supervisor

1.3 Challenges and efforts have been done before

The main challenge for this research project at the beginning is that I have no idea how to start the project. It will be too complex that I implement all the tools from zero, and if I select from existing solutions, first, I need to learn the tool, and second, there are so many choices, it will take a huge amount of effort to select the proper tools, and third, if I start the project with existing tools, I will totally get stuck if the tool could not provide specific functions I need as I am hardly able to re-develop the existing tool again. Secondly, my laptop is not good enough for me to finish the whole project, however, I have no choice as could not get any laboratory computing resources, thus, it also took me lots of effort to solve the compatibility issues.

As for the previous work I have done for this project, the first item is to select the proper tool for this project, I have tried Microsoft® Azure Digital Twins platform and done some research for two open-source digital twin projects, cps-twining and ditto, none of them could be applied for this research project. Then, I asked for advice from my supervisor, he recommended me “comnetsemu” project, which could be set up successfully. However, it has some problems in connecting with the ONOS controller. Following this hint, I tried to build ONOS VM, which has failed as the compatibility issue for Kafka. After lots of trials, I gave up and started to look into container-based solutions, and I succeeded in building the Mininet container and ONOS containers.

After figuring out the proper method to build the SDN-based network topology, I encountered one minor buggy issue, which consumed me lots of time to fix, namely, the original SDN-based network topology and the mirroring SDN-based network topology can not share the same ONOS controller, otherwise, the mirroring SDN-based network topology could not be detected by the ONOS controller properly. Mirroring SDN-based network topology succeeded after fixing this minor issue. During this process, I also found a bug for the Mininet container, “ovs-vswitchd” service sometimes fails to start automatically after starting the Mininet container, which needs to be started manually.

Eventually, for the network traffic replaying part, the first issue is that I have to create Linux namespaces to simulate the hosts in network topologies with Shell scripts (I have also implemented the Python script to create SDN-based network topologies, the hosts created by Python scripts can not be accessed.), so that network traffic capturing can be conducted properly. Plus, the tool “tcpreplay” can not replay TCP network traffic as it can not simulate the TCP connection building process, so in this project, I only implemented ICMP and UDP network traffic replaying.

2. Implementation

2.1 Environment setup

2.1.1 Setup network emulator - Mininet container

Pull the Mininet docker image from dockerhub, run a container named “mininet” from the image, and test the mn commands as follows to create a simple network topology with one switch connecting with two hosts. As the following commands show, everything works well.

```
yanjing@yanjingdeMacBook-Pro ~ % docker pull iwaseyusuke/mininet

yanjing@yanjingdeMacBook-Pro ~ % docker run -it --rm --privileged -e DISPLAY
-v /tmp/.X11-unix:/tmp/.X11-unix -v /Users/yanjing/Desktop/mininet:/tmp
--name mininet iwaseyusuke/mininet
* /etc/openvswitch/conf.db does not exist
* Creating empty database /etc/openvswitch/conf.db
* Starting ovsdb-server
* Configuring Open vSwitch system IDs
* Starting ovs-vswitchd
* Enabling remote OVSDB managers
root@3574bf3255b1:~# mn
*** Error setting resource limits. Mininet's performance may be affected.
*** 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> exit
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 3.814 seconds
```

Get the IP address of the Mininet container with the following command. The IP address might change after stopping/pruning and running the container again.

```
yanjing@yanjingdeMacBook-Pro ~ % docker inspect --format '{{
.NetworkSettings.IPAddress }}' mininet
172.17.0.2
```

2.1.2 Setup SDN controller - ONOS container

Pull the ONOS docker image from dockerhub, run a container named “onos” from the image.

```
yanjing@yanjingdeMacBook-Pro ~ % docker pull onosproject/onos

yanjing@yanjingdeMacBook-Pro ~ % docker run -it -d -p 8181:8181 -p 8101:8101
-p 5005:5005 -p 830:830 --name onos onosproject/onos

yanjing@yanjingdeMacBook-Pro ~ % docker container ps -a
CONTAINER ID   IMAGE                  COMMAND                   CREATED
STATUS        PORTS
NAMES
5436695dfb7e   onosproject/onos       "./bin/onos-service …"   4 hours ago
Up 4 hours    0.0.0.0:830->830/tcp, 6640/tcp, 0.0.0.0:5005->5005/tcp,
0.0.0.0:8101->8101/tcp, 6653/tcp, 0.0.0.0:8181->8181/tcp, 9876/tcp   onos
3574bf3255b1   iwaseyusuke/mininet    "/ENTRYPOINT.sh"          4 hours ago
Up 4 hours    6633/tcp, 6640/tcp, 6653/tcp
mininet
```

Access the “onos” container and install “openssh-server” package in the running container so that it can be accessed through ssh.

```
yanjing@yanjingdeMacBook-Pro ~ % docker exec -it onos /bin/bash
root@5436695dfb7e:~/onos# apt-get update; apt-get install openssh-server
```

Access the “onos” container through ssh with the username/password (karaf/karaf) and activate the two applications, namely “org.onosproject.openflow” and “org.onosproject.fwd” as follows.

```
yanjing@yanjingdeMacBook-Pro  ~  %  ssh  -p  8101  -o  StrictHostKeyChecking=no
karaf@localhost -X
Password authentication
(karaf@localhost) Password:
Welcome to Open Network Operating System (ONOS)!
    ____  _  ______  ____
   / __ \/ |/ / __ \/ __/
  / /_/ /    / /_/ /\ \
  \____/_/|_/\____/___/

Documentation: wiki.onosproject.org
Tutorials:     tutorials.onosproject.org
Mailing lists: lists.onosproject.org

Come help out! Find out how at: contribute.onosproject.org

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'logout' to exit ONOS session.

karaf@root > app activate org.onosproject.openflow
karaf@root > app activate org.onosproject.fwd
```

Get the IP address of the Mininet container with the following command. The IP address might change after stopping/pruning and running the container again.

```
yanjing@yanjingdeMacBook-Pro ~ % docker inspect --format '{{
.NetworkSettings.IPAddress }}' onos
172.17.0.3
```

Access the WEB GUI of the ONOS controller through the following link and username/password.
http://127.0.0.1:8181/onos/ui/login.html (onos/rocks).

2.1.3 Test the connection to the ONOS controller from Mininet

Using the "mn" command with “--controller” parameters, the IP address can be obtained through the commands above. The default port to connect remote ONOS controller is 6653. After setting up the ONOS controller for the default minimum network topology with the “mn” command, use the "dump" command to check the switches, hosts, and the ONOS controller information. As the following commands show, everything works well.

```
root@3574bf3255b1:~# mn --controller=remote,ip=172.17.0.3
```

```
*** Error setting resource limits. Mininet's performance may be affected.
*** Creating network
*** Adding controller
Connecting to remote controller at 172.17.0.3:6653
*** 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> dump
<Host h1: h1-eth0:10.0.0.1 pid=1611>
<Host h2: h2-eth0:10.0.0.2 pid=1613>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=1618>
<RemoteController{'ip': '172.17.0.3'} c0: 172.17.0.3:6653 pid=1605>
mininet> exit
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 7.683 seconds
```

2.2 Build the tree network topology with Mininet Python library

2.2.1 Implement the Python script to create the tree network topology

The following python script is used to create the tree network topology, which includes 4 switches and 5 hosts. Detailed topology can be seen in the figure as follows.

```
yanjing@yanjingdeMacBook-Pro ~ % cat create_tree_topo.py
from mininet.net import Mininet
from mininet.node import Controller, OVSKernelSwitch, RemoteController
from mininet.cli import CLI
# Initiate the network topology
topo = Mininet(controller=RemoteController, switch=OVSKernelSwitch)
# Add controllers to the topology
c1 = topo.addController("c1", controller=RemoteController, ip="172.17.0.3",
port=6653)
# Add hosts to the topology
h1 = topo.addHost("h1", ip="192.168.1.10" )
h2 = topo.addHost("h2", ip="192.168.1.20" )
h3 = topo.addHost("h3", ip="192.168.1.30" )
h4 = topo.addHost("h4", ip="192.168.1.40" )
h5 = topo.addHost("h5", ip="192.168.1.50" )
h6 = topo.addHost("h6", ip="192.168.1.50" )
# Add switches to the topology
```

```
s1 = topo.addSwitch("s1", protocols="OpenFlow14")
s2 = topo.addSwitch("s2", protocols="OpenFlow14")
s3 = topo.addSwitch("s3", protocols="OpenFlow14")
s4 = topo.addSwitch("s4", protocols="OpenFlow14")
# Set links between switches
s1.linkTo( s2 )
s1.linkTo( s3 )
s1.linkTo( s4 )
s1.linkTo( h5 )
s1.linkTo( h6 )
# Set links between switches and hosts
s2.linkTo( h1 )
s3.linkTo( h2 )
s3.linkTo( h3 )
s4.linkTo( h4 )
# Build and start the network topology with the hosts, switches, links and
remote controllers above
topo.build()
c1.start()
s1.start( [c1] )
s2.start( [c1] )
s3.start( [c1] )
s4.start( [c1] )
topo.start()
# Test the arp, ping and iperf
topo.staticArp()
topo.pingAll()
topo.iperf()
CLI( topo )
# If you need to stop the network topology, uncomment the following sentence,
otherwise, using "mn -c" to clear the network topology
# topo.stop()
```

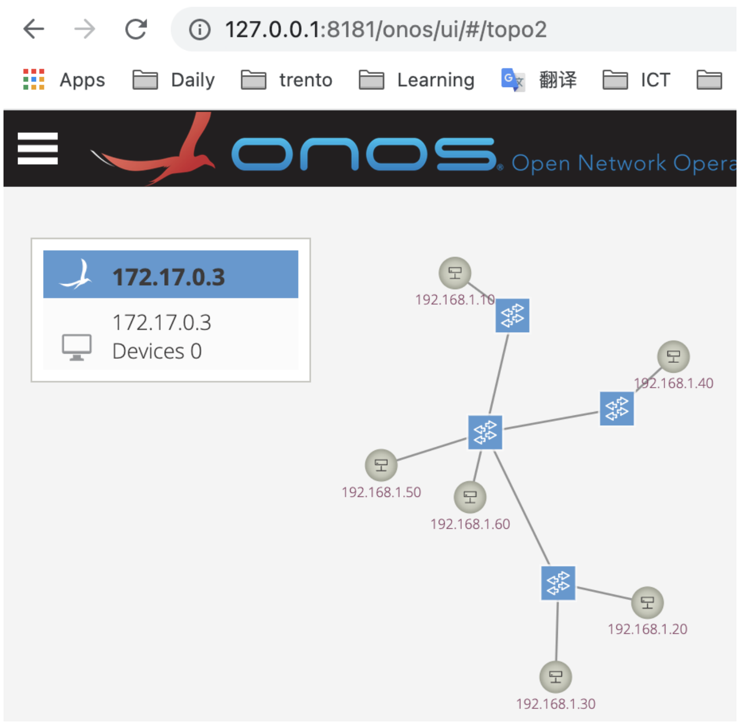
2.2.2 Execute the python script to create the tree network topology

Note: This python script is executed in the Mininet container.

```
root@3574bf3255b1:~# python create_tree_topo.py
*** Error setting resource limits. Mininet's performance may be affected.
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['7.08 Gbits/sec', '7.09 Gbits/sec']
mininet> dump
<Host h1: h1-eth0:192.168.1.10 pid=1948>
<Host h2: h2-eth0:192.168.1.20 pid=1950>
<Host h3: h3-eth0:192.168.2.30 pid=1952>
<Host h4: h4-eth0:192.168.2.40 pid=1954>
<Host h5: h5-eth0:192.168.2.50 pid=1956>
<OVSSwitch s1:
lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None,s1-eth4:None pid=1961>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None pid=1964>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None pid=1967>
<OVSSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None pid=1970>
<RemoteController c1: 172.17.0.3:6653 pid=1942>
```

2.2.3 Check the tree network topology through WEB GUI

Input "http://127.0.0.1:8181/onos/ui/login.html" in the browser with username/password (onos/rocks).



2.2.4 Check the switches, hosts, and links information through CLI

Note: The following information is obtained from the ONOS controller after logging in to ONOS controller with the command "ssh -p 8101 -o StrictHostKeyChecking=no karaf@localhost -X" (username/password: karaf/karaf).

```
karaf@root > nodes
id=172.17.0.3, address=172.17.0.3:9876, state=READY,
version=3.0.0.e8ff53e1c5, updated=4h27m ago *

karaf@root > devices
id=of:0000000000000001, available=true, local-status=connected 26m10s ago,
role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.9.0,
serial=None, chassis=1, driver=ovs, channelId=172.17.0.2:56826,
datapathDescription=None, managementAddress=172.17.0.2, protocol=OF_14
id=of:0000000000000002, available=true, local-status=connected 26m10s ago,
role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.9.0,
serial=None, chassis=2, driver=ovs, channelId=172.17.0.2:56828,
datapathDescription=None, managementAddress=172.17.0.2, protocol=OF_14
id=of:0000000000000003, available=true, local-status=connected 26m10s ago,
role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.9.0,
serial=None, chassis=3, driver=ovs, channelId=172.17.0.2:56830,
datapathDescription=None, managementAddress=172.17.0.2, protocol=OF_14
id=of:0000000000000004, available=true, local-status=connected 26m9s ago,
role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.9.0,
serial=None, chassis=4, driver=ovs, channelId=172.17.0.2:56832,
datapathDescription=None, managementAddress=172.17.0.2, protocol=OF_14

karaf@root > hosts
```

```
id=26:E3:00:6F:D5:33/None, mac=26:E3:00:6F:D5:33,
locations=[of:0000000000000003/2], auxLocations=null, vlan=None,
ip(s)=[192.168.1.20], innerVlan=None, outerTPID=unknown,
provider=of:org.onosproject.provider.host, configured=false
id=36:09:26:A1:10:7F/None, mac=36:09:26:A1:10:7F,
locations=[of:0000000000000002/2], auxLocations=null, vlan=None,
ip(s)=[192.168.1.10], innerVlan=None, outerTPID=unknown,
provider=of:org.onosproject.provider.host, configured=false
id=72:B9:30:5A:33:B0/None, mac=72:B9:30:5A:33:B0,
locations=[of:0000000000000004/2], auxLocations=null, vlan=None,
ip(s)=[192.168.2.40], innerVlan=None, outerTPID=unknown,
provider=of:org.onosproject.provider.host, configured=false
id=B2:DD:36:BC:97:C2/None, mac=B2:DD:36:BC:97:C2,
locations=[of:0000000000000001/4], auxLocations=null, vlan=None,
ip(s)=[192.168.2.50], innerVlan=None, outerTPID=unknown,
provider=of:org.onosproject.provider.host, configured=false
id=BE:86:66:FE:1B:7A/None, mac=BE:86:66:FE:1B:7A,
locations=[of:0000000000000003/3], auxLocations=null, vlan=None,
ip(s)=[192.168.2.30], innerVlan=None, outerTPID=unknown,
provider=of:org.onosproject.provider.host, configured=false

karaf@root > links
src=of:0000000000000001/1, dst=of:0000000000000002/1, type=DIRECT,
state=ACTIVE, expected=false
src=of:0000000000000001/2, dst=of:0000000000000003/1, type=DIRECT,
state=ACTIVE, expected=false
src=of:0000000000000001/3, dst=of:0000000000000004/1, type=DIRECT,
state=ACTIVE, expected=false
src=of:0000000000000002/1, dst=of:0000000000000001/1, type=DIRECT,
state=ACTIVE, expected=false
src=of:0000000000000003/1, dst=of:0000000000000001/2, type=DIRECT,
state=ACTIVE, expected=false
src=of:0000000000000004/1, dst=of:0000000000000001/3, type=DIRECT,
state=ACTIVE, expected=false

karaf@root > ports
10:44:22
id=of:0000000000000001, available=true, local-status=connected 44s ago,
role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.9.0,
serial=None, chassis=1, driver=ovs, channelId=172.17.0.2:56032,
datapathDescription=None, managementAddress=172.17.0.2, protocol=OF_14
 port=LOCAL, state=disabled, type=copper, speed=0 , adminState=disabled,
portMac=b6:bf:f8:e3:f5:41, portName=s1
 port=1, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=0a:b7:86:4c:66:03, portName=s1-eth1
 port=2, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=72:55:11:7e:3c:83, portName=s1-eth2
 port=3, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=0e:59:f9:bc:bf:34, portName=s1-eth3
 port=4, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=5a:d3:6b:cc:b6:d0, portName=s1-eth4
 port=5, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=06:29:b7:e0:91:ab, portName=s1-eth5
id=of:0000000000000002, available=true, local-status=connected 44s ago,
role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.9.0,
serial=None, chassis=2, driver=ovs, channelId=172.17.0.2:56034,
datapathDescription=None, managementAddress=172.17.0.2, protocol=OF_14
 port=LOCAL, state=disabled, type=copper, speed=0 , adminState=disabled,
portMac=da:54:35:5f:af:4b, portName=s2
 port=1, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=7e:92:41:d6:9b:3b, portName=s2-eth1
```

```
 port=2, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=ce:08:11:da:09:bf, portName=s2-eth2
id=of:0000000000000003, available=true, local-status=connected 44s ago,
role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.9.0,
serial=None, chassis=3, driver=ovs, channelId=172.17.0.2:56036,
datapathDescription=None, managementAddress=172.17.0.2, protocol=OF_14
 port=LOCAL, state=disabled, type=copper, speed=0 , adminState=disabled,
portMac=86:7b:f2:91:d7:45, portName=s3
 port=1, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=fa:0c:43:82:c6:60, portName=s3-eth1
 port=2, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=3e:4b:28:02:80:f5, portName=s3-eth2
 port=3, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=4a:03:c3:c2:ac:e6, portName=s3-eth3
id=of:0000000000000004, available=true, local-status=connected 44s ago,
role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.9.0,
serial=None, chassis=4, driver=ovs, channelId=172.17.0.2:56038,
datapathDescription=None, managementAddress=172.17.0.2, protocol=OF_14
 port=LOCAL, state=disabled, type=copper, speed=0 , adminState=disabled,
portMac=16:99:49:ad:ea:4a, portName=s4
 port=1, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=fe:e0:ff:00:68:fa, portName=s4-eth1
 port=2, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=2a:d3:11:5f:20:c5, portName=s4-eth2
```

2.3 Build the tree network topology with Shell command

2.3.1 Implement the Shell script to create the tree network topology

```
# 1. Create namespaces
ip netns add h1
ip netns add h2
ip netns add h3
ip netns add h4
ip netns add h5
ip netns add h6

# 2. Create openvswitch
ovs-vsctl add-br s1
ovs-vsctl set bridge s1 protocols=OpenFlow14
ovs-vsctl add-br s2
ovs-vsctl set bridge s2 protocols=OpenFlow14
ovs-vsctl add-br s3
ovs-vsctl set bridge s3 protocols=OpenFlow14
ovs-vsctl add-br s4
ovs-vsctl set bridge s4 protocols=OpenFlow14

# 3.1 Create vethernet links among switches and hosts
ip link add h1-eth1 type veth peer name s2-eth1
ip link add h2-eth1 type veth peer name s3-eth1
ip link add h3-eth1 type veth peer name s3-eth2
ip link add h4-eth1 type veth peer name s4-eth1
ip link add h5-eth1 type veth peer name s1-eth4
ip link add h6-eth1 type veth peer name s1-eth5

# 3.2 Create vethernet links among switches and switches
ip link add s1-eth1 type veth peer name s2-eth2
ip link add s1-eth2 type veth peer name s3-eth3
ip link add s1-eth3 type veth peer name s4-eth2
```

```
# 4. Move host ports into namespaces
ip link set h1-eth1 netns h1
ip link set h2-eth1 netns h2
ip link set h3-eth1 netns h3
ip link set h4-eth1 netns h4
ip link set h5-eth1 netns h5
ip link set h6-eth1 netns h6

# 5. Connect switch ports to OVS
ovs-vsctl add-port s1 s1-eth1
ovs-vsctl add-port s1 s1-eth2
ovs-vsctl add-port s1 s1-eth3
ovs-vsctl add-port s1 s1-eth4
ovs-vsctl add-port s1 s1-eth5
ovs-vsctl add-port s2 s2-eth1
ovs-vsctl add-port s2 s2-eth2
ovs-vsctl add-port s3 s3-eth1
ovs-vsctl add-port s3 s3-eth2
ovs-vsctl add-port s3 s3-eth3
ovs-vsctl add-port s4 s4-eth1
ovs-vsctl add-port s4 s4-eth2

# 6. Connect controller to switch
ovs-vsctl set-controller s1 tcp:172.17.0.3:6653
ovs-vsctl set-controller s2 tcp:172.17.0.3:6653
ovs-vsctl set-controller s3 tcp:172.17.0.3:6653
ovs-vsctl set-controller s4 tcp:172.17.0.3:6653

# 7.1 Setup ip for hosts
ip netns exec h1 ifconfig h1-eth1 192.168.1.10 hw ether 0E:76:73:DE:95:0B
ip netns exec h1 ifconfig lo up
ip netns exec h2 ifconfig h2-eth1 192.168.1.20 hw ether 46:F1:1F:61:13:BE
ip netns exec h2 ifconfig lo up
ip netns exec h3 ifconfig h3-eth1 192.168.1.30 hw ether C2:CF:6D:1F:04:68
ip netns exec h3 ifconfig lo up
ip netns exec h4 ifconfig h4-eth1 192.168.1.40 hw ether B6:9D:D9:84:A7:64
ip netns exec h4 ifconfig lo up
ip netns exec h5 ifconfig h5-eth1 192.168.1.50 hw ether 42:00:FB:6C:9D:E3
ip netns exec h5 ifconfig lo up
ip netns exec h6 ifconfig h6-eth1 192.168.1.60 hw ether 1A:16:D9:D6:FA:6E
ip netns exec h6 ifconfig lo up

# 7.2 Setup ip for switches
ifconfig s1-eth1 up
ifconfig s1-eth2 up
ifconfig s1-eth3 up
ifconfig s1-eth4 up
ifconfig s1-eth5 up
ifconfig s2-eth1 up
ifconfig s2-eth2 up
ifconfig s3-eth1 up
ifconfig s3-eth2 up
ifconfig s3-eth3 up
ifconfig s4-eth1 up
ifconfig s4-eth2 up

# 8 Test connections between hosts
ip netns exec h1 ping -c1 192.168.1.10
ip netns exec h1 ping -c1 192.168.1.20
ip netns exec h1 ping -c1 192.168.1.30
ip netns exec h1 ping -c1 192.168.1.40
ip netns exec h1 ping -c1 192.168.1.50
ip netns exec h1 ping -c1 192.168.1.60
```

2.3.2 Execute the Shell script to create the tree network topology

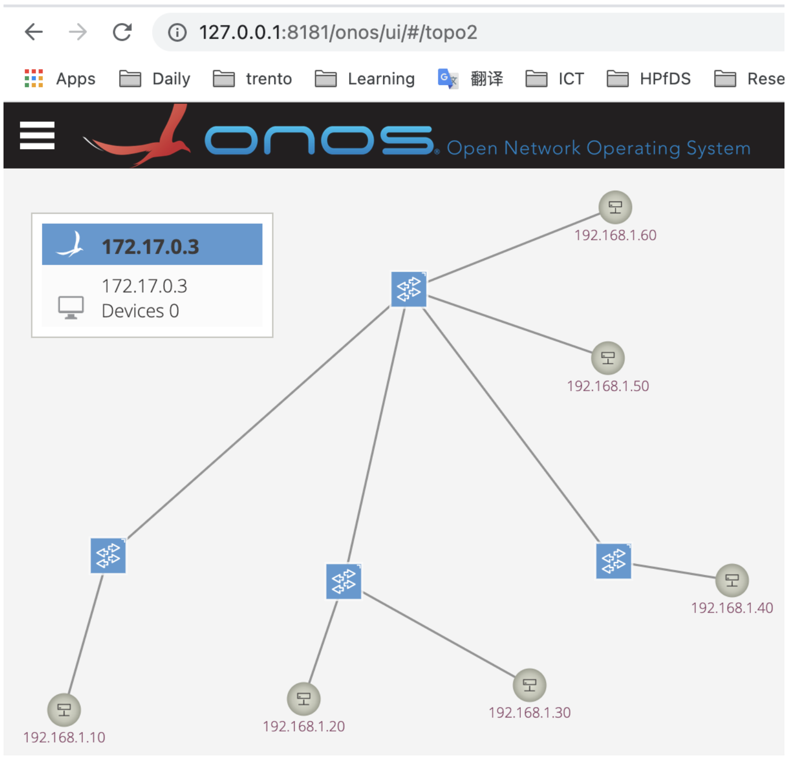
Note: This Shell script is executed in the Mininet container.

```
root@13b87a9c672a:~# sh create_tree_topo.sh
PING 192.168.1.10 (192.168.1.10) 56(84) bytes of data.
64 bytes from 192.168.1.10: icmp_seq=1 ttl=64 time=0.035 ms

--- 192.168.1.10 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.035/0.035/0.035/0.000 ms
PING 192.168.1.20 (192.168.1.20) 56(84) bytes of data.
64 bytes from 192.168.1.20: icmp_seq=1 ttl=64 time=1090 ms

--- 192.168.1.20 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1090.421/1090.421/1090.421/0.000 ms
PING 192.168.1.30 (192.168.1.30) 56(84) bytes of data.
64 bytes from 192.168.1.30: icmp_seq=1 ttl=64 time=49.2 ms

--- 192.168.1.30 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 49.206/49.206/49.206/0.000 ms
PING 192.168.1.40 (192.168.1.40) 56(84) bytes of data.
64 bytes from 192.168.1.40: icmp_seq=1 ttl=64 time=36.9 ms

--- 192.168.1.40 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 36.932/36.932/36.932/0.000 ms
PING 192.168.1.50 (192.168.1.50) 56(84) bytes of data.
64 bytes from 192.168.1.50: icmp_seq=1 ttl=64 time=81.1 ms

--- 192.168.1.50 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 81.194/81.194/81.194/0.000 ms
PING 192.168.1.60 (192.168.1.60) 56(84) bytes of data.
64 bytes from 192.168.1.60: icmp_seq=1 ttl=64 time=104 ms

--- 192.168.1.60 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 104.816/104.816/104.816/0.000 ms
root@13b87a9c672a:~#
root@13b87a9c672a:~# ip netns
h3 (id: 3)
h2 (id: 2)
h6 (id: 6)
h5 (id: 5)
h1 (id: 1)
h4 (id: 4)
root@13b87a9c672a:~# ovs-vsctl list-br
s1
s2
s3
s4
root@13b87a9c672a:~# ovs-vsctl list-ports s1
s1-eth1
```

```
s1-eth2
s1-eth3
s1-eth4
s1-eth5
root@13b87a9c672a:~# ovs-vsctl list-ports s2
s2-eth1
s2-eth2
```

2.3.3 Check the tree network topology through WEB GUI

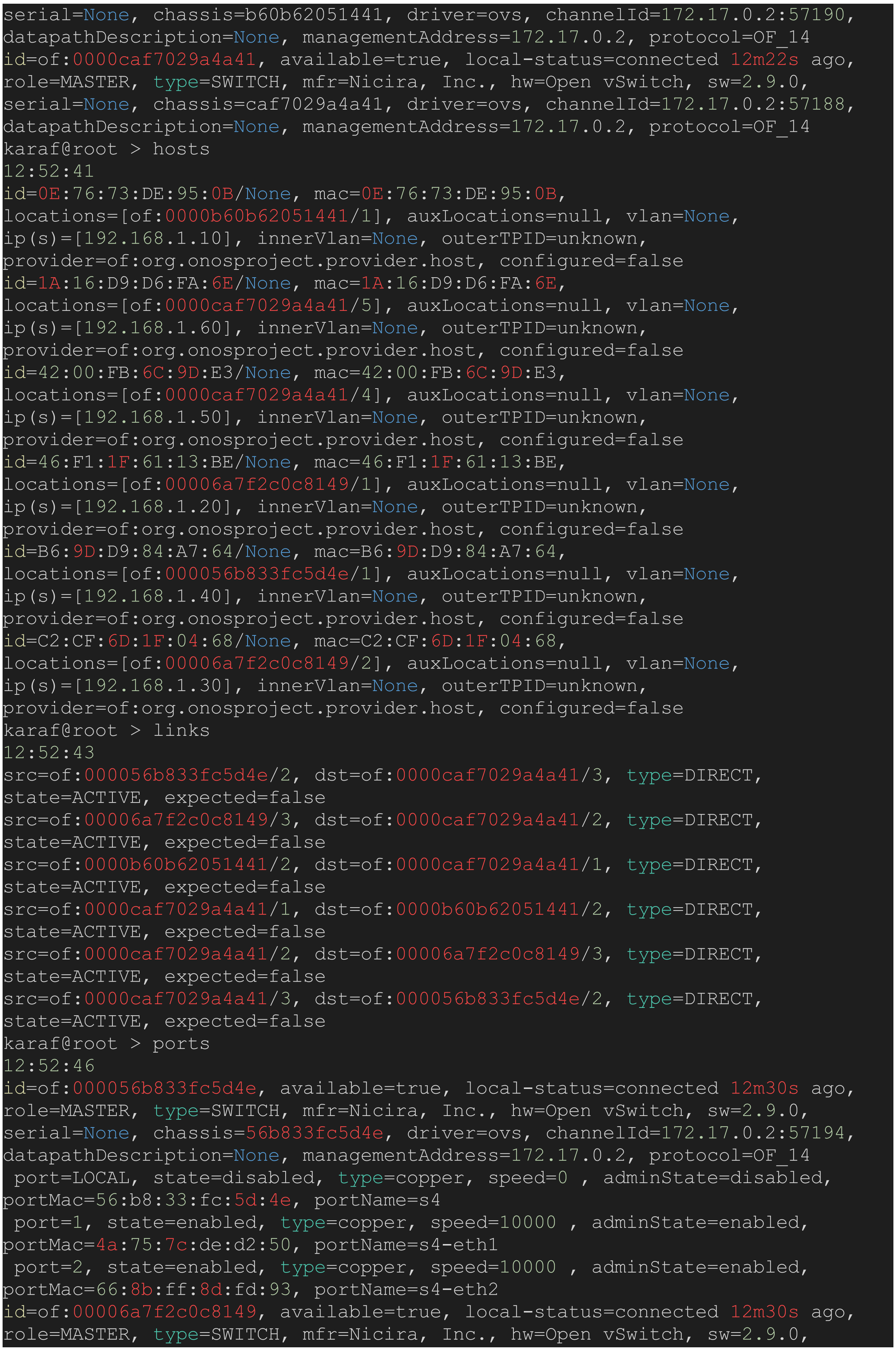
Input "http://127.0.0.1:8181/onos/ui/login.html" in the browser with username/password (onos/rocks).



2.3.4 Check the switches, hosts, and links information through CLI

Note: The following information is obtained from the ONOS controller after logging in to ONOS controller with the command "ssh -p 8101 -o StrictHostKeyChecking=no karaf@localhost -X" (username/password: karaf/karaf).

```
karaf@root > nodes
12:52:37
id=172.17.0.3, address=172.17.0.3:9876, state=READY,
version=3.0.0.8c79f3f7e3, updated=17m55s ago *
karaf@root > devices
12:52:38
id=of:000056b833fc5d4e, available=true, local-status=connected 12m22s ago,
role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.9.0,
serial=None, chassis=56b833fc5d4e, driver=ovs, channelId=172.17.0.2:57194,
datapathDescription=None, managementAddress=172.17.0.2, protocol=OF_14
id=of:00006a7f2c0c8149, available=true, local-status=connected 12m22s ago,
role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.9.0,
serial=None, chassis=6a7f2c0c8149, driver=ovs, channelId=172.17.0.2:57192,
datapathDescription=None, managementAddress=172.17.0.2, protocol=OF_14
id=of:0000b60b62051441, available=true, local-status=connected 12m22s ago,
role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.9.0,
```



```
serial=None, chassis=b60b62051441, driver=ovs, channelId=172.17.0.2:57190,
datapathDescription=None, managementAddress=172.17.0.2, protocol=OF_14
id=of:0000caf7029a4a41, available=true, local-status=connected 12m22s ago,
role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.9.0,
serial=None, chassis=caf7029a4a41, driver=ovs, channelId=172.17.0.2:57188,
datapathDescription=None, managementAddress=172.17.0.2, protocol=OF_14
karaf@root > hosts
12:52:41
id=0E:76:73:DE:95:0B/None, mac=0E:76:73:DE:95:0B,
locations=[of:0000b60b62051441/1], auxLocations=null, vlan=None,
ip(s)=[192.168.1.10], innerVlan=None, outerTPID=unknown,
provider=of:org.onosproject.provider.host, configured=false
id=1A:16:D9:D6:FA:6E/None, mac=1A:16:D9:D6:FA:6E,
locations=[of:0000caf7029a4a41/5], auxLocations=null, vlan=None,
ip(s)=[192.168.1.60], innerVlan=None, outerTPID=unknown,
provider=of:org.onosproject.provider.host, configured=false
id=42:00:FB:6C:9D:E3/None, mac=42:00:FB:6C:9D:E3,
locations=[of:0000caf7029a4a41/4], auxLocations=null, vlan=None,
ip(s)=[192.168.1.50], innerVlan=None, outerTPID=unknown,
provider=of:org.onosproject.provider.host, configured=false
id=46:F1:1F:61:13:BE/None, mac=46:F1:1F:61:13:BE,
locations=[of:00006a7f2c0c8149/1], auxLocations=null, vlan=None,
ip(s)=[192.168.1.20], innerVlan=None, outerTPID=unknown,
provider=of:org.onosproject.provider.host, configured=false
id=B6:9D:D9:84:A7:64/None, mac=B6:9D:D9:84:A7:64,
locations=[of:000056b833fc5d4e/1], auxLocations=null, vlan=None,
ip(s)=[192.168.1.40], innerVlan=None, outerTPID=unknown,
provider=of:org.onosproject.provider.host, configured=false
id=C2:CF:6D:1F:04:68/None, mac=C2:CF:6D:1F:04:68,
locations=[of:00006a7f2c0c8149/2], auxLocations=null, vlan=None,
ip(s)=[192.168.1.30], innerVlan=None, outerTPID=unknown,
provider=of:org.onosproject.provider.host, configured=false
karaf@root > links
12:52:43
src=of:000056b833fc5d4e/2, dst=of:0000caf7029a4a41/3, type=DIRECT,
state=ACTIVE, expected=false
src=of:00006a7f2c0c8149/3, dst=of:0000caf7029a4a41/2, type=DIRECT,
state=ACTIVE, expected=false
src=of:0000b60b62051441/2, dst=of:0000caf7029a4a41/1, type=DIRECT,
state=ACTIVE, expected=false
src=of:0000caf7029a4a41/1, dst=of:0000b60b62051441/2, type=DIRECT,
state=ACTIVE, expected=false
src=of:0000caf7029a4a41/2, dst=of:00006a7f2c0c8149/3, type=DIRECT,
state=ACTIVE, expected=false
src=of:0000caf7029a4a41/3, dst=of:000056b833fc5d4e/2, type=DIRECT,
state=ACTIVE, expected=false
karaf@root > ports
12:52:46
id=of:000056b833fc5d4e, available=true, local-status=connected 12m30s ago,
role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.9.0,
serial=None, chassis=56b833fc5d4e, driver=ovs, channelId=172.17.0.2:57194,
datapathDescription=None, managementAddress=172.17.0.2, protocol=OF_14
 port=LOCAL, state=disabled, type=copper, speed=0 , adminState=disabled,
portMac=56:b8:33:fc:5d:4e, portName=s4
 port=1, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=4a:75:7c:de:d2:50, portName=s4-eth1
 port=2, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=66:8b:ff:8d:fd:93, portName=s4-eth2
id=of:00006a7f2c0c8149, available=true, local-status=connected 12m30s ago,
role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.9.0,
serial=None, chassis=6a7f2c0c8149, driver=ovs, channelId=172.17.0.2:57192,
datapathDescription=None, managementAddress=172.17.0.2, protocol=OF_14
 port=LOCAL, state=disabled, type=copper, speed=0 , adminState=disabled,
portMac=6a:7f:2c:0c:81:49, portName=s3
 port=1, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=96:73:32:cc:85:0f, portName=s3-eth1
 port=2, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=ae:d5:25:c0:ed:e1, portName=s3-eth2
 port=3, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=ce:23:75:98:3a:70, portName=s3-eth3
id=of:0000b60b62051441, available=true, local-status=connected 12m30s ago,
role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.9.0,
serial=None, chassis=b60b62051441, driver=ovs, channelId=172.17.0.2:57190,
datapathDescription=None, managementAddress=172.17.0.2, protocol=OF_14
 port=LOCAL, state=disabled, type=copper, speed=0 , adminState=disabled,
portMac=b6:0b:62:05:14:41, portName=s2
 port=1, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=72:d0:2b:ca:db:1b, portName=s2-eth1
 port=2, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=5a:e1:cf:f5:11:0b, portName=s2-eth2
id=of:0000caf7029a4a41, available=true, local-status=connected 12m30s ago,
role=MASTER, type=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.9.0,
serial=None, chassis=caf7029a4a41, driver=ovs, channelId=172.17.0.2:57188,
datapathDescription=None, managementAddress=172.17.0.2, protocol=OF_14
 port=LOCAL, state=disabled, type=copper, speed=0 , adminState=disabled,
portMac=ca:f7:02:9a:4a:41, portName=s1
 port=1, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=be:4e:24:2e:b1:9d, portName=s1-eth1
 port=2, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=16:ec:9c:02:b7:68, portName=s1-eth2
 port=3, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=72:ec:8d:e1:69:e3, portName=s1-eth3
 port=4, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=ca:4b:db:92:ca:c0, portName=s1-eth4
 port=5, state=enabled, type=copper, speed=10000 , adminState=enabled,
portMac=02:ca:01:68:e4:75, portName=s1-eth5
```

2.4 Mirror the tree network topology

2.4.1 Run the second Mininet container for the mirror env

```
yanjing@yanjingdeMacBook-Pro ~ % docker run -it --rm --privileged -e DISPLAY
-v /tmp/.X11-unix:/tmp/.X11-unix -v /Users/yanjing/Desktop/mininet:/tmp
--name mininet_mirror iwaseyusuke/mininet
* /etc/openvswitch/conf.db does not exist
* Creating empty database /etc/openvswitch/conf.db
* Starting ovsdb-server
* Configuring Open vSwitch system IDs
* Starting ovs-vswitchd
* Enabling remote OVSDB managers
root@03e27b7246ed:~#

yanjing@yanjingdeMacBook-Pro ~ % docker inspect --format '{{
.NetworkSettings.IPAddress }}' mininet_mirror
172.17.0.4
```

2.4.2 Run the second ONOS container for the mirror env

```
yanjing@yanjingdeMacBook-Pro ~ % docker run -it -d -p 8182:8181 -p 8102:8101
-p 5006:5005 -p 831:830 --name onos_mirror onosproject/onos

yanjing@yanjingdeMacBook-Pro ~ % docker exec -it onos_mirror /bin/bash
root@af0fc6c03c6b:~/onos# apt-get update; apt-get install openssh-server

yanjing@yanjingdeMacBook-Pro ~ % ssh -p 8102 -o StrictHostKeyChecking=no
karaf@localhost -X
Warning: Permanently added '[localhost]:8102' (RSA) to the list of known
hosts.
Password authentication
(karaf@localhost) Password:
Welcome to Open Network Operating System (ONOS)!
    ____  _  ______  ____
   / __ \/ |/ / __ \/ __/
  / /_/ /    / /_/ /\ \
  \____/_/|_/\____/___/

Documentation: wiki.onosproject.org
Tutorials:     tutorials.onosproject.org
Mailing lists: lists.onosproject.org

Come help out! Find out how at: contribute.onosproject.org

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'logout' to exit ONOS session.

karaf@root > app activate org.onosproject.fwd
karaf@root > app activate org.onosproject.openflow

yanjing@yanjingdeMacBook-Pro ~ % docker inspect --format '{{
.NetworkSettings.IPAddress }}' onos_mirror
172.17.0.5
```

2.4.3 Detect the tree network topology through REST API

Note: all the content returned from “curl” commands is JSON format, which is similar to the content in 2.2.4, so I do not display the information here.

```
yanjing@yanjingdeMacBook-Pro ~ % curl -X GET -u onos:rocks
http://127.0.0.1:8181/onos/v1/devices
yanjing@yanjingdeMacBook-Pro ~ % curl -X GET -u onos:rocks
http://127.0.0.1:8181/onos/v1/hosts
yanjing@yanjingdeMacBook-Pro ~ % curl -X GET -u onos:rocks
http://127.0.0.1:8181/onos/v1/links
yanjing@yanjingdeMacBook-Pro ~ % curl -X GET -u onos:rocks
http://127.0.0.1:8181/onos/v1/devices/$device_id/ports
```

In addition, the commands are executed on the host, which can be also executed on the Mininet container, in this case, the “url” should be modified from the host IP address to the ONOS container IP address as follows.

```
yanjing@yanjingdeMacBook-Pro ~ % docker run -it --rm --privileged -e DISPLAY
-v /tmp/.X11-unix:/tmp/.X11-unix -v /Users/yanjing/Desktop/mininet:/tmp
--name mininet_mirror iwaseyusuke/mininet
* /etc/openvswitch/conf.db does not exist
* Creating empty database /etc/openvswitch/conf.db
* Starting ovsdb-server
* Configuring Open vSwitch system IDs
* Starting ovs-vswitchd
* Enabling remote OVSDB managers
root@03e27b7246ed:~#
root@03e27b7246ed:~# curl -X GET -u onos:rocks
http://172.17.0.3:8181/onos/v1/devices
root@03e27b7246ed:~# curl -X GET -u onos:rocks
http://172.17.0.3:8181/onos/v1/hosts
root@03e27b7246ed:~# curl -X GET -u onos:rocks
http://172.17.0.3:8181/onos/v1/links
root@03e27b7246ed:~# curl -X GET -u onos:rocks
http://172.17.0.3:8181/onos/v1/devices/$device_id/ports
```

2.4.4 Implement Python script to detect the tree network topology and generate Shell script to mirror the tree network topology

The following python script is used to detect the network topology created from the first Mininet, including switches, hosts, links among hosts and switches, and links among switches and switches.

```
yanjing@yanjingdeMacBook-Pro ~ % cat detect_topo_shell.py
#!/usr/bin/python2.7
import requests, json
from requests.auth import HTTPBasicAuth

url = 'http://127.0.0.1:8181/onos/v1/'

# Request the info of switches, and store the id info to switch_id_list
switch_id_list = []
res_devices = requests.get(url + "devices", auth=HTTPBasicAuth("onos",
"rocks")).json()["devices"]
for index in range(0, len(res_devices)):
   switch_id_list.append(res_devices[index]["id"])
print("switch_id_list is: ", switch_id_list)

# Request the ports info of switches based on the id info obtained above, and
store the name, id and ports info to switch_list
switch_list = []
for switch_id in switch_id_list:
   res_ports = requests.get(url + "devices/" + switch_id + "/ports",
auth=HTTPBasicAuth("onos", "rocks")).json()["ports"]
   switch_ports = []
   for port in res_ports:
       if port["port"] == "local":
           switch_name = port["annotations"]["portName"]
       else:
           switch_ports.append((port["port"],
port["annotations"]["portName"]))
   switch_list.append({"switch_name": switch_name, "switch_id": switch_id,
"switch_ports": switch_ports})

for switch in switch_list:
   print("Each switch info is: ", switch)
```

```
# Request the host_ip, host_locations info, and store the name, ip and
location info to host_list
host_list = []
res_hosts = requests.get(url + "hosts", auth=HTTPBasicAuth("onos",
"rocks")).json()["hosts"]
for index in range(1, len(res_hosts)+1):
   host_list.append({"host_name": "h%s" % index, "host_port": "h%s-eth1" %
index, "host_ip": res_hosts[index-1]["ipAddresses"][0], "host_mac":
res_hosts[index-1]["mac"], "host_switch_connection":
res_hosts[index-1]["locations"][0]})

for host in host_list:
   print("Each host info is: ", host)

# Request the link info among switches, and store the src_switch_port and
dst_switch_port to link_list
link_list = []
res_links = requests.get(url + "links", auth=HTTPBasicAuth("onos",
"rocks")).json()["links"]
for link in res_links:
   for switch in switch_list:
       if link["src"]["device"] == switch["switch_id"]:
           for port in switch["switch_ports"]:
               if link["src"]["port"] == port[0]:
                   src_link = port[1]
   for switch in switch_list:
       if link["dst"]["device"] == switch["switch_id"]:
           for port in switch["switch_ports"]:
               if link["dst"]["port"] == port[0]:
                   dst_link = port[1]
   link_list.append(sorted((src_link, dst_link)))

valid_link_list = []
for link in link_list:
   if link not in valid_link_list:
       valid_link_list.append(link)
print("Connection info among switches is: ", valid_link_list)

# Generate the shell script for mirroring network topology
with open("./mirror_tree_topo.sh", "w") as filename:
   filename.write("# 0. Clean the env\n")
   filename.write("mn -c\n")
   filename.write("\n")

   filename.write("# 1. Create namespaces\n")
   for index in range(1, len(res_hosts)+1):
       filename.write("ip netns add h%s\n" % index)
   filename.write("\n")

   filename.write("# 2. Create openvswitch\n")
   for switch in switch_list:
       filename.write("ovs-vsctl add-br %s\n" % switch["switch_name"])
       filename.write("ovs-vsctl set bridge %s protocols=OpenFlow14\n" %
switch["switch_name"])
   filename.write("\n")

   filename.write("# 3.1 Create vethernet links among switches and hosts\n")
   for host in host_list:
       for switch in switch_list:
           if host["host_switch_connection"]["elementId"] ==
switch["switch_id"]:
```

```
                for port in switch["switch_ports"]:
                    if host["host_switch_connection"]["port"] == port[0]:
                        filename.write("ip link add %s type veth peer name
%s\n" % (host["host_port"], port[1]))
    filename.write("\n")

    filename.write("# 3.2 Create vethernet links among switches and
switches\n")
    for link in valid_link_list:
        filename.write("ip link add %s type veth peer name %s\n" % (link[0],
link[1]))
    filename.write("\n")

    filename.write("# 4. Move host ports into namespaces\n")
    for host in host_list:
        filename.write("ip link set %s netns %s\n" % (host["host_port"],
host["host_name"]))
    filename.write("\n")

    filename.write("# 5. Connect switch ports to OVS\n")
    for switch in switch_list:
        for port in switch["switch_ports"]:
            filename.write("ovs-vsctl add-port %s %s\n" %
(switch["switch_name"], port[1]))
    filename.write("\n")

    filename.write("# 6. Connect controller to switch\n")
    for switch in switch_list:
        filename.write("ovs-vsctl set-controller %s tcp:172.17.0.5:6653\n" %
switch["switch_name"])
    filename.write("\n")

    filename.write("# 7.1 Setup ip for hosts\n")
    for host in host_list:
        filename.write("ip netns exec %s ifconfig %s %s hw ether %s\n" %
(host["host_name"], host["host_port"], host["host_ip"], host["host_mac"]))
        filename.write("ip netns exec %s ifconfig lo up\n" %
host["host_name"])
    filename.write("\n")

    filename.write("# 7.2 Setup ip for switches\n")
    for switch in switch_list:
        for port in switch["switch_ports"]:
            filename.write("ifconfig %s up\n" % port[1])
    filename.write("\n")

    filename.write("# 8 Test connections between hosts\n")
    for host in host_list:
        filename.write("ip netns exec h1 ping -c1 %s\n" % host["host_ip"])
```

2.4.5 Execute Python script to detect the tree network topology and generate the Shell script to mirror the tree network topology

Note: This python script is executed on the host, which can be also executed on the second Mininet container, in this case, the “url” variable in the python script should be modified as “http://172.17.0.3:8181/onos/v1/”, namely replace the host IP address with ONOS container IP address.

```
yanjing@yanjingdeMacBook-Pro ~ % python detect_topo_shell.py
```

```
switch_id_list is:  ['of:000056b833fc5d4e', 'of:0000caf7029a4a41',
'of:0000b60b62051441', 'of:00006a7f2c0c8149']
Each switch info is:  {'switch_name': 's4', 'switch_id':
'of:000056b833fc5d4e', 'switch_ports': [('1', 's4-eth1'), ('2', 's4-eth2')]}
Each switch info is:  {'switch_name': 's1', 'switch_id':
'of:0000caf7029a4a41', 'switch_ports': [('1', 's1-eth1'), ('2', 's1-eth2'),
('3', 's1-eth3'), ('4', 's1-eth4'), ('5', 's1-eth5')]}
Each switch info is:  {'switch_name': 's2', 'switch_id':
'of:0000b60b62051441', 'switch_ports': [('1', 's2-eth1'), ('2', 's2-eth2')]}
Each switch info is:  {'switch_name': 's3', 'switch_id':
'of:00006a7f2c0c8149', 'switch_ports': [('1', 's3-eth1'), ('2', 's3-eth2'),
('3', 's3-eth3')]}
Each host info is:  {'host_name': 'h1', 'host_port': 'h1-eth1', 'host_ip':
'192.168.1.10', 'host_mac': '0E:76:73:DE:95:0B', 'host_switch_connection':
{'elementId': 'of:0000b60b62051441', 'port': '1'}}
Each host info is:  {'host_name': 'h2', 'host_port': 'h2-eth1', 'host_ip':
'192.168.1.20', 'host_mac': '46:F1:1F:61:13:BE', 'host_switch_connection':
{'elementId': 'of:00006a7f2c0c8149', 'port': '1'}}
Each host info is:  {'host_name': 'h3', 'host_port': 'h3-eth1', 'host_ip':
'192.168.1.30', 'host_mac': 'C2:CF:6D:1F:04:68', 'host_switch_connection':
{'elementId': 'of:00006a7f2c0c8149', 'port': '2'}}
Each host info is:  {'host_name': 'h4', 'host_port': 'h4-eth1', 'host_ip':
'192.168.1.50', 'host_mac': '42:00:FB:6C:9D:E3', 'host_switch_connection':
{'elementId': 'of:0000caf7029a4a41', 'port': '4'}}
Each host info is:  {'host_name': 'h5', 'host_port': 'h5-eth1', 'host_ip':
'192.168.1.40', 'host_mac': 'B6:9D:D9:84:A7:64', 'host_switch_connection':
{'elementId': 'of:000056b833fc5d4e', 'port': '1'}}
Each host info is:  {'host_name': 'h6', 'host_port': 'h6-eth1', 'host_ip':
'192.168.1.60', 'host_mac': '1A:16:D9:D6:FA:6E', 'host_switch_connection':
{'elementId': 'of:0000caf7029a4a41', 'port': '5'}}
Connection info among switches is:  [['s1-eth1', 's2-eth2'], ['s1-eth2',
's3-eth3'], ['s1-eth3', 's4-eth2']]

yanjing@yanjingdeMacBook-Pro ~ % ls -l mirror_tree_topo.sh
-rw-r--r--  1 yanjing  staff  964 16 Jan 20:33 mirror_tree_topo.sh

yanjing@yanjingdeMacBook-Pro ~ % cat mirror_tree_topo.sh
# 0. Clean the env
mn -c

# 1. Create namespaces
ip netns add h1
ip netns add h2
ip netns add h3
ip netns add h4
ip netns add h5
ip netns add h6

# 2. Create openvswitch
ovs-vsctl add-br s4
ovs-vsctl set bridge s4 protocols=OpenFlow14
ovs-vsctl add-br s1
ovs-vsctl set bridge s1 protocols=OpenFlow14
ovs-vsctl add-br s2
ovs-vsctl set bridge s2 protocols=OpenFlow14
ovs-vsctl add-br s3
ovs-vsctl set bridge s3 protocols=OpenFlow14

# 3.1 Create vethernet links among switches and hosts
ip link add h1-eth1 type veth peer name s2-eth1
ip link add h2-eth1 type veth peer name s3-eth1
ip link add h3-eth1 type veth peer name s3-eth2
ip link add h4-eth1 type veth peer name s1-eth4
ip link add h5-eth1 type veth peer name s4-eth1
ip link add h6-eth1 type veth peer name s1-eth5

# 3.2 Create vethernet links among switches and switches
ip link add s1-eth1 type veth peer name s2-eth2
ip link add s1-eth2 type veth peer name s3-eth3
ip link add s1-eth3 type veth peer name s4-eth2

# 4. Move host ports into namespaces
ip link set h1-eth1 netns h1
ip link set h2-eth1 netns h2
ip link set h3-eth1 netns h3
ip link set h4-eth1 netns h4
ip link set h5-eth1 netns h5
ip link set h6-eth1 netns h6

# 5. Connect switch ports to OVS
ovs-vsctl add-port s4 s4-eth1
ovs-vsctl add-port s4 s4-eth2
ovs-vsctl add-port s1 s1-eth1
ovs-vsctl add-port s1 s1-eth2
ovs-vsctl add-port s1 s1-eth3
ovs-vsctl add-port s1 s1-eth4
ovs-vsctl add-port s1 s1-eth5
ovs-vsctl add-port s2 s2-eth1
ovs-vsctl add-port s2 s2-eth2
ovs-vsctl add-port s3 s3-eth1
ovs-vsctl add-port s3 s3-eth2
ovs-vsctl add-port s3 s3-eth3

# 6. Connect controller to switch
ovs-vsctl set-controller s4 tcp:172.17.0.5:6653
ovs-vsctl set-controller s1 tcp:172.17.0.5:6653
ovs-vsctl set-controller s2 tcp:172.17.0.5:6653
ovs-vsctl set-controller s3 tcp:172.17.0.5:6653

# 7.1 Setup ip for hosts
ip netns exec h1 ifconfig h1-eth1 192.168.1.10 hw ether 0E:76:73:DE:95:0B
ip netns exec h1 ifconfig lo up
ip netns exec h2 ifconfig h2-eth1 192.168.1.20 hw ether 46:F1:1F:61:13:BE
ip netns exec h2 ifconfig lo up
ip netns exec h3 ifconfig h3-eth1 192.168.1.30 hw ether C2:CF:6D:1F:04:68
ip netns exec h3 ifconfig lo up
ip netns exec h4 ifconfig h4-eth1 192.168.1.50 hw ether 42:00:FB:6C:9D:E3
ip netns exec h4 ifconfig lo up
ip netns exec h5 ifconfig h5-eth1 192.168.1.40 hw ether B6:9D:D9:84:A7:64
ip netns exec h5 ifconfig lo up
ip netns exec h6 ifconfig h6-eth1 192.168.1.60 hw ether 1A:16:D9:D6:FA:6E
ip netns exec h6 ifconfig lo up

# 7.2 Setup ip for switches
ifconfig s4-eth1 up
ifconfig s4-eth2 up
ifconfig s1-eth1 up
ifconfig s1-eth2 up
ifconfig s1-eth3 up
ifconfig s1-eth4 up
ifconfig s1-eth5 up
ifconfig s2-eth1 up
ifconfig s2-eth2 up
ifconfig s3-eth1 up
ifconfig s3-eth2 up
ifconfig s3-eth3 up

# 8 Test connections between hosts
ip netns exec h1 ping -c1 192.168.1.10
ip netns exec h1 ping -c1 192.168.1.20
ip netns exec h1 ping -c1 192.168.1.30
ip netns exec h1 ping -c1 192.168.1.50
ip netns exec h1 ping -c1 192.168.1.40
ip netns exec h1 ping -c1 192.168.1.60
```

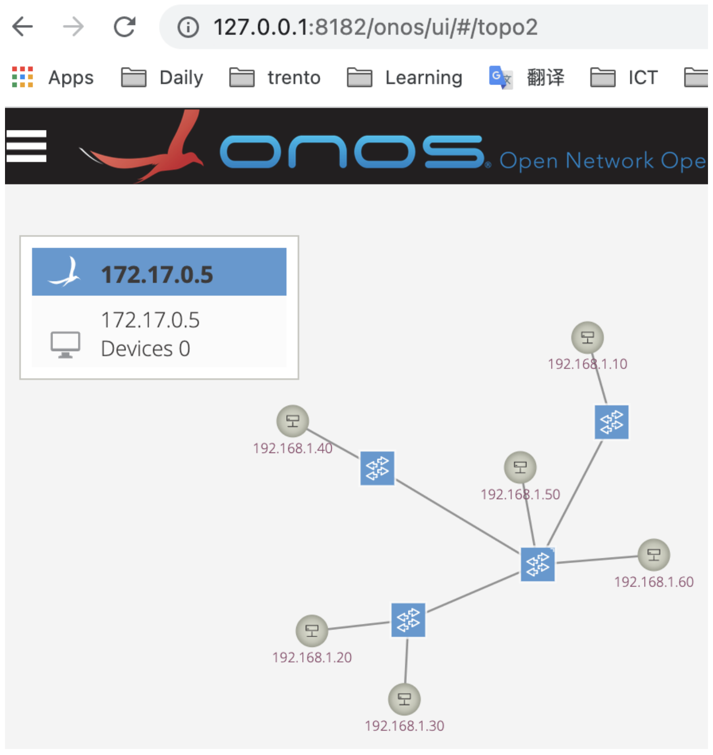
2.4.6 Execute Shell script to mirror the tree network topology

Execute the mirror_tree_topo.sh script in the mininet_mirror container, and test "ping" command among hosts.

```
root@7e5ab849492c:~# sh mirror_tree_topo.sh
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core
ovs-openflowd ovs-controller udpbwtest mnexec ivs 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core
ovs-openflowd ovs-controller udpbwtest mnexec ivs 2> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
***  Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
PING 192.168.1.20 (192.168.1.20) 56(84) bytes of data.
64 bytes from 192.168.1.20: icmp_seq=1 ttl=64 time=0.075 ms

--- 192.168.1.20 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.075/0.075/0.075/0.000 ms
PING 192.168.1.10 (192.168.1.10) 56(84) bytes of data.
64 bytes from 192.168.1.10: icmp_seq=1 ttl=64 time=1089 ms

--- 192.168.1.10 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1089.563/1089.563/1089.563/0.000 ms
PING 192.168.1.40 (192.168.1.40) 56(84) bytes of data.
64 bytes from 192.168.1.40: icmp_seq=1 ttl=64 time=43.6 ms

--- 192.168.1.40 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

```
rtt min/avg/max/mdev = 43.696/43.696/43.696/0.000 ms
PING 192.168.1.30 (192.168.1.30) 56(84) bytes of data.
64 bytes from 192.168.1.30: icmp_seq=1 ttl=64 time=15.0 ms

--- 192.168.1.30 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 15.060/15.060/15.060/0.000 ms
PING 192.168.1.50 (192.168.1.50) 56(84) bytes of data.
64 bytes from 192.168.1.50: icmp_seq=1 ttl=64 time=25.3 ms

--- 192.168.1.50 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 25.360/25.360/25.360/0.000 ms
PING 192.168.1.60 (192.168.1.60) 56(84) bytes of data.
64 bytes from 192.168.1.60: icmp_seq=1 ttl=64 time=26.0 ms

--- 192.168.1.60 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 26.028/26.028/26.028/0.000 ms
```

Check the network topology in the WEB GUI of the onos_mirror container.



2.5 Mirror network traffic in the tree network topology

Note: I captured the network traffic from the original network topology which was created by Shell script.

2.5.1 Install network traffic replay tool in the Mininet container in the mirror environment

```
On my host:
```

```
yanjing@yanjingdeMacBook-Pro mininet % wget
https://github.com/appneta/tcpreplay/releases/download/v4.3.4/tcpreplay-4.3.4
.tar.xz
yanjing@yanjingdeMacBook-Pro mininet % tar -xvf tcpreplay-4.3.4.tar.xz

On the mininet_mirror container:
root@a468c088934c:/tmp/tcpreplay-4.3.4# apt-get update -y
root@a468c088934c:/tmp/tcpreplay-4.3.4# apt-get install build-essential
libpcap-dev -y
root@a468c088934c:/tmp/tcpreplay-4.3.4# ./configure
root@a468c088934c:/tmp/tcpreplay-4.3.4# make
root@a468c088934c:/tmp/tcpreplay-4.3.4# make install
root@a468c088934c:/tmp/tcpreplay-4.3.4# tcpreplay --help
```

2.5.2 Initiate and capture ICMP packets in the tree network topology

ICMP network traffic is generated by the ping command and captured by the “tcpdump” command in the first Mininet container. As it is shown above, when starting the Mininet container, the host directory “/Users/yanjing/Desktop/mininet” is mapped to directory “/tmp” in the Mininet container, so all the network packets written to the “/tmp” directory in the Mininet container will be visible in the “/Users/yanjing/Desktop/mininet” directory in the host.

Generate network traffic in the first terminal of the Mininet container as follows,

```
root@cc761835abce:~# ip netns exec h1 ping -c1 192.168.1.20
PING 192.168.1.20 (192.168.1.20) 56(84) bytes of data.
64 bytes from 192.168.1.20: icmp_seq=1 ttl=64 time=38.3 ms

--- 192.168.1.20 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 38.368/38.368/38.368/0.000 ms

root@cc761835abce:~# ip netns exec h1 ping -c1 192.168.1.30
PING 192.168.1.30 (192.168.1.30) 56(84) bytes of data.
64 bytes from 192.168.1.30: icmp_seq=1 ttl=64 time=31.1 ms

--- 192.168.1.30 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 31.169/31.169/31.169/0.000 ms

root@cc761835abce:~# ip netns exec h1 ping -c1 192.168.1.40
PING 192.168.1.40 (192.168.1.40) 56(84) bytes of data.
64 bytes from 192.168.1.40: icmp_seq=1 ttl=64 time=35.7 ms

--- 192.168.1.40 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 35.722/35.722/35.722/0.000 ms

root@cc761835abce:~# ip netns exec h1 ping -c1 192.168.1.50
PING 192.168.1.50 (192.168.1.50) 56(84) bytes of data.
64 bytes from 192.168.1.50: icmp_seq=1 ttl=64 time=21.4 ms

--- 192.168.1.50 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 21.496/21.496/21.496/0.000 ms

root@cc761835abce:~# ip netns exec h1 ping -c1 192.168.1.60
PING 192.168.1.60 (192.168.1.60) 56(84) bytes of data.
```

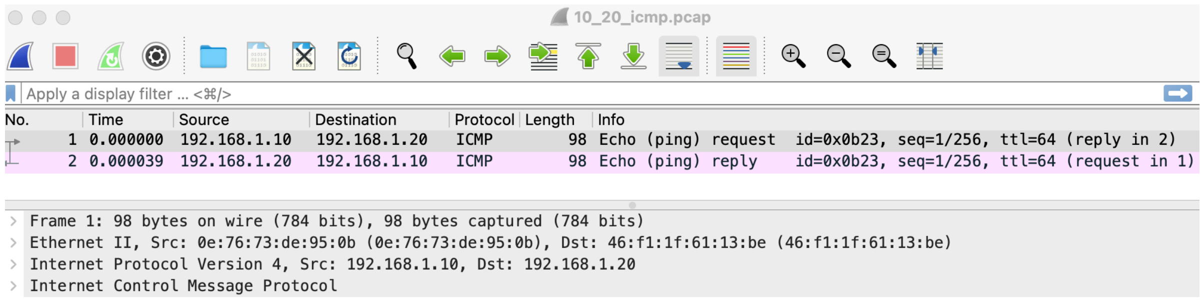
```
64 bytes from 192.168.1.60: icmp_seq=1 ttl=64 time=18.9 ms

--- 192.168.1.60 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 18.975/18.975/18.975/0.000 ms
```

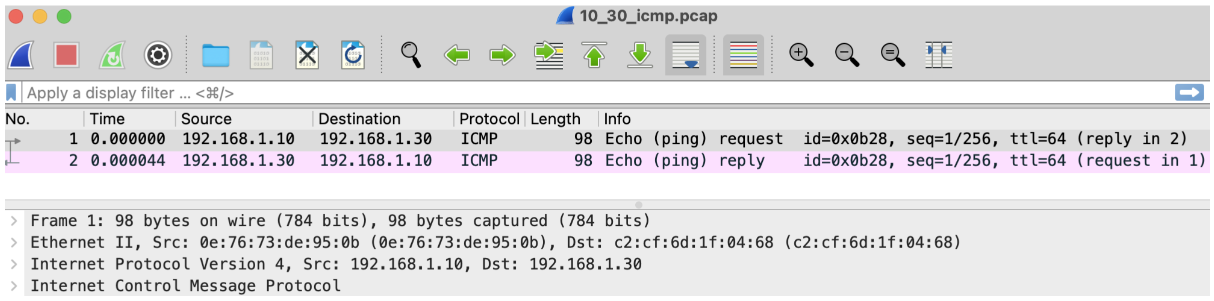
Capture the network traffic in the second terminal of Mininet container as follows,

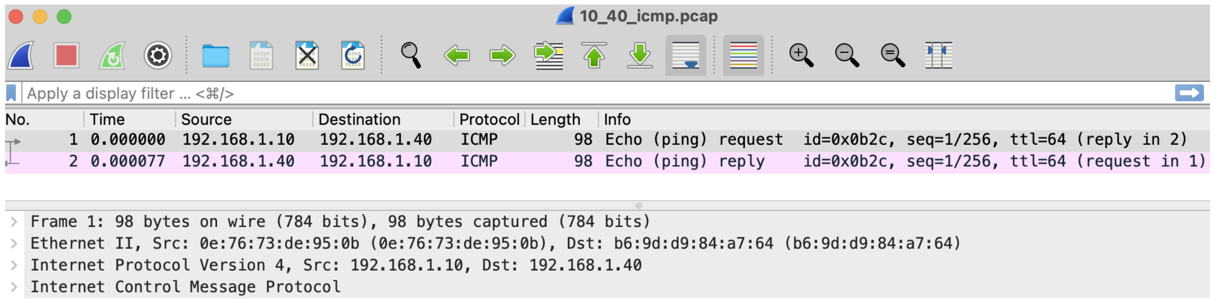
```
root@cc761835abce:/tmp#  ip  netns  exec  h2  tcpdump  -i  h2-eth1  icmp  -w
10_20_icmp.pcap
tcpdump:  listening  on  h2-eth1,  link-type  EN10MB  (Ethernet),  capture  size
262144 bytes
^C2 packets captured
2 packets received by filter
0 packets dropped by kernel

root@cc761835abce:/tmp#  ip  netns  exec  h3  tcpdump  -i  h3-eth1  icmp  -w
10_30_icmp.pcap
tcpdump:  listening  on  h3-eth1,  link-type  EN10MB  (Ethernet),  capture  size
262144 bytes
^C2 packets captured
2 packets received by filter
0 packets dropped by kernel

root@cc761835abce:/tmp#  ip  netns  exec  h4  tcpdump  -i  h4-eth1  icmp  -w
10_40_icmp.pcap
tcpdump:  listening  on  h4-eth1,  link-type  EN10MB  (Ethernet),  capture  size
262144 bytes
^C2 packets captured
2 packets received by filter
0 packets dropped by kernel

root@cc761835abce:/tmp#  ip  netns  exec  h5  tcpdump  -i  h5-eth1  icmp  -w
10_50_icmp.pcap
tcpdump:  listening  on  h5-eth1,  link-type  EN10MB  (Ethernet),  capture  size
262144 bytes
^C2 packets captured
2 packets received by filter
0 packets dropped by kernel

root@cc761835abce:/tmp#  ip  netns  exec  h6  tcpdump  -i  h6-eth1  icmp  -w
10_60_icmp.pcap
tcpdump:  listening  on  h6-eth1,  link-type  EN10MB  (Ethernet),  capture  size
262144 bytes
^C2 packets captured
2 packets received by filter
0 packets dropped by kernel
```

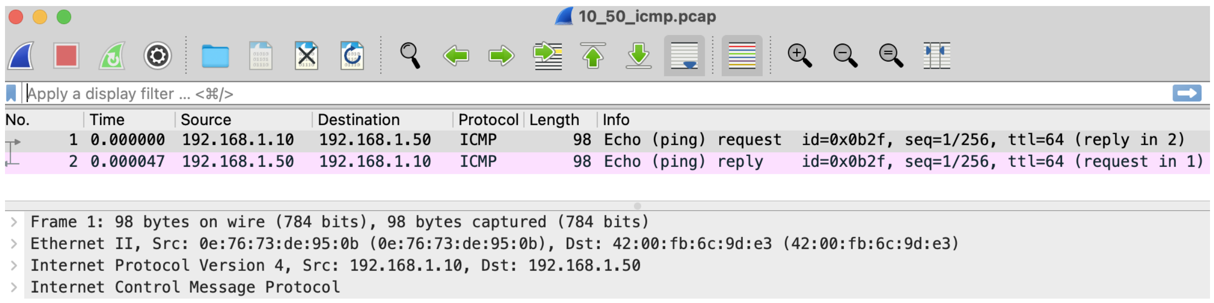
Check the “pcap” files on the host and have a glance with Wireshark

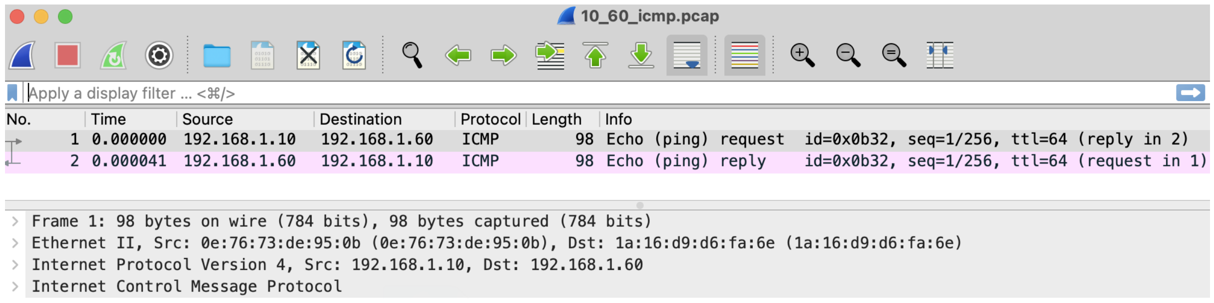
```
yanjing@yanjingdeMacBook-Pro ~ % ls -l /Users/yanjing/Desktop/mininet
total 40
-rw-r--r--@ 1 yanjing  staff  252 Jan 31 21:44 10_20_icmp.pcap
-rw-r--r--@ 1 yanjing  staff  252 Jan 31 21:45 10_30_icmp.pcap
-rw-r--r--  1 yanjing  staff  252 Jan 31 21:47 10_40_icmp.pcap
-rw-r--r--  1 yanjing  staff  252 Jan 31 21:47 10_50_icmp.pcap
-rw-r--r--  1 yanjing  staff  252 Jan 31 21:48 10_60_icmp.pcap
```


```
10_20_icmp.pcap
Apply a display filter ... <⌘/>
No.  Time      Source        Destination   Protocol Length Info
   1 0.000000  192.168.1.10  192.168.1.20  ICMP         98 Echo (ping) request  id=0x0b23, seq=1/256, ttl=64 (reply in 2)
   2 0.000039  192.168.1.20  192.168.1.10  ICMP         98 Echo (ping) reply    id=0x0b23, seq=1/256, ttl=64 (request in 1)

> Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
> Ethernet II, Src: 0e:76:73:de:95:0b (0e:76:73:de:95:0b), Dst: 46:f1:1f:61:13:be (46:f1:1f:61:13:be)
> Internet Protocol Version 4, Src: 192.168.1.10, Dst: 192.168.1.20
> Internet Control Message Protocol
```


```
10_30_icmp.pcap
Apply a display filter ... <⌘/>
No.  Time      Source        Destination   Protocol Length Info
   1 0.000000  192.168.1.10  192.168.1.30  ICMP         98 Echo (ping) request  id=0x0b28, seq=1/256, ttl=64 (reply in 2)
   2 0.000044  192.168.1.30  192.168.1.10  ICMP         98 Echo (ping) reply    id=0x0b28, seq=1/256, ttl=64 (request in 1)

> Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
> Ethernet II, Src: 0e:76:73:de:95:0b (0e:76:73:de:95:0b), Dst: c2:cf:6d:1f:04:68 (c2:cf:6d:1f:04:68)
> Internet Protocol Version 4, Src: 192.168.1.10, Dst: 192.168.1.30
> Internet Control Message Protocol
```


```
10_40_icmp.pcap
Apply a display filter ... <⌘/>
No.  Time      Source        Destination   Protocol Length Info
   1 0.000000  192.168.1.10  192.168.1.40  ICMP         98 Echo (ping) request  id=0x0b2c, seq=1/256, ttl=64 (reply in 2)
   2 0.000077  192.168.1.40  192.168.1.10  ICMP         98 Echo (ping) reply    id=0x0b2c, seq=1/256, ttl=64 (request in 1)

> Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
> Ethernet II, Src: 0e:76:73:de:95:0b (0e:76:73:de:95:0b), Dst: b6:9d:d9:84:a7:64 (b6:9d:d9:84:a7:64)
> Internet Protocol Version 4, Src: 192.168.1.10, Dst: 192.168.1.40
> Internet Control Message Protocol
```


```
10_50_icmp.pcap
Apply a display filter ... <⌘/>
No.  Time      Source        Destination   Protocol Length Info
   1 0.000000  192.168.1.10  192.168.1.50  ICMP         98 Echo (ping) request  id=0x0b2f, seq=1/256, ttl=64 (reply in 2)
   2 0.000047  192.168.1.50  192.168.1.10  ICMP         98 Echo (ping) reply    id=0x0b2f, seq=1/256, ttl=64 (request in 1)

> Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
> Ethernet II, Src: 0e:76:73:de:95:0b (0e:76:73:de:95:0b), Dst: 42:00:fb:6c:9d:e3 (42:00:fb:6c:9d:e3)
> Internet Protocol Version 4, Src: 192.168.1.10, Dst: 192.168.1.50
> Internet Control Message Protocol
```


```
10_60_icmp.pcap
Apply a display filter ... <⌘/>
No.  Time      Source        Destination   Protocol Length Info
   1 0.000000  192.168.1.10  192.168.1.60  ICMP         98 Echo (ping) request  id=0x0b32, seq=1/256, ttl=64 (reply in 2)
   2 0.000041  192.168.1.60  192.168.1.10  ICMP         98 Echo (ping) reply    id=0x0b32, seq=1/256, ttl=64 (request in 1)

> Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
> Ethernet II, Src: 0e:76:73:de:95:0b (0e:76:73:de:95:0b), Dst: 1a:16:d9:d6:fa:6e (1a:16:d9:d6:fa:6e)
> Internet Protocol Version 4, Src: 192.168.1.10, Dst: 192.168.1.60
> Internet Control Message Protocol
```

2.5.3 Replay ICMP network traffic in the mirror environment

In the mirroring Mininet container, check the IP address for each host as follows.

```
root@bb73fd6e7a26:/tmp# for i in {1..6}
> do
> ip netns exec h$i ifconfig h$i-eth1 | grep "inet 192" | awk '{print $2}'
> done
```

```
192.168.1.10
192.168.1.20
192.168.1.30
192.168.1.40
192.168.1.50
192.168.1.60
```

On the first terminal, replay the ICMP network traffic on h1 namespace.
On the second terminal, capture the ICMP network traffic on h2-h6 namesapce.

```
root@bb73fd6e7a26:/tmp# ip netns exec h1 tcpreplay -i h1-eth1 10_20_icmp.pcap
Actual: 2 packets (196 bytes) sent in 0.000385 seconds
Rated: 509090.9 Bps, 4.07 Mbps, 5194.80 pps
Flows: 2 flows, 5194.80 fps, 2 flow packets, 0 non-flow
Statistics for network device: h1-eth1
   Successful packets:        2
   Failed packets:            0
   Truncated packets:         0
   Retried packets (ENOBUFS): 0
   Retried packets (EAGAIN):  0

root@bb73fd6e7a26:~# ip netns exec h2 tcpdump -i h2-eth1 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
09:21:56.139605 IP 192.168.1.10 > 192.168.1.20: ICMP echo request, id 2851,
seq 1, length 64
09:21:56.139634 IP 192.168.1.20 > 192.168.1.10: ICMP echo reply, id 2851, seq
1, length 64
```

```
root@bb73fd6e7a26:/tmp# ip netns exec h1 tcpreplay -i h1-eth1 10_30_icmp.pcap
Actual: 2 packets (196 bytes) sent in 0.000149 seconds
Rated: 1315436.2 Bps, 10.52 Mbps, 13422.81 pps
Flows: 2 flows, 13422.81 fps, 2 flow packets, 0 non-flow
Statistics for network device: h1-eth1
   Successful packets:        2
   Failed packets:            0
   Truncated packets:         0
   Retried packets (ENOBUFS): 0
   Retried packets (EAGAIN):  0

root@bb73fd6e7a26:~# ip netns exec h3 tcpdump -i h3-eth1 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
09:28:09.797456 IP 192.168.1.10 > 192.168.1.30: ICMP echo request, id 2856,
seq 1, length 64
09:28:09.797537 IP 192.168.1.30 > 192.168.1.10: ICMP echo reply, id 2856, seq
1, length 64
```

```
root@bb73fd6e7a26:/tmp# ip netns exec h1 tcpreplay -i h1-eth1 10_40_icmp.pcap
Actual: 2 packets (196 bytes) sent in 0.000206 seconds
Rated: 951456.3 Bps, 7.61 Mbps, 9708.73 pps
Flows: 2 flows, 9708.73 fps, 2 flow packets, 0 non-flow
Statistics for network device: h1-eth1
   Successful packets:        2
   Failed packets:            0
   Truncated packets:         0
   Retried packets (ENOBUFS): 0
   Retried packets (EAGAIN):  0
```

```
root@bb73fd6e7a26:~# ip netns exec h4 tcpdump -i h4-eth1 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h4-eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
09:29:37.148022 IP 192.168.1.10 > 192.168.1.40: ICMP echo request, id 2860,
seq 1, length 64
09:29:37.148101 IP 192.168.1.40 > 192.168.1.10: ICMP echo reply, id 2860, seq
1, length 64
```

```
root@bb73fd6e7a26:/tmp# ip netns exec h1 tcpreplay -i h1-eth1 10_50_icmp.pcap
Actual: 2 packets (196 bytes) sent in 0.000606 seconds
Rated: 323432.3 Bps, 2.58 Mbps, 3300.33 pps
Flows: 2 flows, 3300.33 fps, 2 flow packets, 0 non-flow
Statistics for network device: h1-eth1
   Successful packets:        2
   Failed packets:            0
   Truncated packets:         0
   Retried packets (ENOBUFS): 0
   Retried packets (EAGAIN):  0

root@bb73fd6e7a26:~# ip netns exec h5 tcpdump -i h5-eth1 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h5-eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
09:31:23.551721 IP 192.168.1.10 > 192.168.1.50: ICMP echo request, id 2863,
seq 1, length 64
09:31:23.551784 IP 192.168.1.50 > 192.168.1.10: ICMP echo reply, id 2863, seq
1, length 64
```

```
root@bb73fd6e7a26:/tmp# ip netns exec h1 tcpreplay -i h1-eth1 10_60_icmp.pcap
Actual: 2 packets (196 bytes) sent in 0.000137 seconds
Rated: 1430656.9 Bps, 11.44 Mbps, 14598.54 pps
Flows: 2 flows, 14598.54 fps, 2 flow packets, 0 non-flow
Statistics for network device: h1-eth1
   Successful packets:        2
   Failed packets:            0
   Truncated packets:         0
   Retried packets (ENOBUFS): 0
   Retried packets (EAGAIN):  0

root@bb73fd6e7a26:~# ip netns exec h6 tcpdump -i h6-eth1 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h6-eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
09:32:20.349739 IP 192.168.1.10 > 192.168.1.60: ICMP echo request, id 2866,
seq 1, length 64
09:32:20.349891 IP 192.168.1.60 > 192.168.1.10: ICMP echo reply, id 2866, seq
1, length 64
```

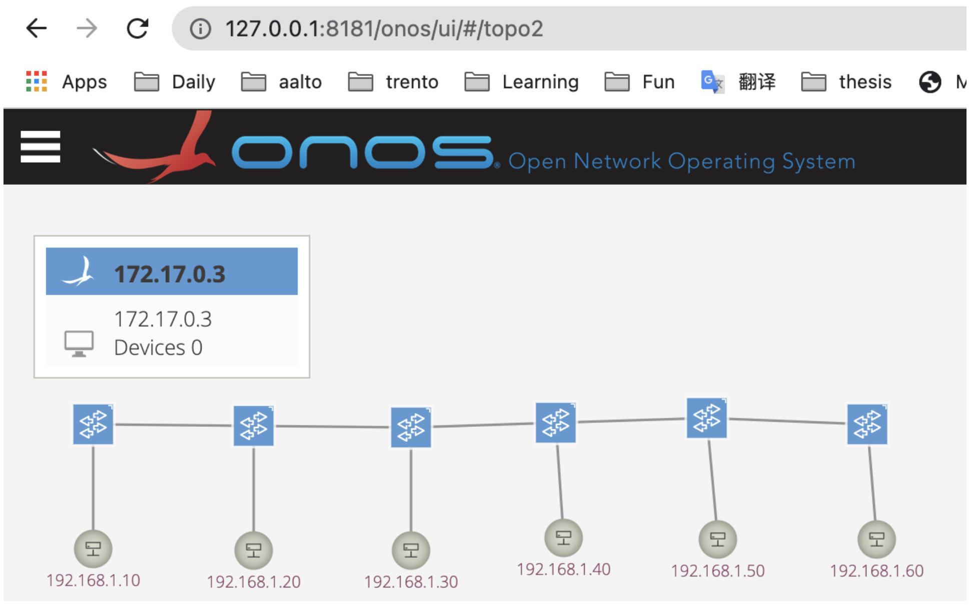
3. Mirror linear and star network topologies

3.1 Mirror the linear network topology

3.1.1 Create the original linear network topology

```
yanjing@yanjingdeMacBook-Pro ~ % create_linear_topo.sh
# 1. Create namespaces
ip netns add h1
ip netns add h2
ip netns add h3
ip netns add h4
ip netns add h5
ip netns add h6

# 2. Create openvswitch
ovs-vsctl add-br s1
ovs-vsctl set bridge s1 protocols=OpenFlow14
ovs-vsctl add-br s2
ovs-vsctl set bridge s2 protocols=OpenFlow14
ovs-vsctl add-br s3
ovs-vsctl set bridge s3 protocols=OpenFlow14
ovs-vsctl add-br s4
ovs-vsctl set bridge s4 protocols=OpenFlow14
ovs-vsctl add-br s5
ovs-vsctl set bridge s5 protocols=OpenFlow14
ovs-vsctl add-br s6
ovs-vsctl set bridge s6 protocols=OpenFlow14

# 3.1 Create vethernet links among switches and hosts
ip link add h1-eth1 type veth peer name s1-eth1
ip link add h2-eth1 type veth peer name s2-eth1
ip link add h3-eth1 type veth peer name s3-eth1
ip link add h4-eth1 type veth peer name s4-eth1
ip link add h5-eth1 type veth peer name s5-eth1
ip link add h6-eth1 type veth peer name s6-eth1

# 3.2 Create vethernet links among switches and switches
ip link add s1-eth2 type veth peer name s2-eth2
ip link add s2-eth3 type veth peer name s3-eth2
ip link add s3-eth3 type veth peer name s4-eth2
ip link add s4-eth3 type veth peer name s5-eth2
ip link add s5-eth3 type veth peer name s6-eth2

# 4. Move host ports into namespaces
ip link set h1-eth1 netns h1
ip link set h2-eth1 netns h2
ip link set h3-eth1 netns h3
ip link set h4-eth1 netns h4
ip link set h5-eth1 netns h5
ip link set h6-eth1 netns h6

# 5. Connect switch ports to OVS
ovs-vsctl add-port s1 s1-eth1
ovs-vsctl add-port s1 s1-eth2
ovs-vsctl add-port s2 s2-eth1
ovs-vsctl add-port s2 s2-eth2
ovs-vsctl add-port s2 s2-eth3
ovs-vsctl add-port s3 s3-eth1
ovs-vsctl add-port s3 s3-eth2
ovs-vsctl add-port s3 s3-eth3
ovs-vsctl add-port s4 s4-eth1
ovs-vsctl add-port s4 s4-eth2
ovs-vsctl add-port s4 s4-eth3
ovs-vsctl add-port s5 s5-eth1
ovs-vsctl add-port s5 s5-eth2
ovs-vsctl add-port s5 s5-eth3
ovs-vsctl add-port s6 s6-eth1
ovs-vsctl add-port s6 s6-eth2

# 6. Connect controller to switch
ovs-vsctl set-controller s1 tcp:172.17.0.3:6653
```

```
ovs-vsctl set-controller s2 tcp:172.17.0.3:6653
ovs-vsctl set-controller s3 tcp:172.17.0.3:6653
ovs-vsctl set-controller s4 tcp:172.17.0.3:6653
ovs-vsctl set-controller s5 tcp:172.17.0.3:6653
ovs-vsctl set-controller s6 tcp:172.17.0.3:6653

# 7.1 Setup ip for hosts
ip netns exec h1 ifconfig h1-eth1 192.168.1.10 hw ether 0E:76:73:DE:95:0B
ip netns exec h1 ifconfig lo up
ip netns exec h2 ifconfig h2-eth1 192.168.1.20 hw ether 46:F1:1F:61:13:BE
ip netns exec h2 ifconfig lo up
ip netns exec h3 ifconfig h3-eth1 192.168.1.30 hw ether C2:CF:6D:1F:04:68
ip netns exec h3 ifconfig lo up
ip netns exec h4 ifconfig h4-eth1 192.168.1.40 hw ether B6:9D:D9:84:A7:64
ip netns exec h4 ifconfig lo up
ip netns exec h5 ifconfig h5-eth1 192.168.1.50 hw ether 42:00:FB:6C:9D:E3
ip netns exec h5 ifconfig lo up
ip netns exec h6 ifconfig h6-eth1 192.168.1.60 hw ether 1A:16:D9:D6:FA:6E
ip netns exec h6 ifconfig lo up

# 7.2 Setup ip for switches
ifconfig s1-eth1 up
ifconfig s1-eth2 up
ifconfig s2-eth1 up
ifconfig s2-eth2 up
ifconfig s2-eth3 up
ifconfig s3-eth1 up
ifconfig s3-eth2 up
ifconfig s3-eth3 up
ifconfig s4-eth1 up
ifconfig s4-eth2 up
ifconfig s4-eth3 up
ifconfig s5-eth1 up
ifconfig s5-eth2 up
ifconfig s5-eth3 up
ifconfig s6-eth1 up
ifconfig s6-eth2 up

# 8 Test connections between hosts
ip netns exec h1 ping -c1 192.168.1.10
ip netns exec h1 ping -c1 192.168.1.20
ip netns exec h1 ping -c1 192.168.1.30
ip netns exec h1 ping -c1 192.168.1.40
ip netns exec h1 ping -c1 192.168.1.50
ip netns exec h1 ping -c1 192.168.1.60
```



3.1.2 Generate the script for mirroring the linear network topology

Change the line 61 code in the detect_topo_shell.py file as follows and execute the script.

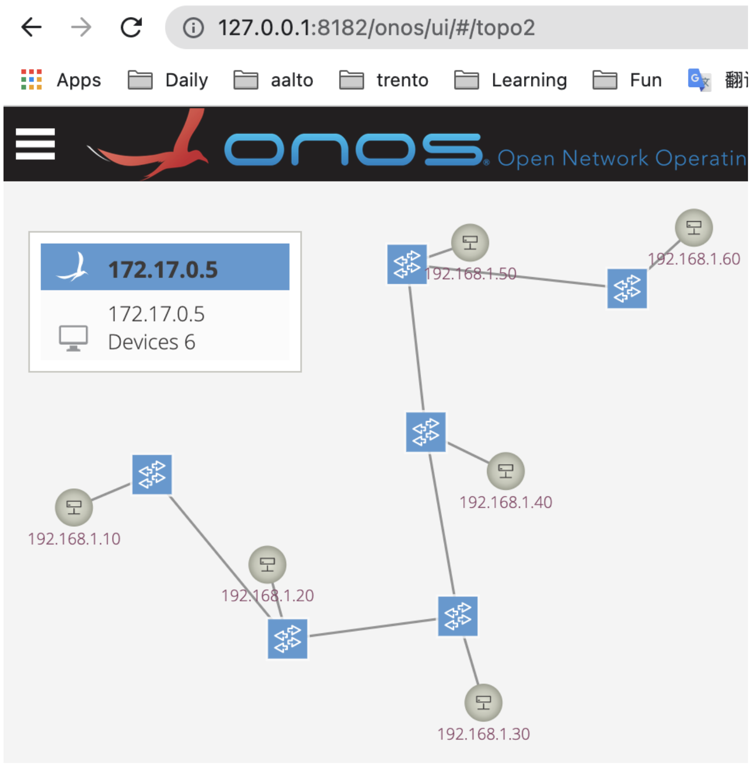
```
with open("./mirror_linear_topo.sh", "w") as filename:
```

```
yanjing@yanjingdeMacBook-Pro ~ % python detect_topo_shell.py
switch_id_list is:  ['of:00004a703083de4a', 'of:00005a5f90b06748',
'of:00009a9728f30b4b', 'of:000096ad8c5ab540', 'of:0000ded0735b3949',
'of:00002276c03afb47']
Each switch info is:  {'switch_name': 's1', 'switch_id':
'of:00004a703083de4a', 'switch_ports': [('1', 's1-eth1'), ('2', 's1-eth2')]}
Each switch info is:  {'switch_name': 's4', 'switch_id':
'of:00005a5f90b06748', 'switch_ports': [('1', 's4-eth1'), ('2', 's4-eth2'),
('3', 's4-eth3')]}
Each switch info is:  {'switch_name': 's5', 'switch_id':
'of:00009a9728f30b4b', 'switch_ports': [('1', 's5-eth1'), ('2', 's5-eth2'),
('3', 's5-eth3')]}
Each switch info is:  {'switch_name': 's6', 'switch_id':
'of:000096ad8c5ab540', 'switch_ports': [('1', 's6-eth1'), ('2', 's6-eth2')]}
Each switch info is:  {'switch_name': 's3', 'switch_id':
'of:0000ded0735b3949', 'switch_ports': [('1', 's3-eth1'), ('2', 's3-eth2'),
('3', 's3-eth3')]}
Each switch info is:  {'switch_name': 's2', 'switch_id':
'of:00002276c03afb47', 'switch_ports': [('1', 's2-eth1'), ('2', 's2-eth2'),
('3', 's2-eth3')]}
Each host info is:  {'host_name': 'h1', 'host_port': 'h1-eth1', 'host_ip':
'192.168.1.10', 'host_mac': '0E:76:73:DE:95:0B', 'host_switch_connection':
{'elementId': 'of:00004a703083de4a', 'port': '1'}}
Each host info is:  {'host_name': 'h2', 'host_port': 'h2-eth1', 'host_ip':
'192.168.1.20', 'host_mac': '46:F1:1F:61:13:BE', 'host_switch_connection':
{'elementId': 'of:00002276c03afb47', 'port': '1'}}
Each host info is:  {'host_name': 'h3', 'host_port': 'h3-eth1', 'host_ip':
'192.168.1.30', 'host_mac': 'C2:CF:6D:1F:04:68', 'host_switch_connection':
{'elementId': 'of:0000ded0735b3949', 'port': '1'}}
Each host info is:  {'host_name': 'h4', 'host_port': 'h4-eth1', 'host_ip':
'192.168.1.50', 'host_mac': '42:00:FB:6C:9D:E3', 'host_switch_connection':
{'elementId': 'of:00009a9728f30b4b', 'port': '1'}}
```

```
Each host info is:  {'host_name': 'h5', 'host_port': 'h5-eth1', 'host_ip':
'192.168.1.40', 'host_mac': 'B6:9D:D9:84:A7:64', 'host_switch_connection':
{'elementId': 'of:00005a5f90b06748', 'port': '1'}}
Each host info is:  {'host_name': 'h6', 'host_port': 'h6-eth1', 'host_ip':
'192.168.1.60', 'host_mac': '1A:16:D9:D6:FA:6E', 'host_switch_connection':
{'elementId': 'of:000096ad8c5ab540', 'port': '1'}}
Connection info among switches is:  [['s4-eth3', 's5-eth2'], ['s3-eth3',
's4-eth2'], ['s2-eth3', 's3-eth2'], ['s5-eth3', 's6-eth2'], ['s1-eth2',
's2-eth2']]

yanjing@yanjingdeMacBook-Pro ~ % ls mirror_linear_topo.sh
mirror_linear_topo.sh
```

3.1.3 Mirror the linear network topology

Execute the script generated in the previous step in the mirroring environment and check the WEB GUI of onos_mirror container



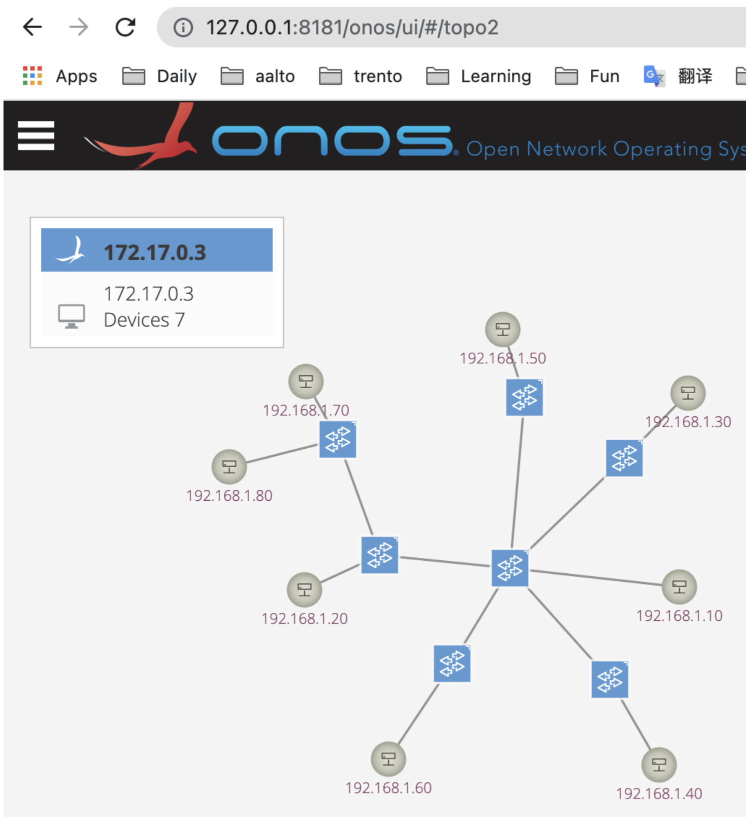
3.2 Mirror the star network topology

3.2.1 Create the original star network topology

```
yanjing@yanjingdeMacBook-Pro ~ % create_star_topo.sh
# 1. Create namespaces
ip netns add h1
ip netns add h2
ip netns add h3
ip netns add h4
ip netns add h5
ip netns add h6
ip netns add h7
ip netns add h8
```

```
# 2. Create openvswitch
ovs-vsctl add-br s1
ovs-vsctl set bridge s1 protocols=OpenFlow14
ovs-vsctl add-br s2
ovs-vsctl set bridge s2 protocols=OpenFlow14
ovs-vsctl add-br s3
ovs-vsctl set bridge s3 protocols=OpenFlow14
ovs-vsctl add-br s4
ovs-vsctl set bridge s4 protocols=OpenFlow14
ovs-vsctl add-br s5
ovs-vsctl set bridge s5 protocols=OpenFlow14
ovs-vsctl add-br s6
ovs-vsctl set bridge s6 protocols=OpenFlow14
ovs-vsctl add-br s7
ovs-vsctl set bridge s7 protocols=OpenFlow14

# 3.1 Create vethernet links among switches and hosts
ip link add h1-eth1 type veth peer name s1-eth1
ip link add h2-eth1 type veth peer name s2-eth1
ip link add h3-eth1 type veth peer name s3-eth1
ip link add h4-eth1 type veth peer name s4-eth1
ip link add h5-eth1 type veth peer name s5-eth1
ip link add h6-eth1 type veth peer name s6-eth1
ip link add h7-eth1 type veth peer name s7-eth1
ip link add h8-eth1 type veth peer name s7-eth2

# 3.2 Create vethernet links among switches and switches
ip link add s2-eth2 type veth peer name s1-eth2
ip link add s3-eth2 type veth peer name s1-eth3
ip link add s4-eth2 type veth peer name s1-eth4
ip link add s5-eth2 type veth peer name s1-eth5
ip link add s6-eth2 type veth peer name s1-eth6
ip link add s7-eth3 type veth peer name s2-eth3

# 4. Move host ports into namespaces
ip link set h1-eth1 netns h1
ip link set h2-eth1 netns h2
ip link set h3-eth1 netns h3
ip link set h4-eth1 netns h4
ip link set h5-eth1 netns h5
ip link set h6-eth1 netns h6
ip link set h7-eth1 netns h7
ip link set h8-eth1 netns h8

# 5. Connect switch ports to OVS
ovs-vsctl add-port s1 s1-eth1
ovs-vsctl add-port s1 s1-eth2
ovs-vsctl add-port s1 s1-eth3
ovs-vsctl add-port s1 s1-eth4
ovs-vsctl add-port s1 s1-eth5
ovs-vsctl add-port s1 s1-eth6
ovs-vsctl add-port s2 s2-eth1
ovs-vsctl add-port s2 s2-eth2
ovs-vsctl add-port s2 s2-eth3
ovs-vsctl add-port s3 s3-eth1
ovs-vsctl add-port s3 s3-eth2
ovs-vsctl add-port s4 s4-eth1
ovs-vsctl add-port s4 s4-eth2
ovs-vsctl add-port s5 s5-eth1
ovs-vsctl add-port s5 s5-eth2
ovs-vsctl add-port s6 s6-eth1
```

```
ovs-vsctl add-port s6 s6-eth2
ovs-vsctl add-port s7 s7-eth1
ovs-vsctl add-port s7 s7-eth2
ovs-vsctl add-port s7 s7-eth3

# 6. Connect controller to switch
ovs-vsctl set-controller s1 tcp:172.17.0.3:6653
ovs-vsctl set-controller s2 tcp:172.17.0.3:6653
ovs-vsctl set-controller s3 tcp:172.17.0.3:6653
ovs-vsctl set-controller s4 tcp:172.17.0.3:6653
ovs-vsctl set-controller s5 tcp:172.17.0.3:6653
ovs-vsctl set-controller s6 tcp:172.17.0.3:6653
ovs-vsctl set-controller s7 tcp:172.17.0.3:6653

# 7.1 Setup ip for hosts
ip netns exec h1 ifconfig h1-eth1 192.168.1.10 hw ether 0E:76:73:DE:95:0B
ip netns exec h1 ifconfig lo up
ip netns exec h2 ifconfig h2-eth1 192.168.1.20 hw ether 46:F1:1F:61:13:BE
ip netns exec h2 ifconfig lo up
ip netns exec h3 ifconfig h3-eth1 192.168.1.30 hw ether C2:CF:6D:1F:04:68
ip netns exec h3 ifconfig lo up
ip netns exec h4 ifconfig h4-eth1 192.168.1.40 hw ether B6:9D:D9:84:A7:64
ip netns exec h4 ifconfig lo up
ip netns exec h5 ifconfig h5-eth1 192.168.1.50 hw ether 42:00:FB:6C:9D:E3
ip netns exec h5 ifconfig lo up
ip netns exec h6 ifconfig h6-eth1 192.168.1.60 hw ether 1A:16:D9:D6:FA:6E
ip netns exec h6 ifconfig lo up
ip netns exec h7 ifconfig h7-eth1 192.168.1.70 hw ether 89:00:03:6C:9D:E3
ip netns exec h7 ifconfig lo up
ip netns exec h8 ifconfig h8-eth1 192.168.1.80 hw ether 9A:16:D9:D6:FA:4C
ip netns exec h8 ifconfig lo up

# 7.2 Setup ip for switches
ifconfig s1-eth1 up
ifconfig s1-eth2 up
ifconfig s1-eth3 up
ifconfig s1-eth4 up
ifconfig s1-eth5 up
ifconfig s1-eth6 up
ifconfig s2-eth1 up
ifconfig s2-eth2 up
ifconfig s2-eth3 up
ifconfig s3-eth1 up
ifconfig s3-eth2 up
ifconfig s4-eth1 up
ifconfig s4-eth2 up
ifconfig s5-eth1 up
ifconfig s5-eth2 up
ifconfig s6-eth1 up
ifconfig s6-eth2 up
ifconfig s7-eth1 up
ifconfig s7-eth2 up
ifconfig s7-eth3 up

# 8 Test connections between hosts
ip netns exec h1 ping -c1 192.168.1.10
ip netns exec h1 ping -c1 192.168.1.20
ip netns exec h1 ping -c1 192.168.1.30
ip netns exec h1 ping -c1 192.168.1.40
ip netns exec h1 ping -c1 192.168.1.50
ip netns exec h1 ping -c1 192.168.1.60
ip netns exec h1 ping -c1 192.168.1.70
ip netns exec h1 ping -c1 192.168.1.80
```



3.2.2 Generate the script for mirroring the star network topology

Change the line 61 code in the detect_topo_shell.py file as follows and execute the script.

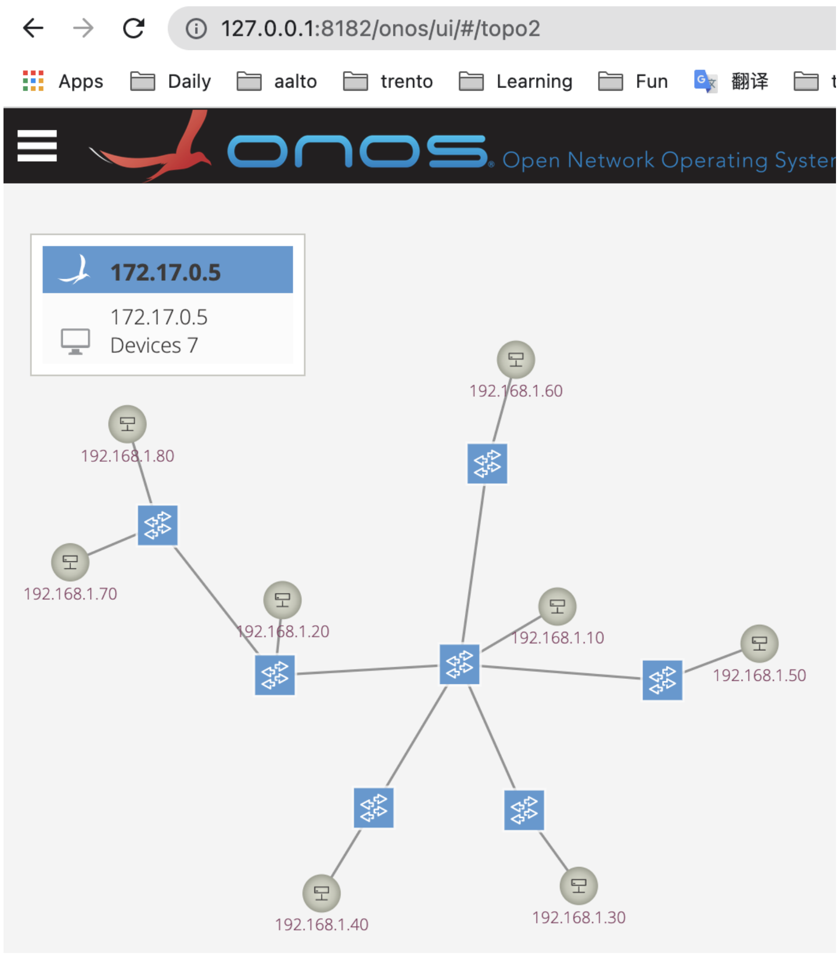
```
with open("./mirror_star_topo.sh", "w") as filename:
```

```
yanjing@yanjingdeMacBook-Pro ~ % python detect_topo_shell.py
switch_id_list is:  ['of:0000d2c8f727c944', 'of:0000c68a2f51c345',
'of:000016bec42e7c4a', 'of:0000e63aaf7f2449', 'of:0000aa47ece8ed4f',
'of:0000fa7499777449', 'of:00002e08d8aab64e']
Each switch info is:  {'switch_name': 's6', 'switch_id':
'of:0000d2c8f727c944', 'switch_ports': [('1', 's6-eth1'), ('2', 's6-eth2')]}
Each switch info is:  {'switch_name': 's1', 'switch_id':
'of:0000c68a2f51c345', 'switch_ports': [('1', 's1-eth1'), ('2', 's1-eth2'),
('3', 's1-eth3'), ('4', 's1-eth4'), ('5', 's1-eth5'), ('6', 's1-eth6')]}
Each switch info is:  {'switch_name': 's7', 'switch_id':
'of:000016bec42e7c4a', 'switch_ports': [('1', 's7-eth1'), ('2', 's7-eth2'),
('3', 's7-eth3')]}
Each switch info is:  {'switch_name': 's5', 'switch_id':
'of:0000e63aaf7f2449', 'switch_ports': [('1', 's5-eth1'), ('2', 's5-eth2')]}
Each switch info is:  {'switch_name': 's2', 'switch_id':
'of:0000aa47ece8ed4f', 'switch_ports': [('1', 's2-eth1'), ('2', 's2-eth2'),
('3', 's2-eth3')]}
Each switch info is:  {'switch_name': 's3', 'switch_id':
'of:0000fa7499777449', 'switch_ports': [('1', 's3-eth1'), ('2', 's3-eth2')]}
Each switch info is:  {'switch_name': 's4', 'switch_id':
'of:00002e08d8aab64e', 'switch_ports': [('1', 's4-eth1'), ('2', 's4-eth2')]}
```

```
Each host info is:  {'host_name': 'h1', 'host_port': 'h1-eth1', 'host_ip':
'192.168.1.10', 'host_mac': '0E:76:73:DE:95:0B', 'host_switch_connection':
{'elementId': 'of:0000c68a2f51c345', 'port': '1'}}
Each host info is:  {'host_name': 'h2', 'host_port': 'h2-eth1', 'host_ip':
'192.168.1.20', 'host_mac': '46:F1:1F:61:13:BE', 'host_switch_connection':
{'elementId': 'of:0000aa47ece8ed4f', 'port': '1'}}
Each host info is:  {'host_name': 'h3', 'host_port': 'h3-eth1', 'host_ip':
'192.168.1.30', 'host_mac': 'C2:CF:6D:1F:04:68', 'host_switch_connection':
{'elementId': 'of:0000fa7499777449', 'port': '1'}}
Each host info is:  {'host_name': 'h4', 'host_port': 'h4-eth1', 'host_ip':
'192.168.1.80', 'host_mac': '9A:16:D9:D6:FA:4C', 'host_switch_connection':
{'elementId': 'of:000016bec42e7c4a', 'port': '2'}}
Each host info is:  {'host_name': 'h5', 'host_port': 'h5-eth1', 'host_ip':
'192.168.1.50', 'host_mac': '42:00:FB:6C:9D:E3', 'host_switch_connection':
{'elementId': 'of:0000e63aaf7f2449', 'port': '1'}}
Each host info is:  {'host_name': 'h6', 'host_port': 'h6-eth1', 'host_ip':
'192.168.1.40', 'host_mac': 'B6:9D:D9:84:A7:64', 'host_switch_connection':
{'elementId': 'of:00002e08d8aab64e', 'port': '1'}}
Each host info is:  {'host_name': 'h7', 'host_port': 'h7-eth1', 'host_ip':
'192.168.1.70', 'host_mac': 'D2:5A:D9:B1:5B:76', 'host_switch_connection':
{'elementId': 'of:000016bec42e7c4a', 'port': '1'}}
Each host info is:  {'host_name': 'h8', 'host_port': 'h8-eth1', 'host_ip':
'192.168.1.60', 'host_mac': '1A:16:D9:D6:FA:6E', 'host_switch_connection':
{'elementId': 'of:0000d2c8f727c944', 'port': '1'}}
Connection info among switches is:  [['s2-eth3', 's7-eth3'], ['s1-eth2',
's2-eth2'], ['s1-eth6', 's6-eth2'], ['s1-eth5', 's5-eth2'], ['s1-eth3',
's3-eth2'], ['s1-eth4', 's4-eth2']]

yanjing@yanjingdeMacBook-Pro ~ % ls mirror_star_topo.sh
mirror_star_topo.sh
```

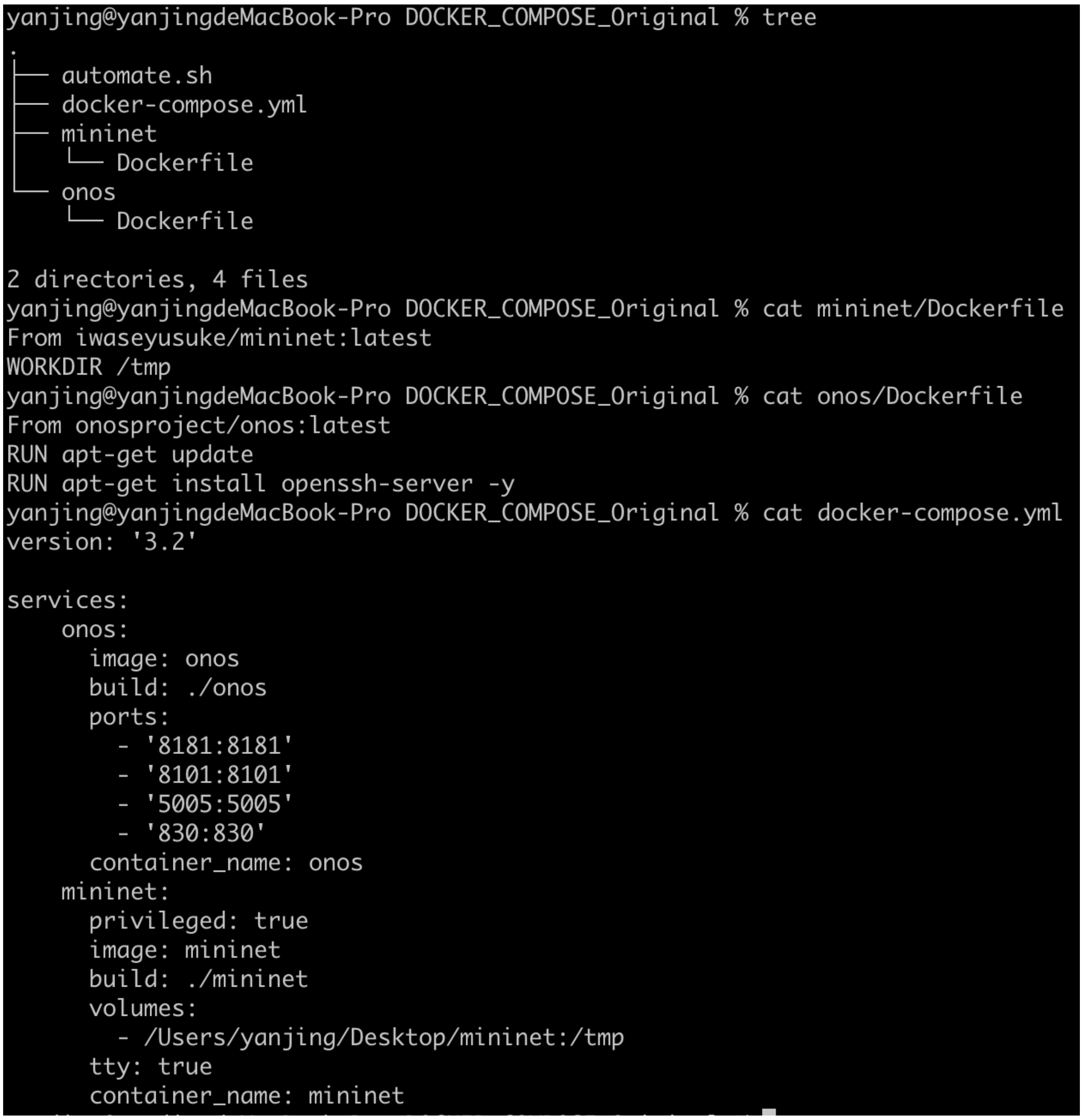
3.2.3 Mirror the star network topology

Execute the script generated in the previous step in the mirroring environment and check the WEB GUI of onos_mirror container


127.0.0.1:8182/onos/ui/#/topo2
Apps
Daily
aalto
trento
Learning
Fun
翻译
onos
Open Network Operating Syster
172.17.0.5
172.17.0.5
Devices 7
192.168.1.60
192.168.1.80
192.168.1.70
192.168.1.20
192.168.1.10
192.168.1.50
192.168.1.40
192.168.1.30

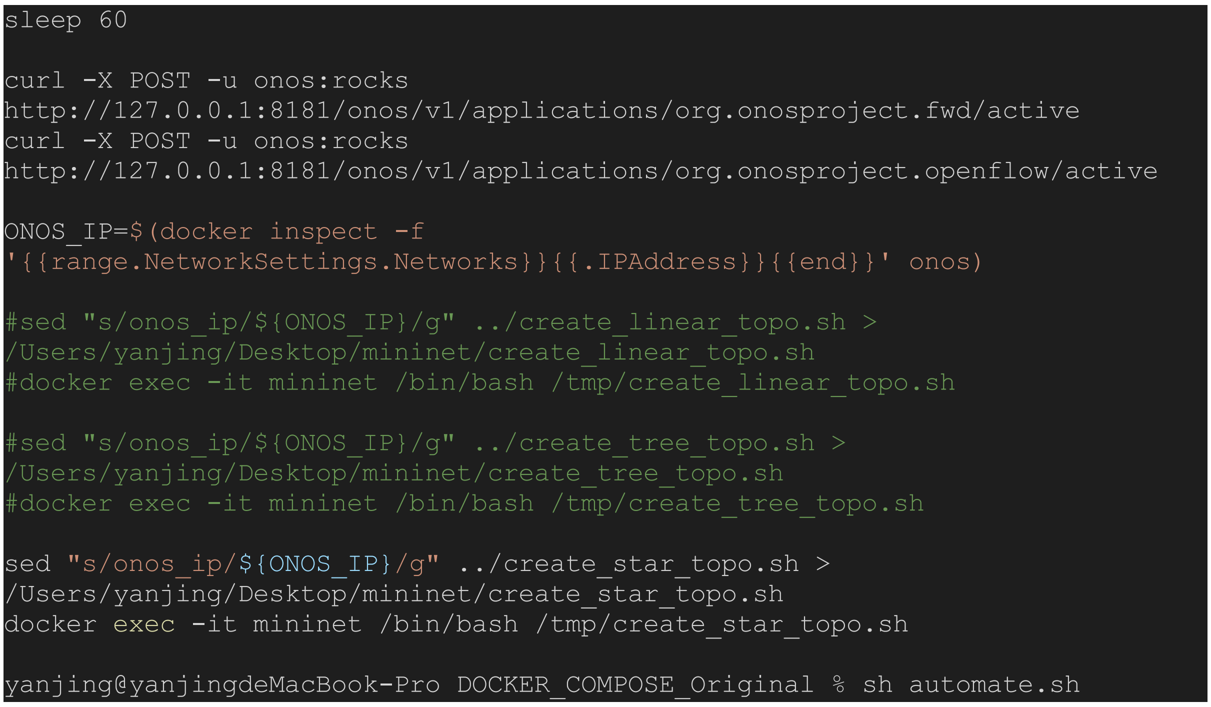
4. Automation network topology creation and mirroring docker-compose

4.1 Scripts and execution for automating network topology creation



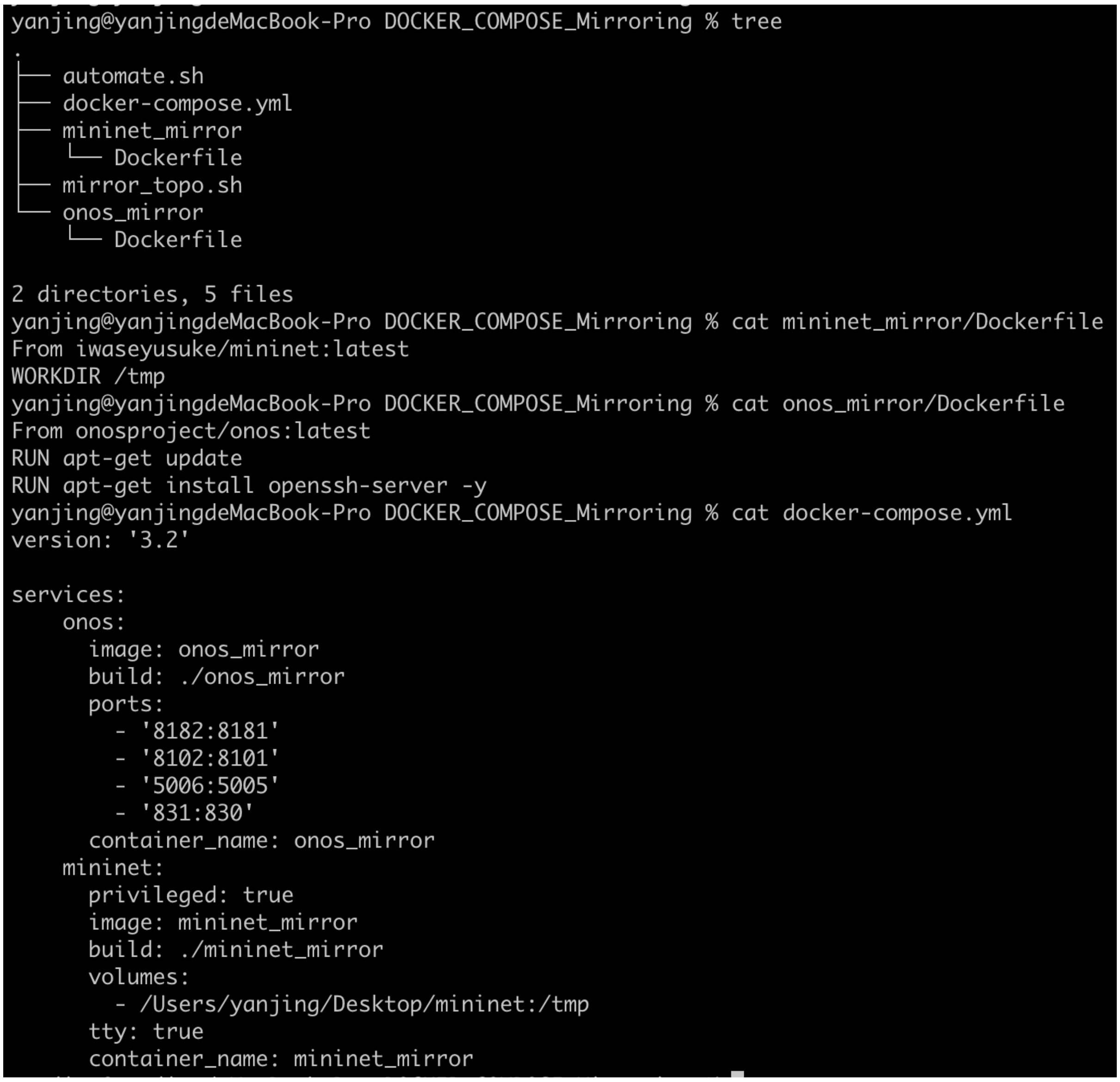
```
yanjing@yanjingdeMacBook-Pro DOCKER_COMPOSE_Original % tree
.
├── automate.sh
├── docker-compose.yml
├── mininet
│   └── Dockerfile
└── onos
    └── Dockerfile

2 directories, 4 files
yanjing@yanjingdeMacBook-Pro DOCKER_COMPOSE_Original % cat mininet/Dockerfile
From iwaseyusuke/mininet:latest
WORKDIR /tmp
yanjing@yanjingdeMacBook-Pro DOCKER_COMPOSE_Original % cat onos/Dockerfile
From onosproject/onos:latest
RUN apt-get update
RUN apt-get install openssh-server -y
yanjing@yanjingdeMacBook-Pro DOCKER_COMPOSE_Original % cat docker-compose.yml
version: '3.2'

services:
    onos:
      image: onos
      build: ./onos
      ports:
        - '8181:8181'
        - '8101:8101'
        - '5005:5005'
        - '830:830'
      container_name: onos
    mininet:
      privileged: true
      image: mininet
      build: ./mininet
      volumes:
        - /Users/yanjing/Desktop/mininet:/tmp
      tty: true
      container_name: mininet
```

```
yanjing@yanjingdeMacBook-Pro DOCKER_COMPOSE_Original % cat automate.sh
docker container stop mininet
docker container stop onos
docker container prune -f
docker image rm mininet
docker image rm onos

docker-compose up -d
```



```
sleep 60

curl -X POST -u onos:rocks
http://127.0.0.1:8181/onos/v1/applications/org.onosproject.fwd/active
curl -X POST -u onos:rocks
http://127.0.0.1:8181/onos/v1/applications/org.onosproject.openflow/active

ONOS_IP=$(docker inspect -f
'{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' onos)

#sed "s/onos_ip/${ONOS_IP}/g" ../create_linear_topo.sh >
/Users/yanjing/Desktop/mininet/create_linear_topo.sh
#docker exec -it mininet /bin/bash /tmp/create_linear_topo.sh

#sed "s/onos_ip/${ONOS_IP}/g" ../create_tree_topo.sh >
/Users/yanjing/Desktop/mininet/create_tree_topo.sh
#docker exec -it mininet /bin/bash /tmp/create_tree_topo.sh

sed "s/onos_ip/${ONOS_IP}/g" ../create_star_topo.sh >
/Users/yanjing/Desktop/mininet/create_star_topo.sh
docker exec -it mininet /bin/bash /tmp/create_star_topo.sh

yanjing@yanjingdeMacBook-Pro DOCKER_COMPOSE_Original % sh automate.sh
```

4.2 Scripts and execution for automating network topology mirroring



```
yanjing@yanjingdeMacBook-Pro DOCKER_COMPOSE_Mirroring % tree
.
├── automate.sh
├── docker-compose.yml
├── mininet_mirror
│   └── Dockerfile
├── mirror_topo.sh
└── onos_mirror
    └── Dockerfile

2 directories, 5 files
yanjing@yanjingdeMacBook-Pro DOCKER_COMPOSE_Mirroring % cat mininet_mirror/Dockerfile
From iwaseyusuke/mininet:latest
WORKDIR /tmp
yanjing@yanjingdeMacBook-Pro DOCKER_COMPOSE_Mirroring % cat onos_mirror/Dockerfile
From onosproject/onos:latest
RUN apt-get update
RUN apt-get install openssh-server -y
yanjing@yanjingdeMacBook-Pro DOCKER_COMPOSE_Mirroring % cat docker-compose.yml
version: '3.2'

services:
    onos:
      image: onos_mirror
      build: ./onos_mirror
      ports:
        - '8182:8181'
        - '8102:8101'
        - '5006:5005'
        - '831:830'
      container_name: onos_mirror
    mininet:
      privileged: true
      image: mininet_mirror
      build: ./mininet_mirror
      volumes:
        - /Users/yanjing/Desktop/mininet:/tmp
      tty: true
      container_name: mininet_mirror
```

```
yanjing@yanjingdeMacBook-Pro DOCKER_COMPOSE_Mirroring % cat automate.sh
docker container stop mininet_mirror
docker container stop onos_mirror
docker container prune -f
docker image rm mininet_mirror
docker image rm onos_mirror
rm -rf mirror_topo.sh
rm -rf /Users/yanjing/Desktop/mininet/mirror_topo.sh

docker-compose up -d
sleep 60

curl -X POST -u onos:rocks
http://127.0.0.1:8182/onos/v1/applications/org.onosproject.fwd/active
curl -X POST -u onos:rocks
http://127.0.0.1:8182/onos/v1/applications/org.onosproject.openflow/active
```

```
python3 ../detect_topo_shell.py

ONOS_IP=$(docker inspect -f
'{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' onos_mirror)

sed "s/onos_ip/${ONOS_IP}/g" mirror_topo.sh >
/Users/yanjing/Desktop/mininet/mirror_topo.sh
docker exec -it mininet_mirror /bin/bash /tmp/mirror_topo.sh
```

5. Final discussion

Indeed, one more function can be added to this research project. However, as my master thesis internship has already started, limited by time and effort, I won't implement this function detailedly. The basic ideas will be introduced as follows.

For the original SDN-based network topology, traffic rules can be set through ONOS to mirror or block specific network traffic and so on, which can be implemented through the ONOS intent API. Similarly, the traffic rules can be detected through the RestAPI provided by the ONOS controller and mirrored in the mirroring SDN-based network topology.

Furthermore, network traffic capturing and replaying can be improved. I will try to work on it in the future if time and effort allow, or others who are interested in this project could continue further research. Currently, I have no idea how to mirror the network traffic in a real-time way. To be more specific, in this project, it will be more perfect that all the network traffic generated in the original SDN-based network topology should be mirrored in the mirroring network topology immediately. To implement this, two issues are needed to be solved. The first one is detect all the connections among all the devices in the original SDN-based network topology, and capture the network traffic once there is, then share it to the mirroring SDN-based network topology, finally replay all the network traffic immediately. The second one is how to replay TCP-based network traffic.

Finally, the research project is still imperfect. However, I have learned a lot from this project, including researching, implementation and debugging. Also, many thanks to the support from my supervisor. Without his advice and guidance, this research project can not be finished smoothly.

Reference

[1] https://en.wikipedia.org/wiki/Digital_twin