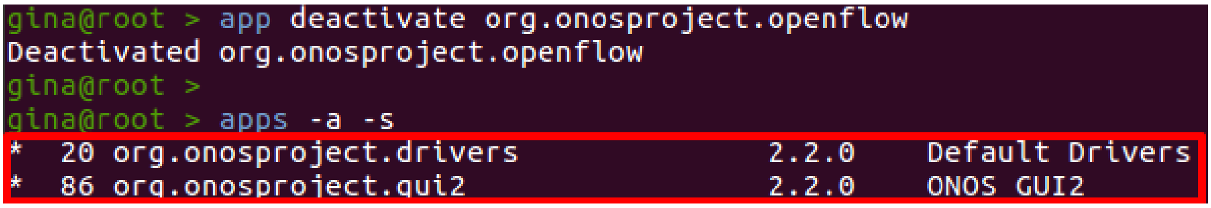
Part 1

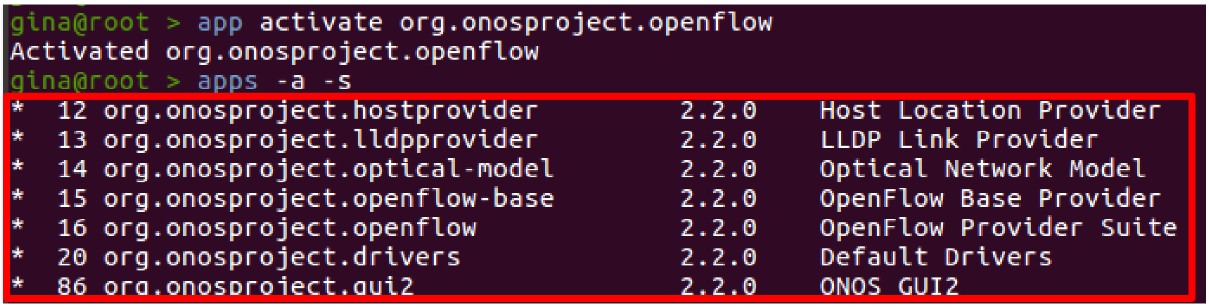
1. When ONOS activates “org.onosproject.openflow,” what are the APPs which it also activates?

Answer:
After deactivating the “org.onosproject.openflow” app, I list the active app in the following picture.



```
gina@root > app deactivate org.onosproject.openflow
Deactivated org.onosproject.openflow
gina@root >
gina@root > apps -a -s
*  20 org.onosproject.drivers                 2.2.0     Default Drivers
*  86 org.onosproject.gui2                    2.2.0     ONOS GUI2
```

It shows apps id about 20 and 86.
Then, I activate the “org.onosproject.openflow” app to compare which apps would be affected in the following picture.



```
gina@root > app activate org.onosproject.openflow
Activated org.onosproject.openflow
gina@root > apps -a -s
*  12 org.onosproject.hostprovider          2.2.0     Host Location Provider
*  13 org.onosproject.lldpprovider          2.2.0     LLDP Link Provider
*  14 org.onosproject.optical-model         2.2.0     Optical Network Model
*  15 org.onosproject.openflow-base         2.2.0     OpenFlow Base Provider
*  16 org.onosproject.openflow              2.2.0     OpenFlow Provider Suite
*  20 org.onosproject.drivers               2.2.0     Default Drivers
*  86 org.onosproject.gui2                  2.2.0     ONOS GUI2
```

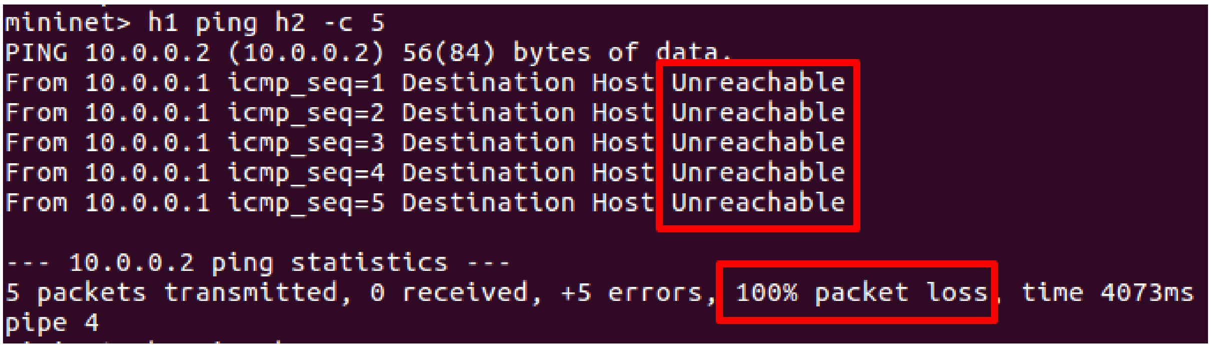
As we can see, the affected apps without id 20 and 86 are:

```
*  12 org.onosproject.hostprovider       2.2.0    Host Location Provider
*  13 org.onosproject.lldpprovider       2.2.0    LLDP Link Provider
*  14 org.onosproject.optical-model      2.2.0    Optical Network Model
*  15 org.onosproject.openflow-base      2.2.0    OpenFlow Base Provider
*  16 org.onosproject.openflow           2.2.0    OpenFlow Provider Suite
```

---

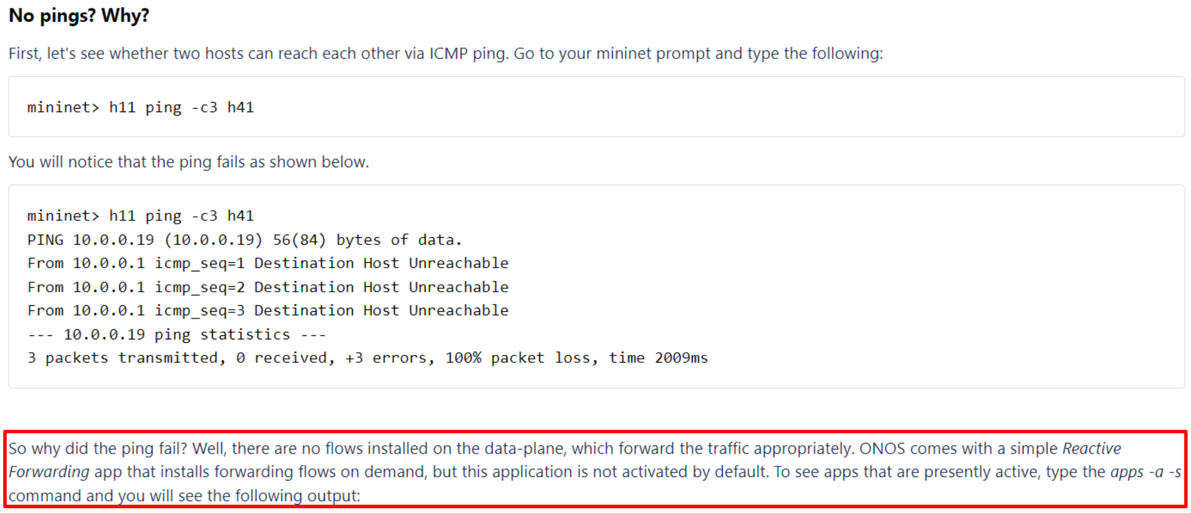
2. As topology in p.22, can H1 ping H2 successfully? Why or why not?

Answer:
No, H1 can’t ping H2.



```
mininet> h1 ping h2 -c 5
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable
From 10.0.0.1 icmp_seq=5 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 0 received, +5 errors, 100% packet loss, time 4073ms
pipe 4
```

Because of the reference ONOS document mentioned below, there are no flows installed on the data-plane, which forward the traffic appropriately.



No pings? Why?

First, let's see whether two hosts can reach each other via ICMP ping. Go to your mininet prompt and type the following:

```
mininet> h11 ping -c3 h41
```

You will notice that the ping fails as shown below.

```
mininet> h11 ping -c3 h41
PING 10.0.0.19 (10.0.0.19) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
--- 10.0.0.19 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2009ms
```

So why did the ping fail? Well, 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 see apps that are presently active, type the *apps -a -s* command and you will see the following output:

So we can understand that Reactive Forwarding app need to be activated to forward the traffic appropriately.

---

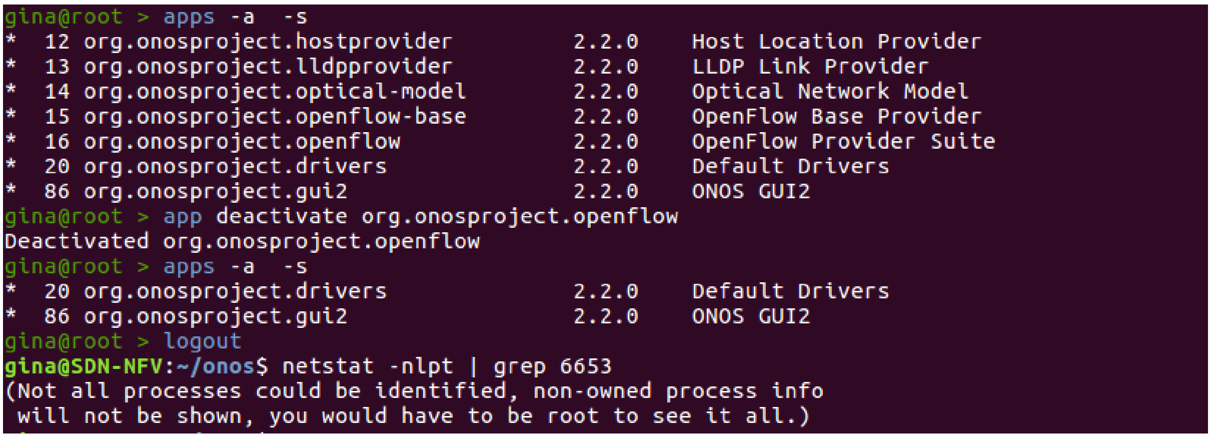
3. Which TCP port the controller listens for the OpenFlow connection request from the switch?

Answer:
6653 port.

---

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

Answer:
After deactivating "org.onosproject.openflow" app, and can't see any listening port with 6653.



```
gina@root > apps -a  -s
*  12 org.onosproject.hostprovider         2.2.0    Host Location Provider
*  13 org.onosproject.lldpprovider         2.2.0    LLDP Link Provider
*  14 org.onosproject.optical-model        2.2.0    Optical Network Model
*  15 org.onosproject.openflow-base        2.2.0    OpenFlow Base Provider
*  16 org.onosproject.openflow             2.2.0    OpenFlow Provider Suite
*  20 org.onosproject.drivers              2.2.0    Default Drivers
*  86 org.onosproject.gui2                 2.2.0    ONOS GUI2
gina@root > app deactivate org.onosproject.openflow
Deactivated org.onosproject.openflow
gina@root > apps -a  -s
*  20 org.onosproject.drivers              2.2.0    Default Drivers
*  86 org.onosproject.gui2                 2.2.0    ONOS GUI2
gina@root > logout
gina@SDN-NFV:~/onos$ netstat -nlpt | grep 6653
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
```

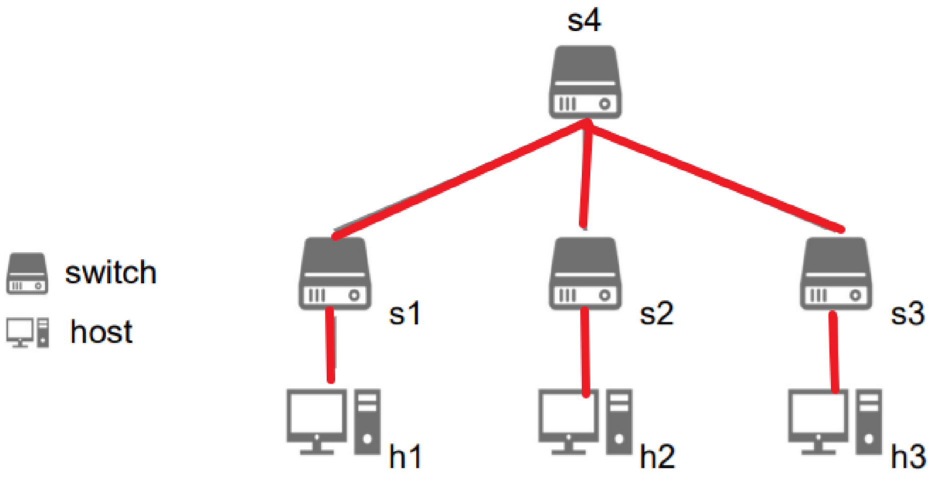
And do a test about activating each of the apps, then check if the listening ports have 6653 or not. The following table shows the results.

app id	app name	app listen 6653 port after activate (Y/N)
12	org.onosproject.hostprovider	N
13	org.onosproject.lldpprovider	N
14	org.onosproject.optical-model	N
15	org.onosproject.openflow-base	Y **(this app is one of “org.onosproject.openflow” dependencies)**
16	org.onosproject.openflow	Y

So we can know that the computer would create a listening port with 6653 after one of the **“org.onosproject.openflow”** and **“org.onosproject.openflow-base”** apps are activated.

Part 2

Write a Python script to build the following topology:



Answer:

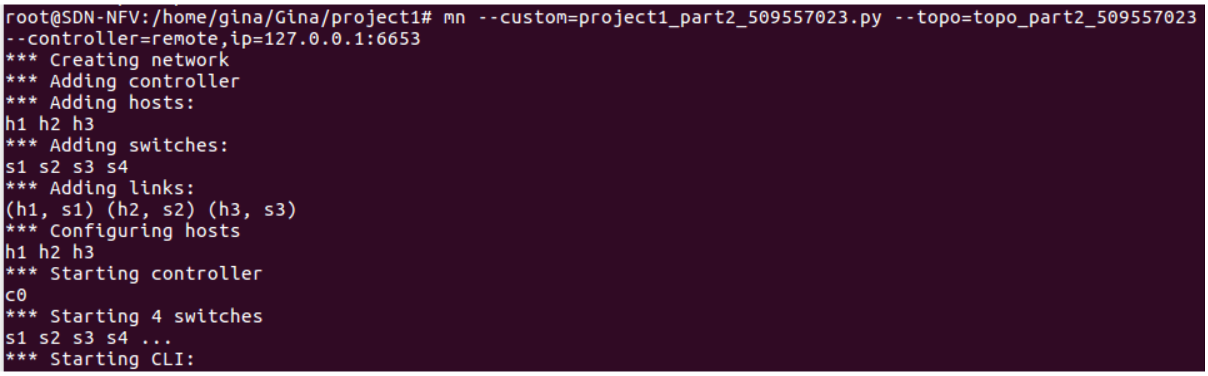
We can know the topology from above picture that have to create:

- Hosts: h1, h2, h3
- Switchs: s1, s2, s3
- Links: (the red color lines in the above picture)
    - h1 to s1
    - s1 to s4
    - h2 to s2
    - s2 to s4
    - h3 to s3
    - s3 to s4

So the python code is:

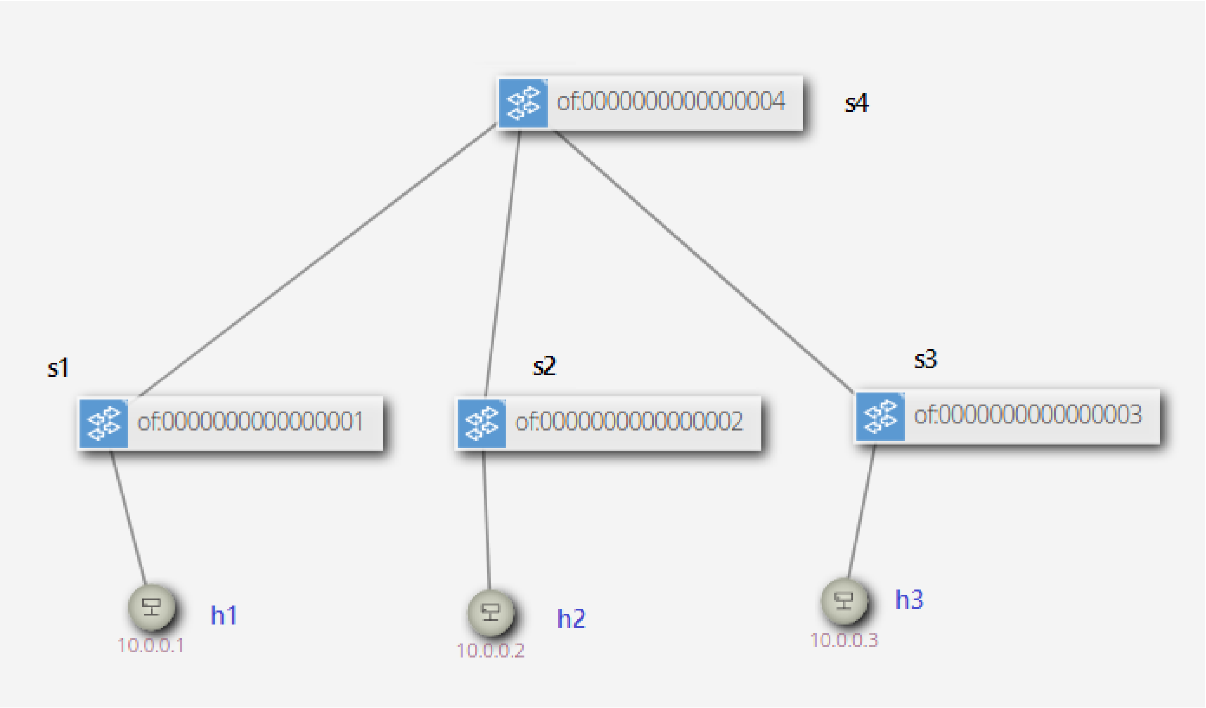
```
from mininet.topo import Topo


class Project1_Topo_509557023(Topo):
    def __init__(self):
        Topo.__init__(self)

        # Add hosts
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        h3 = self.addHost('h3')

        # 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, s4)
        self.addLink(s2, s4)
        self.addLink(s3, s4)


topos = {'topo_part2_509557023': Project1_Topo_509557023}
```

Then use the specified py file to create the topology.

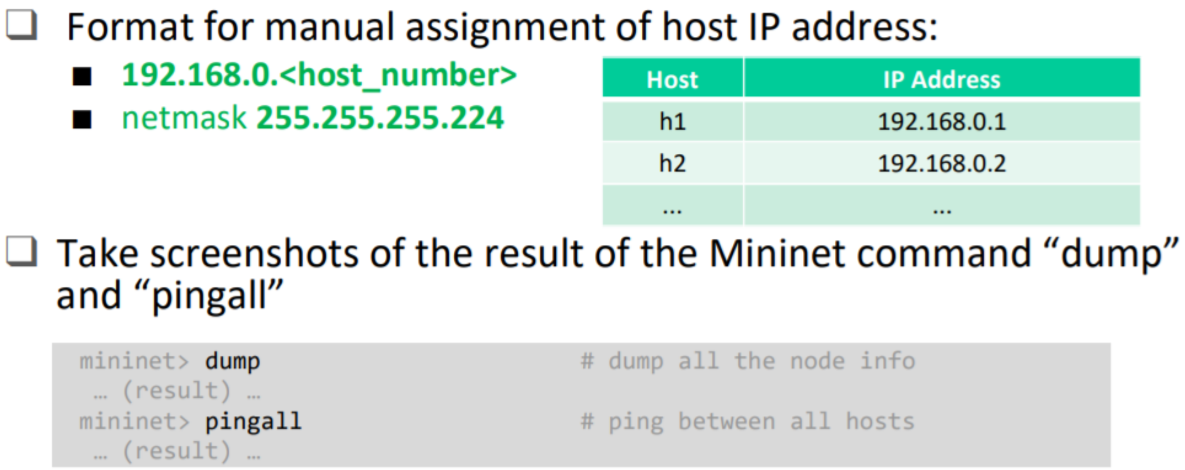


```
root@SDN-NFV:/home/gina/Gina/project1# mn --custom=project1_part2_509557023.py --topo=topo_part2_509557023
--controller=remote,ip=127.0.0.1:6653
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s2) (h3, s3)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
```

And can see the topology view on the ONOS GUI.



Part 3



❑ Format for manual assignment of host IP address:

- 192.168.0.<host_number>
- netmask 255.255.255.224

Host	IP Address
h1	192.168.0.1
h2	192.168.0.2
…	…

❑ Take screenshots of the result of the Mininet command "dump" and "pingall"

```
mininet> dump                           # dump all the node info
 … (result) …
mininet> pingall                        # ping between all hosts
 … (result) …
```

Answer:
Need to assign host IP address to these 2 requirement.

- 192.168.0.<host_number>
- netmask 255.255.255.224

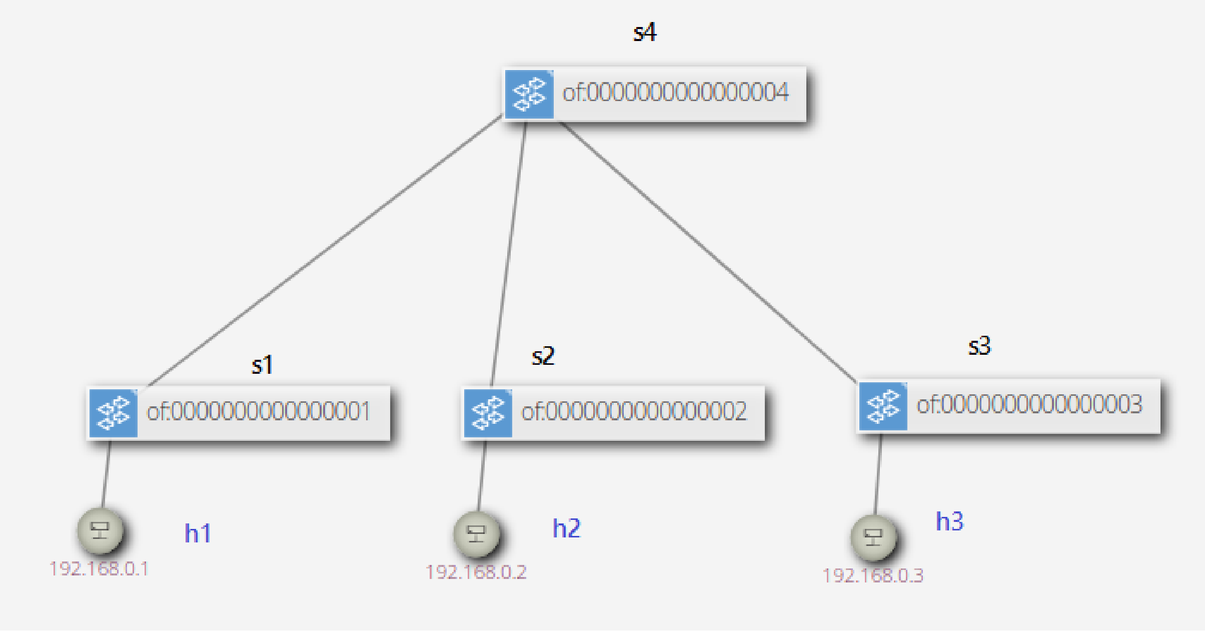
netmask 255.255.255.224
can convert to binary 32 bits:
11111111 11111111 11111111 11100000
So, we can get the first 27 bits are all 1, then set to host ip address with CIDR format.

- h1: 192.168.0.1/27
- h2: 192.168.0.2/27
- h3: 192.168.0.3/27

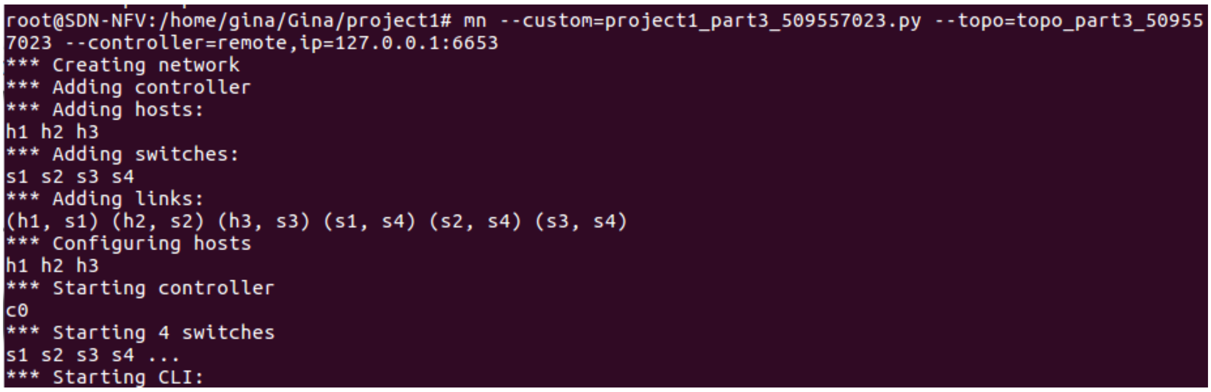
And the python code in the following.

```python
from mininet.topo import Topo


class Project1_Topo_509557023(Topo):
    def __init__(self):
        Topo.__init__(self)

        # 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')

        # 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, s4)
        self.addLink(s2, s4)
        self.addLink(s3, s4)


topos = {'topo_part3_509557023': Project1_Topo_509557023}
```

And can see the topology view on the ONOS GUI.

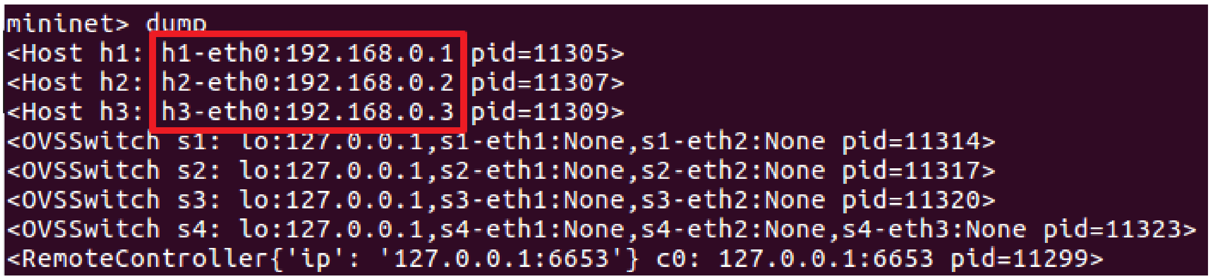


Then check the corresponding configuration.
First, create the topology with specific ip python code.

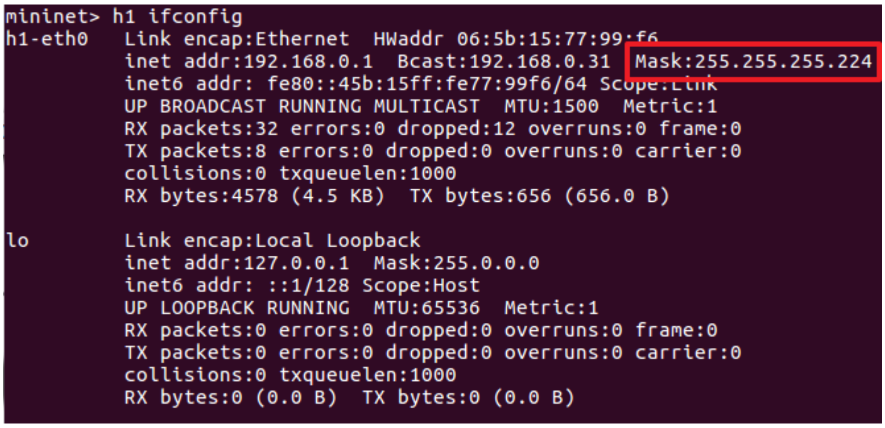


```
root@SDN-NFV:/home/gina/Gina/project1# mn --custom=project1_part3_509557023.py --topo=topo_part3_50955
7023 --controller=remote,ip=127.0.0.1:6653
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s1, s4) (s2, s4) (s3, s4)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
```

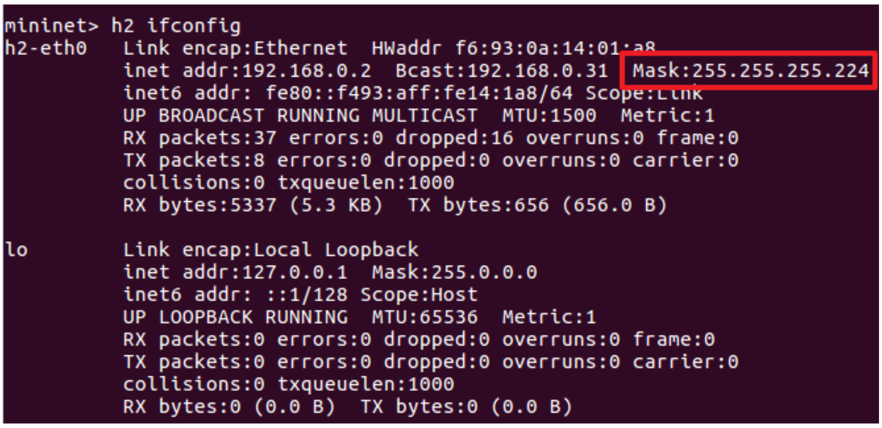
Then, check the “dump” command results and host mask information.



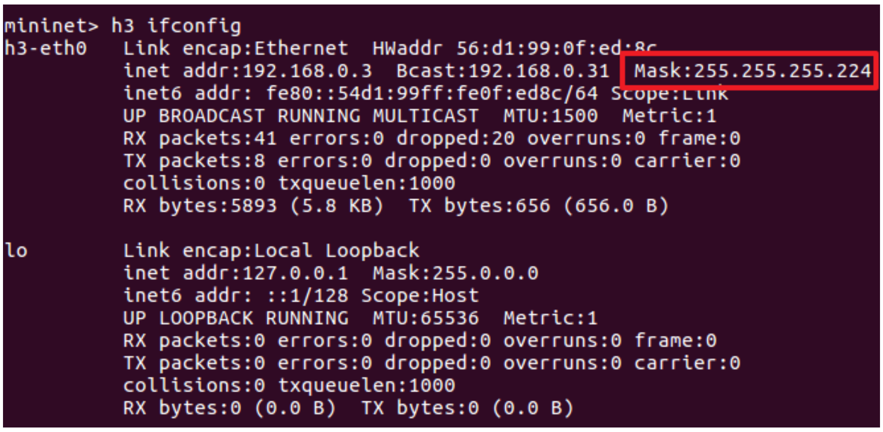
```
mininet> dump
<Host h1: h1-eth0:192.168.0.1 pid=11305>
<Host h2: h2-eth0:192.168.0.2 pid=11307>
<Host h3: h3-eth0:192.168.0.3 pid=11309>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=11314>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None pid=11317>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None pid=11320>
<OVSSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None,s4-eth3:None pid=11323>
<RemoteController{'ip': '127.0.0.1:6653'} c0: 127.0.0.1:6653 pid=11299>
```



```
mininet> h1 ifconfig
h1-eth0   Link encap:Ethernet  HWaddr 06:5b:15:77:99:f6
          inet addr:192.168.0.1  Bcast:192.168.0.31  Mask:255.255.255.224
          inet6 addr: fe80::45b:15ff:fe77:99f6/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:32 errors:0 dropped:12 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4578 (4.5 KB)  TX bytes:656 (656.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

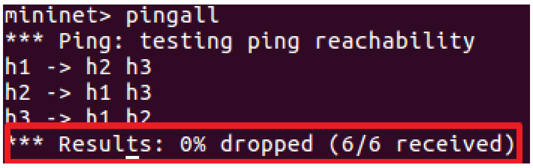


```
mininet> h2 ifconfig
h2-eth0   Link encap:Ethernet  HWaddr f6:93:0a:14:01:a8
          inet addr:192.168.0.2  Bcast:192.168.0.31  Mask:255.255.255.224
          inet6 addr: fe80::f493:aff:fe14:1a8/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:37 errors:0 dropped:16 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5337 (5.3 KB)  TX bytes:656 (656.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```



```
mininet> h3 ifconfig
h3-eth0   Link encap:Ethernet  HWaddr 56:d1:99:0f:ed:8c
          inet addr:192.168.0.3  Bcast:192.168.0.31  Mask:255.255.255.224
          inet6 addr: fe80::54d1:99ff:fe0f:ed8c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:41 errors:0 dropped:20 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5893 (5.8 KB)  TX bytes:656 (656.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

And check the “pingall” command result.



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
```