Project 1

Part 1: Answer Questions

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

```
zhuit@root > apps -a -s
*   8 org.onosproject.drivers              2.7.0    Default Drivers
*  31 org.onosproject.optical-model        2.7.0    Optical Network Model
*  32 org.onosproject.openflow-base        2.7.0    OpenFlow Base Provider
*  33 org.onosproject.lldpprovider         2.7.0    LLDP Link Provider
*  34 org.onosproject.hostprovider         2.7.0    Host Location Provider
*  80 org.onosproject.openflow             2.7.0    OpenFlow Provider Suite
* 111 org.onosproject.gui2                 2.7.0    ONOS GUI2
zhuit@root > app deactivate org.onosproject.openflow
Deactivated org.onosproject.openflow
zhuit@root > apps -a -s
*   8 org.onosproject.drivers              2.7.0    Default Drivers
* 111 org.onosproject.gui2                 2.7.0    ONOS GUI2
```

as shown in the picture, we can konw that when deactivating “org.onosproject.openflow”, it also deactivates other apps, which are:

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

2. After activat ing ONOS and run ning the commands on P.17 and P. 20
Will H1 ping H2 successfully? Why or why not?

Well it can’t, because there are no flows installed on the data-plane. There is an app , **“org.onosproject.fwd”** which in respond to forwarding. But the default is deactivated so we can’t ping other hosts.

3. Which TCP port the controller listens for the OpenFlow connection request from the switch? screenshot

```
zhuit@root > apps -a -s
*   8 org.onosproject.drivers              2.7.0    Default Drivers
* 111 org.onosproject.gui2                2.7.0    ONOS GUI2
zhuit@root >
zhuit@SDN-NFV:~/onos$ sudo netstat -nlpt
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      457/systemd-resolve
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      722/sshd: /usr/sbin
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      699/cupsd
tcp        0      0 127.0.0.1:5005          0.0.0.0:*               LISTEN      4598/java
tcp6       0      0 :::1099                 :::*                    LISTEN      4598/java
tcp6       0      0 ::1:43983               :::*                    LISTEN      4310/bazel(onos)
tcp6       0      0 :::9876                 :::*                    LISTEN      4598/java
tcp6       0      0 :::22                   :::*                    LISTEN      722/sshd: /usr/sbin
tcp6       0      0 :::45063                :::*                    LISTEN      4598/java
tcp6       0      0 ::1:631                 :::*                    LISTEN      699/cupsd
tcp6       0      0 :::8101                 :::*                    LISTEN      4598/java
tcp6       0      0 :::8181                 :::*                    LISTEN      4598/java
tcp6       0      0 127.0.0.1:39547         :::*                    LISTEN      4598/java
```

```
zhuit@root > apps -a -s
*   8 org.onosproject.drivers              2.7.0    Default Drivers
*  31 org.onosproject.optical-model        2.7.0    Optical Network Model
*  32 org.onosproject.openflow-base        2.7.0    OpenFlow Base Provider
*  33 org.onosproject.lldpprovider         2.7.0    LLDP Link Provider
*  34 org.onosproject.hostprovider         2.7.0    Host Location Provider
*  80 org.onosproject.openflow             2.7.0    OpenFlow Provider Suite
* 111 org.onosproject.gui2                 2.7.0    ONOS GUI2
zhuit@root >
zhuit@SDN-NFV:~/onos$ sudo netstat -nlpt
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      457/systemd-resolve
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      722/sshd: /usr/sbin
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      699/cupsd
tcp        0      0 127.0.0.1:5005          0.0.0.0:*               LISTEN      4598/java
tcp6       0      0 :::1099                 :::*                    LISTEN      4598/java
tcp6       0      0 ::1:43983               :::*                    LISTEN      4310/bazel(onos)
tcp6       0      0 :::9876                 :::*                    LISTEN      4598/java
tcp6       0      0 :::22                   :::*                    LISTEN      722/sshd: /usr/sbin
tcp6       0      0 :::45063                :::*                    LISTEN      4598/java
tcp6       0      0 ::1:631                 :::*                    LISTEN      699/cupsd
tcp6       0      0 :::8101                 :::*                    LISTEN      4598/java
tcp6       0      0 :::8181                 :::*                    LISTEN      4598/java
tcp6       0      0 :::6653                 :::*                    LISTEN      4598/java
tcp6       0      0 :::6633                 :::*                    LISTEN      4598/java
tcp6       0      0 127.0.0.1:39547         :::*                    LISTEN      4598/java
zhuit@SDN-NFV:~/onos$
```

Before activating openFlow, the TCP connection is shown by command `netstat -nlpt` on the first picutre (upper part). Compared to the first picture, we can see that there are two more TCP connections shown, which is port **6653** and **6633**.

The comparison show that **port 6653/tcp** is the port openFlow controller listening on for switches. The **port 6633/tcp** is used in ealier version of openFlow. Reference: https://www.speedguide.net/port.php?port=6633

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

After deactivating each app one by one. We can know that ***OpenFlow Base Provider*** is the APP that enables the controller to listen on the TCP port.

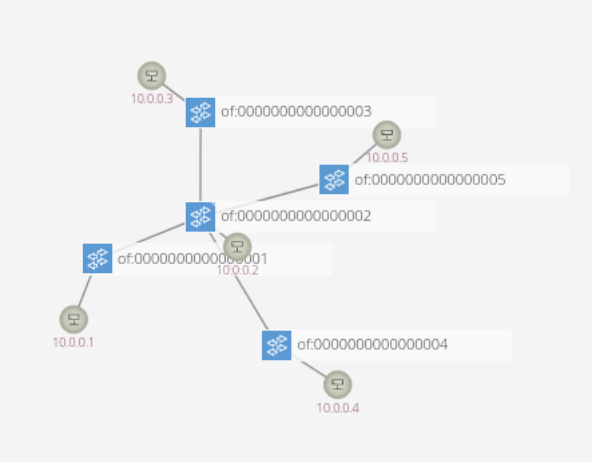
Part 2: Create a custom Topology

Code:

```
from mininet.topo import Topo

class Project1_Topo_111550093( 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
        self.addLink(h4, s4)
        self.addLink(s2, s4)
        self.addLink(s2, s1)
        self.addLink(s2, s3)
        self.addLink(s2, s5)
        self.addLink(s2, h2)
        self.addLink(s1, h1)
        self.addLink(s3, h3)
        self.addLink(s5, h5)

topos = { 'topo_part2_111550093':  Project1_Topo_111550093 }
```

By the topology shown in the file, we need to

1. Create and add five hosts by `addHost`
2. Create and add five switch by `addSwitch`
3. Link switches and hosts by `addLink`



then we need to create the topology on mininet, run this command on terminal

```
$ sudo mn custom= project1_part2_<studentID>.py
 --topo= topo_part2_<studentID>
 --controller=remote,ip= 127.0.0.1:6653
 --switch= ovs,protocols =OpenFlow14
```

Part 3: Create a custom Topology

Code:

```
from mininet.topo import Topo

class Project1_Topo_111550093( 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')
        s5 = self.addSwitch('s5')

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

topos = { 'topo_part3_111550093':  Project1_Topo_111550093 }
```

1. We can set subnet mask by setting the ip address to 27 bits
2. Steps are almost the same with part2, but we need to add a new parameter when we creating hosts.

```
mininet> dump
<Host h1: h1-eth0:192.168.0.1 pid=92670>
<Host h2: h2-eth0:192.168.0.2 pid=92672>
<Host h3: h3-eth0:192.168.0.3 pid=92674>
<Host h4: h4-eth0:192.168.0.4 pid=92676>
<Host h5: h5-eth0:192.168.0.5 pid=92678>
<OVSSwitch{'protocols': 'OpenFlow14'} s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=92683>
<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=92686>
<OVSSwitch{'protocols': 'OpenFlow14'} s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None pid=92689>
<OVSSwitch{'protocols': 'OpenFlow14'} s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None pid=92692>
<OVSSwitch{'protocols': 'OpenFlow14'} s5: lo:127.0.0.1,s5-eth1:None,s5-eth2:None pid=92695>
<RemoteController{'ip': '127.0.0.1:6653'} c0: 127.0.0.1:6653 pid=92664>
```

```
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::90ef:bbff:fe12:7397  prefixlen 64  scopeid 0x20<link>
        ether 92:ef:bb:12:73:97  txqueuelen 1000  (Ethernet)
        RX packets 59  bytes 7922 (7.9 KB)
        RX errors 0  dropped 36  overruns 0  frame 0
        TX packets 9  bytes 726 (726.0 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::82e:66ff:fea9:b1e0  prefixlen 64  scopeid 0x20<link>
        ether 0a:2e:66:a9:b1:e0  txqueuelen 1000  (Ethernet)
        RX packets 81  bytes 10897 (10.8 KB)
        RX errors 0  dropped 56  overruns 0  frame 0
        TX packets 10  bytes 796 (796.0 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
Help    inet 192.168.0.3  netmask 255.255.255.224  broadcast 192.168.0.31
        inet6 fe80::f48b:efff:fe68:fcc4  prefixlen 64  scopeid 0x20<link>
        ether f6:8b:ef:68:fc:c4  txqueuelen 1000  (Ethernet)
        RX packets 101  bytes 13677 (13.6 KB)
        RX errors 0  dropped 76  overruns 0  frame 0
        TX packets 10  bytes 796 (796.0 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::28a2:43ff:fe14:4df2  prefixlen 64  scopeid 0x20<link>
        ether 2a:a2:43:14:4d:f2  txqueuelen 1000  (Ethernet)
        RX packets 110  bytes 14859 (14.8 KB)
Terminal X errors 0  dropped 84  overruns 0  frame 0
        TX packets 10  bytes 796 (796.0 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::c022:ccff:fea5:7a5b  prefixlen 64  scopeid 0x20<link>
        ether c2:22:cc:a5:7a:5b  txqueuelen 1000  (Ethernet)
        RX packets 120  bytes 16304 (16.3 KB)
        RX errors 0  dropped 94  overruns 0  frame 0
        TX packets 11  bytes 866 (866.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

What you’ve learned or solved.

After doing this project, I’m much more familiar with the ubuntu system and some basic function of mininet and onos, such as checking apps activated and the TCP connection. Also, now I can create a simple topology and give each of them an IP address.