2023 SDN-NFV Project1

312552017 董家鳴

Part1 : Answer Questions

1. When ONOS activates "org.onosproject.openflow," what are the APPs which it also activates?

First, I deactivate all apps. And I activate the *openflow* to get the result. According to the Figure 1, the apps activated are:

- **org.onosproject.optical-model**
- **org.onosproject.hostprovider**
- **org.onosproject.lldpprovider**
- **org.onosproject.openflow-base**

Figure 1: Apps activated by openflow

2. After activating ONOS and running the commands on P.17 and P.20. Will H1 ping H2 successfully? Why or why not? **Answer: No**

Since no flows installed on the data-plane, which forward the traffic appropriately. ONOS comes with a simple Reactive Forwarding app that installs forwarding flows on demand, but this application is not activated by default [1].

Therefore, under the default app, ping will fail, as shown in Figure 2. After enabling app org.onosproject.fwd, ping will succeed, as shown in Figure 3.

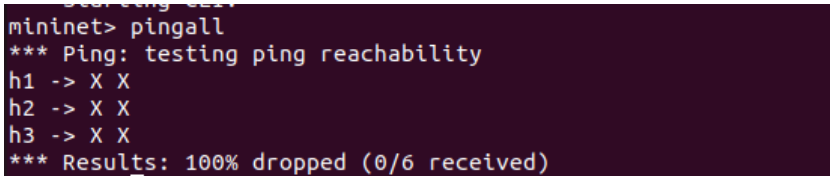

Figure 2: failed ping

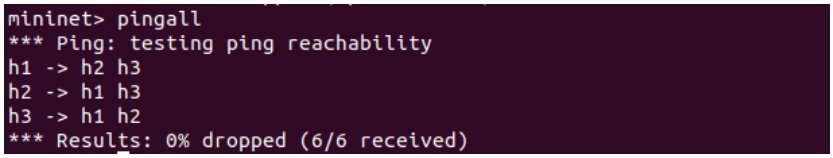

Figure 3: Successful ping

3. Which TCP port the controller listens for the OpenFlow connection request from the switch? **Answer: 6653**

Let us observe the packets through wireshark. From Figure 4 we can see the TCP packets of the openflow connection, and from Figure 5 we can see the ephemeral source port of the switch (from which the switch established its connection to the controller), so we can infer port 6653 TCP port for controller listens for.

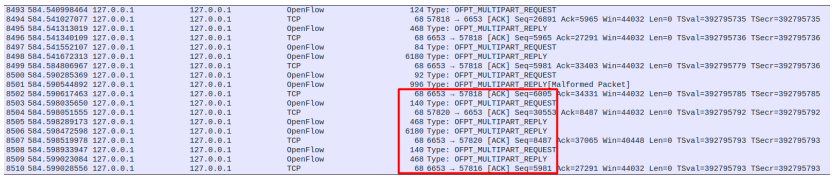

Figure 4: Wireshark packets

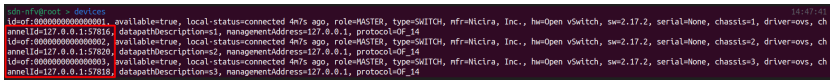

Figure 5: Switch devices

4. In question 3, which APP enables the controller to listen on the TCP port?

Answer: org.onosproject.openflowbase

Because the openflow app opened four other apps at one time, I first turned off the openflow app and found that the 6653 port was still open, as shown in Figure 6. Next, I turned off the openflow-base app and found that the 6653 port was closed, as shown in Figure 7, and the wireshark packets could not be transmitted, as shown in Figure 8, so it was inferred that the openflow-base app enables the controller to listen on the TCP port. The closing process is shown in Figure 9.

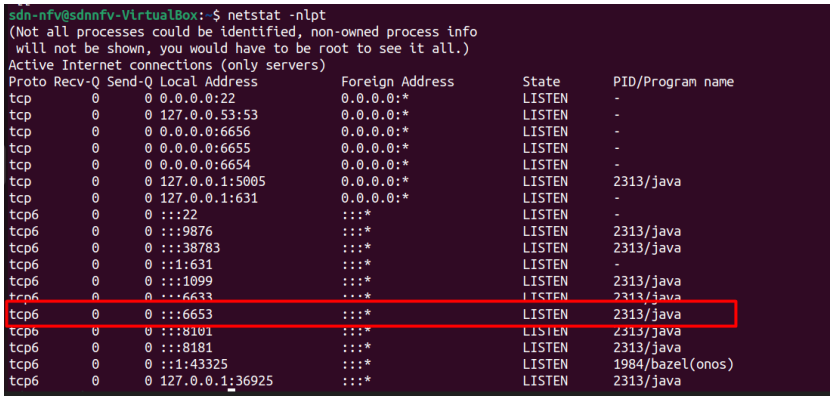

Figure 6: Wireshark packets

Figure 7: Switch devices

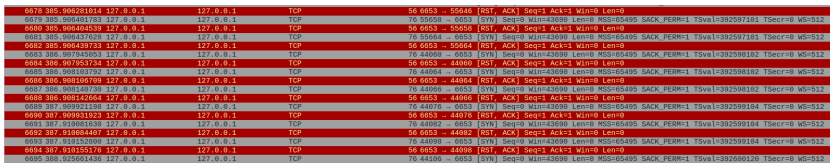

Figure 8: Switch devices

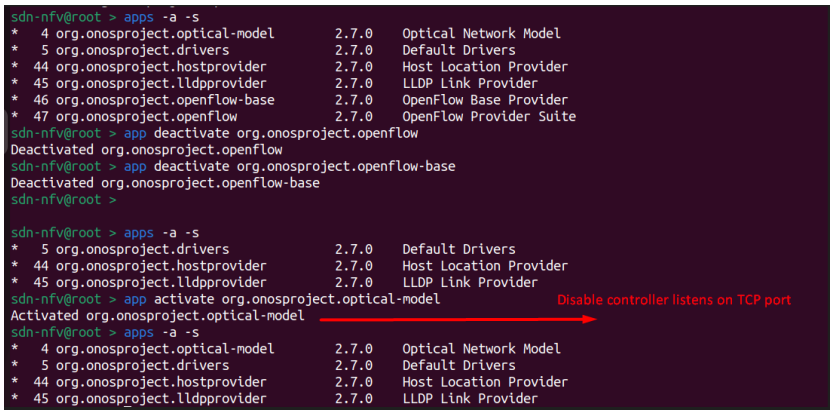

Figure 9: Switch devices

Part2 : Create a custom Topology

Write a Python script to build the following topology

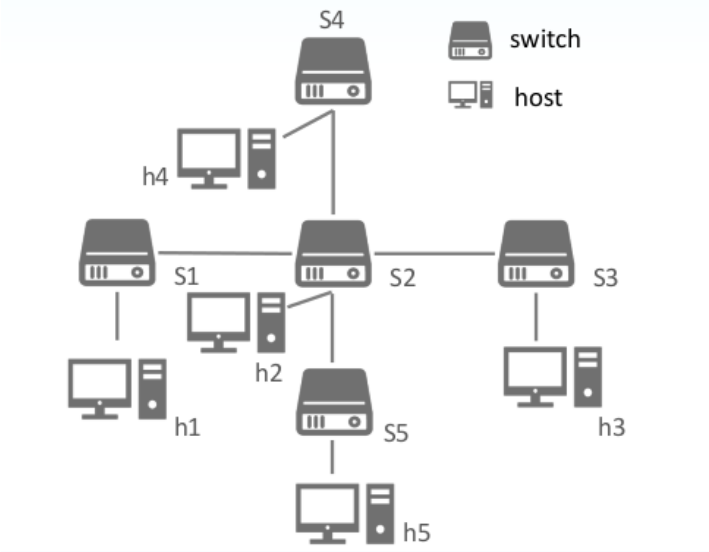

Figure 10: Target topology

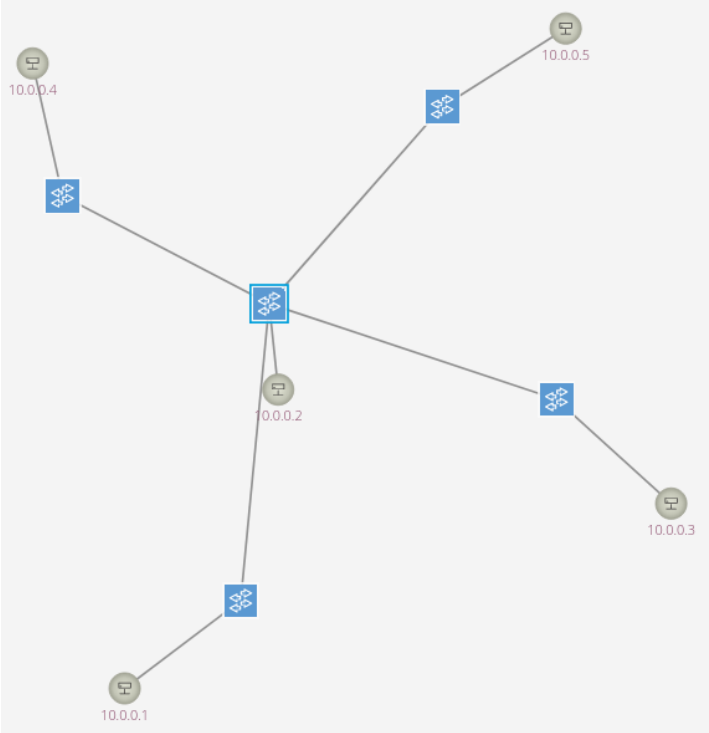

Figure 11: My topology on GUI

```python
from mininet.topo import Topo

class Project1_Topo_312552017( Topo ):
    def __init__( self ):
        Topo.__init__( self )
        # Add hosts
        h1 = self.addHost( 'h1' )
        h2 = self.addHost( 'h2' )
        h3 = self.addHost( 'h3' )
        h4 = self.addHost( 'h4' )
        h5 = self.addHost( 'h5' )

        # Add switches
        s1 = self.addSwitch( 's1' )
        s2 = self.addSwitch( 's2' )
        s3 = self.addSwitch( 's3' )
        s4 = self.addSwitch( 's4' )
        s5 = self.addSwitch( 's5' )

        # Add links, host to switch
        self.addLink( h1, s1 )
        self.addLink( h2, s2 )
        self.addLink( h3, s3 )
        self.addLink( h4, s4 )
        self.addLink( h5, s5 )

        # Add links, switch to switch
        self.addLink( s1, s2 )
        self.addLink( s3, s2 )
        self.addLink( s4, s2 )
        self.addLink( s5, s2 )

topos = { 'topo_part2_312552017': Project1_Topo_312552017 }
```

Figure 12: Python script for part 2

Part3 : Statically assign Hosts IP Address in Mininet

The format for manual assignment of host IP address is in Figure 13 shown. All the components are shown in Figure 14 by using *dump*. And the *ifconfig* of all hosts are shown in Figure 15, 16, 17.

- Format for manual assignment of host IP address:
  - 192.168.0.0/27
  - netmask 255.255.255.224

Host	IP Address
h1	192.168.0.1
h2	192.168.0.2
	...

Figure 13: IP format

```
mininet> dump
<Host h1: h1-eth0:192.168.0.1 pid=12736>
<Host h2: h2-eth0:192.168.0.2 pid=12738>
<Host h3: h3-eth0:192.168.0.3 pid=12740>
<Host h4: h4-eth0:192.168.0.4 pid=12742>
<Host h5: h5-eth0:192.168.0.5 pid=12744>
<OVSSwitch{'protocols': 'OpenFlow14'} s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=12749>
<OVSSwitch{'protocols': 'OpenFlow14'} s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None,s2-eth4:None,s2-eth5:None pid=12752>
<OVSSwitch{'protocols': 'OpenFlow14'} s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None pid=12755>
<OVSSwitch{'protocols': 'OpenFlow14'} s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None pid=12758>
<OVSSwitch{'protocols': 'OpenFlow14'} s5: lo:127.0.0.1,s5-eth1:None,s5-eth2:None pid=12761>
<RemoteController{'ip': '127.0.0.1', 'port': 6653} c0: 127.0.0.1:6653 pid=12730>
```

Figure 14: mininet dump

```
mininet> h1 ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.1  netmask 255.255.255.224  broadcast 192.168.0.31
        inet6 fe80::647b:60ff:fe2d:6e5b  prefixlen 64  scopeid 0x20<link>
        ether 66:7b:60:2d:6e:5b  txqueuelen 1000  (Ethernet)
        RX packets 137  bytes 17740 (17.7 KB)
        RX errors 0  dropped 88  overruns 0  frame 0
        TX packets 27  bytes 1986 (1.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

mininet> h2 ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.2  netmask 255.255.255.224  broadcast 192.168.0.31
        inet6 fe80::c429:deff:febe:8735  prefixlen 64  scopeid 0x20<link>
        ether c6:29:de:be:87:35  txqueuelen 1000  (Ethernet)
        RX packets 141  bytes 18296 (18.2 KB)
        RX errors 0  dropped 92  overruns 0  frame 0
        TX packets 27  bytes 1986 (1.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 15: h1 and h2 interface config

```
mininet> h3 ifconfig
h3-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.3  netmask 255.255.255.224  broadcast 192.168.0.31
        inet6 fe80::b402:2ff:fe4f:5b00  prefixlen 64  scopeid 0x20<link>
        ether b6:02:02:4f:5b:00  txqueuelen 1000  (Ethernet)
        RX packets 145  bytes 18852 (18.8 KB)
        RX errors 0  dropped 96  overruns 0  frame 0
        TX packets 27  bytes 1986 (1.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

mininet> h4 ifconfig
h4-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.4  netmask 255.255.255.224  broadcast 192.168.0.31
        inet6 fe80::40de:2dff:fe33:1750  prefixlen 64  scopeid 0x20<link>
        ether 42:de:2d:33:17:50  txqueuelen 1000  (Ethernet)
        RX packets 149  bytes 19408 (19.4 KB)
        RX errors 0  dropped 100  overruns 0  frame 0
        TX packets 27  bytes 1986 (1.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 16: h3 and h4 interface config

```
mininet> h5 ifconfig
h5-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.5  netmask 255.255.255.224  broadcast 192.168.0.31
        inet6 fe80::5898:53ff:fe45:cba8  prefixlen 64  scopeid 0x20<link>
        ether 5a:98:53:45:cb:a8  txqueuelen 1000  (Ethernet)
        RX packets 150  bytes 19498 (19.4 KB)
        RX errors 0  dropped 100  overruns 0  frame 0
        TX packets 27  bytes 1986 (1.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 17: h5 interface config

```python
from mininet.topo import Topo

class Project1_Topo_312552017( Topo ):
    def __init__( self ):
        Topo.__init__( self )

        # IP Format
        base_ip = '192.168.0.{}'
        subnet = '/27'

        # Add hosts
        h1 = self.addHost( 'h1', ip=base_ip.format(1) + subnet)
        h2 = self.addHost( 'h2', ip=base_ip.format(2) + subnet)
        h3 = self.addHost( 'h3', ip=base_ip.format(3) + subnet)
        h4 = self.addHost( 'h4', ip=base_ip.format(4) + subnet)
        h5 = self.addHost( 'h5', ip=base_ip.format(5) + subnet)

        # Add switches
        s1 = self.addSwitch( 's1' )
        s2 = self.addSwitch( 's2' )
        s3 = self.addSwitch( 's3' )
        s4 = self.addSwitch( 's4' )
        s5 = self.addSwitch( 's5' )

        # Add links, host to switch
        self.addLink( h1, s1 )
        self.addLink( h2, s2 )
        self.addLink( h3, s3 )
        self.addLink( h4, s4 )
        self.addLink( h5, s5 )

        # Add links, switch to switch
        self.addLink( s1, s2 )
        self.addLink( s3, s2 )
        self.addLink( s4, s2 )
        self.addLink( s5, s2 )

topos = { 'topo_part3_312552017': Project1_Topo_312552017 }
```

Figure 18: Python script for part 3

What you' ve learned or solved

1. **An app may require other apps.** For example, when we want to activate the *org.onosproject.openflow* app, it automatically activates other apps.

2. **Reactive Forwarding.** Forwarding application is not activated by default. Every time

we open ONOS, we need to activate *org.onosproject.fwd* app and use ping (ARP request and ICMP Echo request) to identify the MAC address of the remote computer and confirm whether the target computer is accessible [2]. In addition, we can also see how forward works through open source code [3].

3. Port that the controller and switches listens for. The managementAddress of all switches is the same in the Figure 5. This means that any communication between ONOS and the switch will take place at the IP address of ONOS and managementAddress, besides, the individual switches are distinguished by the port number in the channelId field [4].

References

[1] ONOS Wiki. *Basic ONOS Tutorial*. 2019. URL: `https://wiki.onosproject.org/display/ONOS/Basic+ONOS+Tutorial`.

[2] Sandip Roy. *Protocols Used for PING*. URL: `https://www.baeldung.com/cs/protocols-ping`.

[3] ONOS Project. *onos*. URL: `https://github.com/opennetworkinglab/onos/tree/master/apps/fwd/src/main/java/org/onosproject/fwd`.

[4] mirantis. *Software Defined Networking*. URL: `http://trainer.edu.mirantis.com/SDN50/sdn.html`.