Lab 1

Part 1: Answer Questions

1. When ONOS activate “org.onosproject.openflow,” what APPs does it activate?

```
lilywu@root > apps -a -s
*   9 org.onosproject.optical-model        2.7.0    Optical Network Model
*  19 org.onosproject.drivers              2.7.0    Default Drivers
*  52 org.onosproject.openflow-base        2.7.0    OpenFlow Base Provider
*  56 org.onosproject.hostprovider         2.7.0    Host Location Provider
*  57 org.onosproject.lldpprovider         2.7.0    LLDP Link Provider
*  58 org.onosproject.openflow             2.7.0    OpenFlow Provider Suite
*  82 org.onosproject.gui2                 2.7.0    ONOS GUI2
lilywu@root > app deactivate org.onosproject.openflow
Deactivated org.onosproject.openflow
lilywu@root > apps -a -s
*  19 org.onosproject.drivers              2.7.0    Default Drivers
*  82 org.onosproject.gui2                 2.7.0    ONOS GUI2
```

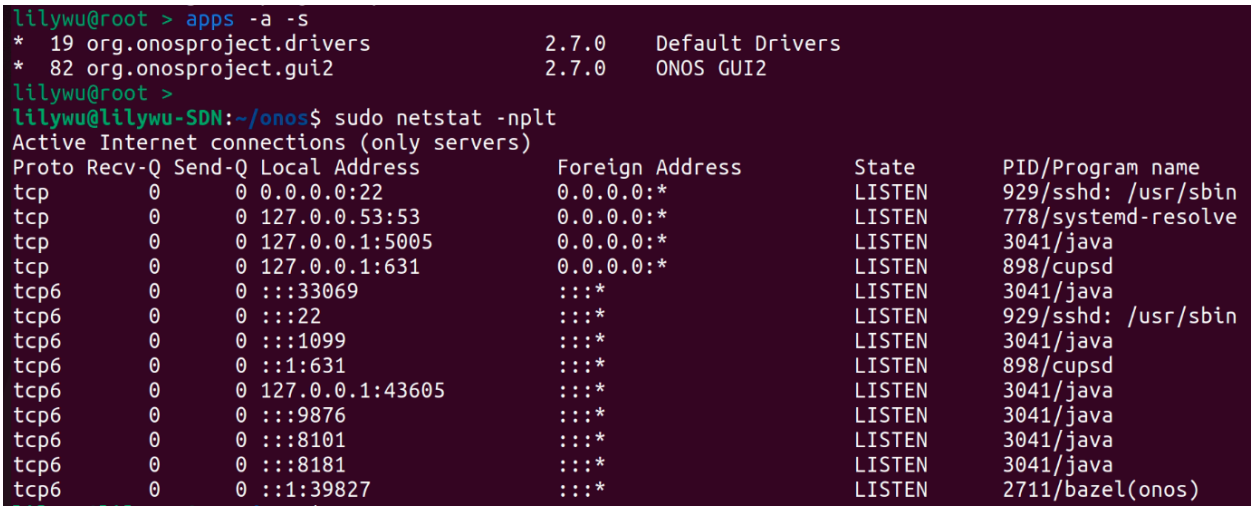
   As shown in the screenshot, these are the apps that also become deactivated when “org.onosproject.openflow” is no longer activated:
   a. org.onosproject.optical-model
   b. org.onosproject.openflow-base
   c. org.onosproject.hostprovider
   d. org.onosproject.lldpprovider

2. After we activate ONOS and run P.17 Mininet command, will H1 ping H2 successfully? Why or why not?

   According to the reference website, the ping between H1 and H2 will fail because there are 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. To ping successfully between the two hosts, we have to type the following command in ONOS: *app activate org.onosproject.fwd* to activate the forwarding app. After the app is activated successfully, H1 can ping H2.

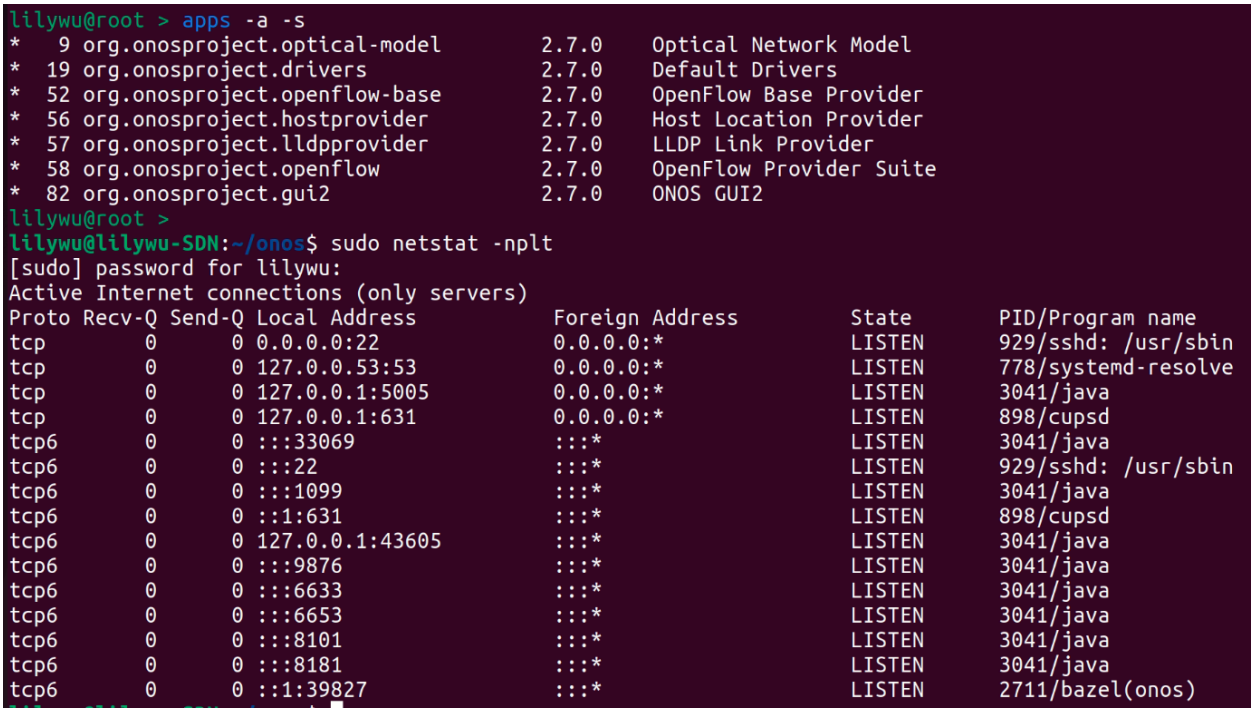
3. Which TCP port does the controller listen to the OpenFlow connection request from the switch? (Take screenshots and explain your answer.)

When org.onosproject.openflow is deactivated:



```
lilywu@root > apps -a -s
*  19 org.onosproject.drivers             2.7.0    Default Drivers
*  82 org.onosproject.gui2                2.7.0    ONOS GUI2
lilywu@root >
lilywu@lilywu-SDN:~/onos$ sudo netstat -nplt
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      929/sshd: /usr/sbin
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      778/systemd-resolve
tcp        0      0 127.0.0.1:5005          0.0.0.0:*               LISTEN      3041/java
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      898/cupsd
tcp6       0      0 :::33069                :::*                    LISTEN      3041/java
tcp6       0      0 :::22                   :::*                    LISTEN      929/sshd: /usr/sbin
tcp6       0      0 :::1099                 :::*                    LISTEN      3041/java
tcp6       0      0 ::1:631                 :::*                    LISTEN      898/cupsd
tcp6       0      0 127.0.0.1:43605         :::*                    LISTEN      3041/java
tcp6       0      0 :::9876                 :::*                    LISTEN      3041/java
tcp6       0      0 :::8101                 :::*                    LISTEN      3041/java
tcp6       0      0 :::8181                 :::*                    LISTEN      3041/java
tcp6       0      0 ::1:39827               :::*                    LISTEN      2711/bazel(onos)
```

When org.onosproject.openflow is activated:



```
lilywu@root > apps -a -s
*   9 org.onosproject.optical-model       2.7.0    Optical Network Model
*  19 org.onosproject.drivers             2.7.0    Default Drivers
*  52 org.onosproject.openflow-base       2.7.0    OpenFlow Base Provider
*  56 org.onosproject.hostprovider        2.7.0    Host Location Provider
*  57 org.onosproject.lldpprovider        2.7.0    LLDP Link Provider
*  58 org.onosproject.openflow            2.7.0    OpenFlow Provider Suite
*  82 org.onosproject.gui2                2.7.0    ONOS GUI2
lilywu@root >
lilywu@lilywu-SDN:~/onos$ sudo netstat -nplt
[sudo] password for lilywu:
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      929/sshd: /usr/sbin
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      778/systemd-resolve
tcp        0      0 127.0.0.1:5005          0.0.0.0:*               LISTEN      3041/java
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      898/cupsd
tcp6       0      0 :::33069                :::*                    LISTEN      3041/java
tcp6       0      0 :::22                   :::*                    LISTEN      929/sshd: /usr/sbin
tcp6       0      0 :::1099                 :::*                    LISTEN      3041/java
tcp6       0      0 ::1:631                 :::*                    LISTEN      898/cupsd
tcp6       0      0 127.0.0.1:43605         :::*                    LISTEN      3041/java
tcp6       0      0 :::9876                 :::*                    LISTEN      3041/java
tcp6       0      0 :::6633                 :::*                    LISTEN      3041/java
tcp6       0      0 :::6653                 :::*                    LISTEN      3041/java
tcp6       0      0 :::8101                 :::*                    LISTEN      3041/java
tcp6       0      0 :::8181                 :::*                    LISTEN      3041/java
tcp6       0      0 ::1:39827               :::*                    LISTEN      2711/bazel(onos)
```

When the org.onosproject.openflow app is activated, two additional TCP connections appear on ports 6633 and 6653. Port 6653 is used by the controller to listen for OpenFlow connection requests from switches, while port 6633 was used in older versions of OpenFlow.

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

   After some trials and errors, I found out that it is org.onosproject.openflow-base, or the OpenFlow Base Provider that enables the controller to liten on the TCP port.

Part 2: Create a Custom Topology

Code: lab1_part2_110550091.py

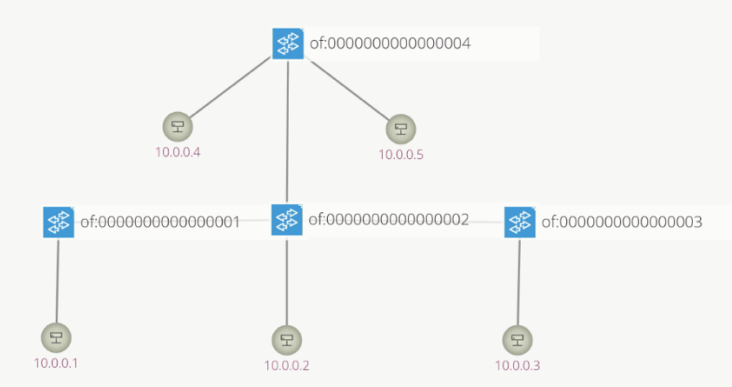
```python
from mininet.topo import Topo

class Lab1_Topo_110550091( 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')

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

topos = { 'topo_part2_110550091':  Lab1_Topo_110550091 }
```

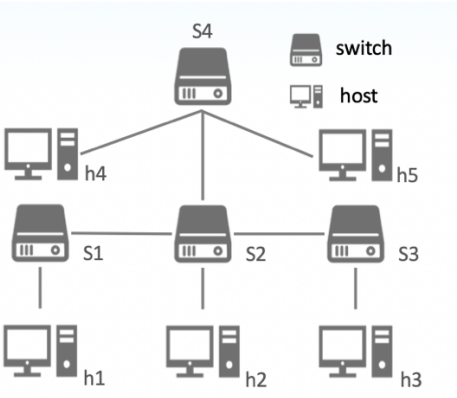
In the Lab1_Topo_110550091 class, I added 5 hosts using addHost(), 4 switches using addSwitch(), and 8 links using addLink() to create the desired custom topology.

To create this custom topology in mininet, I ran this command in the terminal:

```
$ sudo mn --custom=lab1_part2_110550091.py \
--topo=topo_part2_110550091 \
--controller=remote,ip=127.0.0.1:6653 \
--switch=ovs,protocols=OpenFlow14
```

Topology:





Part3: Statically Assign Hosts IP Address in Mininet

Code: lab1_part3_110550091.py

```python
from mininet.topo import Topo

class Lab1_Topo_110550091( Topo ):
    def __init__(self):
        Topo.__init__(self)
        # subnet mask = 255.255.255.224

        # Add hosts
        h1 = self.addHost('h1', ip='192.168.0.1/27')
        h2 = self.addHost('h2', ip='192.168.0.2/27')
        h3 = self.addHost('h3', ip='192.168.0.3/27')
        h4 = self.addHost('h4', ip='192.168.0.4/27')
        h5 = self.addHost('h5', ip='192.168.0.5/27')

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

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

topos = { 'topo_part3_110550091':  Lab1_Topo_110550091 }
```

To have a netmask of 255.255.255.224 and format the IP address of the hosts to 192.168.0.0/27, I modified the host section to specify our need by adding a new parameter.

Dump:

```
mininet> dump
<Host h1: h1-eth0:192.168.0.1 pid=87409>
<Host h2: h2-eth0:192.168.0.2 pid=87411>
<Host h3: h3-eth0:192.168.0.3 pid=87413>
<Host h4: h4-eth0:192.168.0.4 pid=87415>
<Host h5: h5-eth0:192.168.0.5 pid=87417>
<OVSSwitch{'protocols': 'OpenFlow14'} s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=87422>
<OVSSwitch{'protocols': 'OpenFlow14'} s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None,s2-eth4:None pid=87425>
<OVSSwitch{'protocols': 'OpenFlow14'} s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None pid=87428>
<OVSSwitch{'protocols': 'OpenFlow14'} s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None,s4-eth3:None pid=87431>
<RemoteController{'ip': '127.0.0.1:6653'} c0: 127.0.0.1:6653 pid=87403>
```

H1 ~ H5 ificonfig:

```
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::5821:4eff:fe3e:1d8a  prefixlen 64  scopeid 0x20<link>
        ether 5a:21:4e:3e:1d:8a  txqueuelen 1000  (Ethernet)
        RX packets 143  bytes 18388 (18.3 KB)
        RX errors 0  dropped 94  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::2462:1eff:fe85:c351  prefixlen 64  scopeid 0x20<link>
        ether 26:62:1e:85:c3:51  txqueuelen 1000  (Ethernet)
        RX packets 177  bytes 23114 (23.1 KB)
        RX errors 0  dropped 128  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> 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::70b8:e0ff:fe22:c3db  prefixlen 64  scopeid 0x20<link>
        ether 72:b8:e0:22:c3:db  txqueuelen 1000  (Ethernet)
        RX packets 199  bytes 26034 (26.0 KB)
        RX errors 0  dropped 148  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::88f1:adff:feea:dc86  prefixlen 64  scopeid 0x20<link>
        ether 8a:f1:ad:ea:dc:86  txqueuelen 1000  (Ethernet)
        RX packets 226  bytes 29851 (29.8 KB)
        RX errors 0  dropped 174  overruns 0  frame 0
        TX packets 28  bytes 2056 (2.0 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> 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::d877:14ff:fe53:b25a  prefixlen 64  scopeid 0x20<link>
        ether da:77:14:53:b2:5a  txqueuelen 1000  (Ethernet)
        RX packets 243  bytes 32145 (32.1 KB)
        RX errors 0  dropped 190  overruns 0  frame 0
        TX packets 28  bytes 2056 (2.0 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
```

What I've learned or solved.

1. **Building an SDN Network**: I learned how to build a virtual network with ONOS and Mininet, activate the control plane, and connect the controller to the switches.
2. **Custom Topologies**: I learned how to write a Python script to create a custom network topology using Mininet, and manually assigning IP addresses to hosts.

3. **Controller-Switch Communication**: I understood how the SDN controller communicates with switches through OpenFlow, and investigated which ports (e.g., 6653) the controller uses for communication.
4. **Basic ONOS Operation**: I used the ONOS CLI and GUI to manage applications, monitor network connections, and activate basic SDN apps.

Overall, this lab helped me understand how to set up and simulate a software-defined network using ONOS as the controller and Mininet for network emulation