111550139

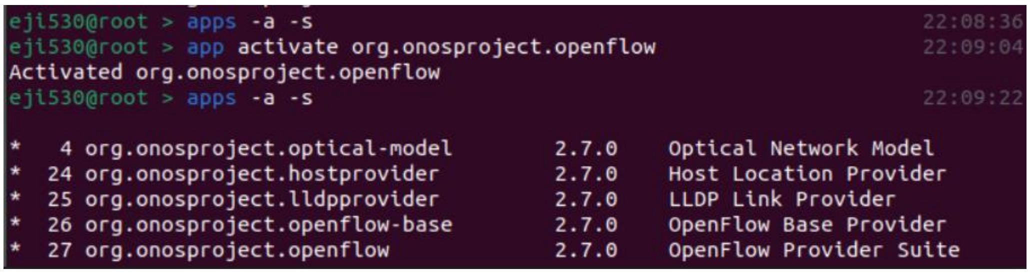
郭芷安

SDN LAB1 report

Part1 answer questions

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

   Ans:



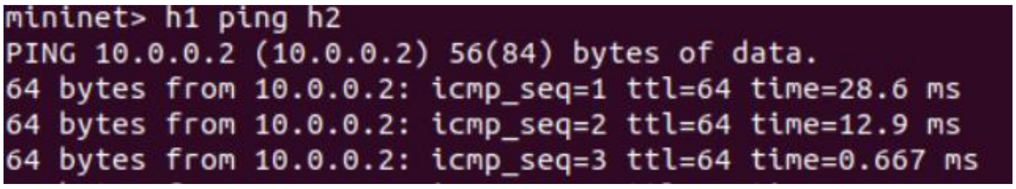
```
eji530@root > apps -a -s                                              22:08:36
eji530@root > app activate org.onosproject.openflow                   22:09:04
Activated org.onosproject.openflow
eji530@root > apps -a -s                                              22:09:22

*   4 org.onosproject.optical-model        2.7.0    Optical Network Model
*  24 org.onosproject.hostprovider         2.7.0    Host Location Provider
*  25 org.onosproject.lldpprovider         2.7.0    LLDP Link Provider
*  26 org.onosproject.openflow-base        2.7.0    OpenFlow Base Provider
*  27 org.onosproject.openflow             2.7.0    OpenFlow Provider Suite
```

   As shown in the above screenshot, I deactivate all the apps at first, then activate org.onosproject.openflow and command “apps -a -s” to see what apps will be activated. the result turns out the following apps will be activated:

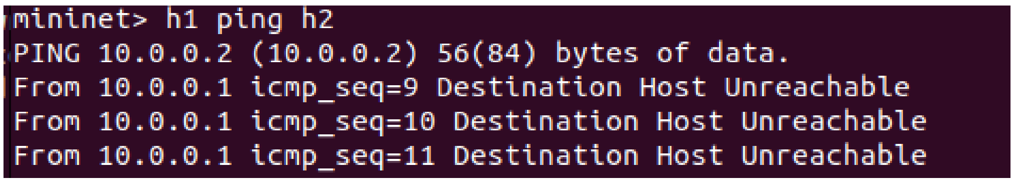
   a. Optical Network Model
   b. Host Location Provider
   c. LLDP Location Provider
   d. OpenFlow Base Provider
   e. OpenFlow Provider Suite

2. After we activate ONOS and run P.18 Mininet command, will H1 ping H2 successfully? Why or why not?

   Ans:



```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=28.6 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=12.9 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.667 ms
```

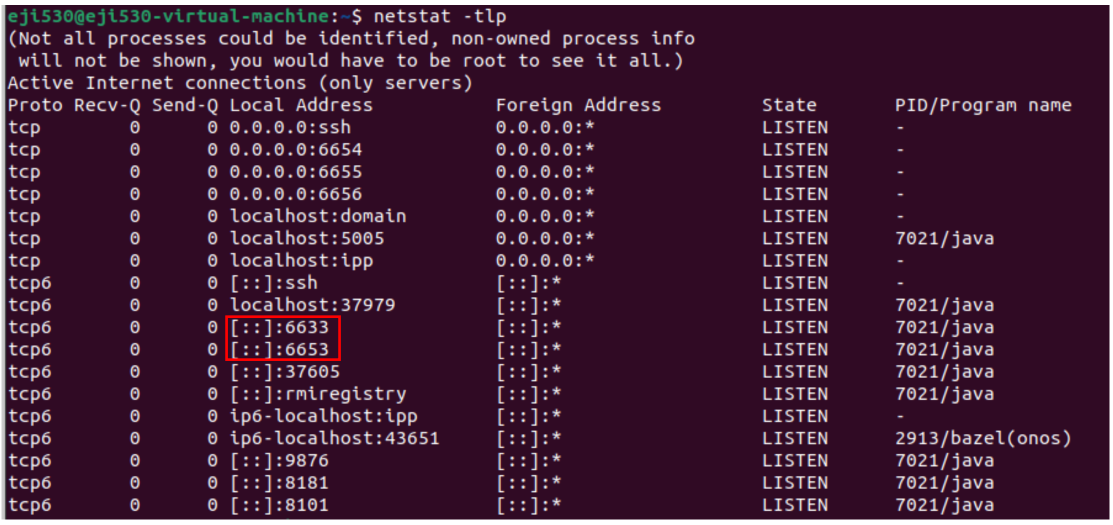


```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=9 Destination Host Unreachable
From 10.0.0.1 icmp_seq=10 Destination Host Unreachable
From 10.0.0.1 icmp_seq=11 Destination Host Unreachable
```

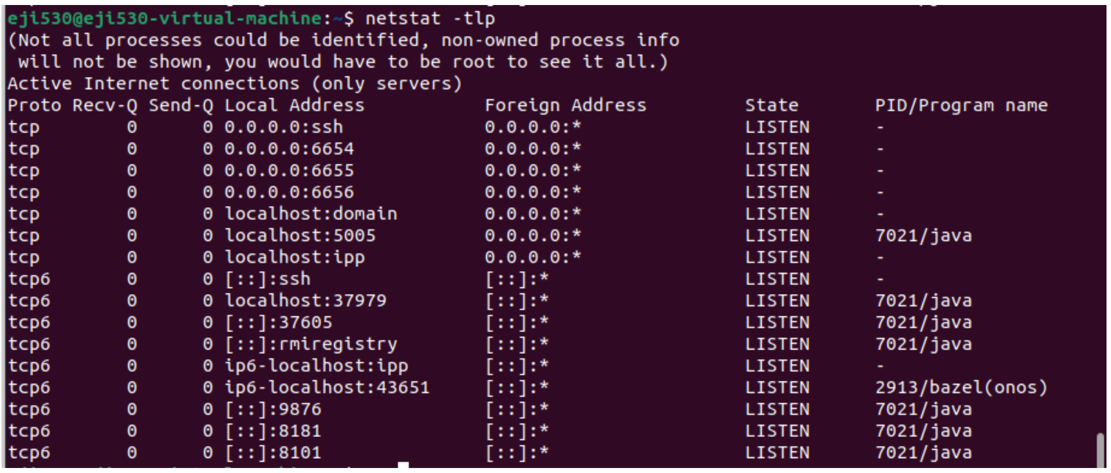
This depends on whether the app “Reactive Forwarding” is activated. Without this app, flows will not be installed on the data-plane, so the traffic will not be forward appropriately. This, h2 will be unreachable to h1. I have activated org.onosproject.fwd after starting ONOS, so when I command h1 ping h2 in mininet, h1 can ping h2 successfully. The first picture above shows the result of h1 ping h2 when reactive forwarding is already activated, and the second picture above shows the result when reactive forwarding is not activated.

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

Ans:



```
eji530@eji530-virtual-machine:~$ netstat -tlp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
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:ssh             0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:6654            0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:6655            0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:6656            0.0.0.0:*               LISTEN      -
tcp        0      0 localhost:domain        0.0.0.0:*               LISTEN      -
tcp        0      0 localhost:5005          0.0.0.0:*               LISTEN      7021/java
tcp        0      0 localhost:ipp           0.0.0.0:*               LISTEN      -
tcp6       0      0 [::]:ssh                [::]:*                  LISTEN      -
tcp6       0      0 localhost:37979         [::]:*                  LISTEN      7021/java
tcp6       0      0 [::]:6633               [::]:*                  LISTEN      7021/java
tcp6       0      0 [::]:6653               [::]:*                  LISTEN      7021/java
tcp6       0      0 [::]:37605              [::]:*                  LISTEN      7021/java
tcp6       0      0 [::]:rmiregistry        [::]:*                  LISTEN      7021/java
tcp6       0      0 ip6-localhost:ipp       [::]:*                  LISTEN      -
tcp6       0      0 ip6-localhost:43651     [::]:*                  LISTEN      2913/bazel(onos)
tcp6       0      0 [::]:9876               [::]:*                  LISTEN      7021/java
tcp6       0      0 [::]:8181               [::]:*                  LISTEN      7021/java
tcp6       0      0 [::]:8101               [::]:*                  LISTEN      7021/java
```



```
eji530@eji530-virtual-machine:~$ netstat -tlp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
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:ssh             0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:6654            0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:6655            0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:6656            0.0.0.0:*               LISTEN      -
tcp        0      0 localhost:domain        0.0.0.0:*               LISTEN      -
tcp        0      0 localhost:5005          0.0.0.0:*               LISTEN      7021/java
tcp        0      0 localhost:ipp           0.0.0.0:*               LISTEN      -
tcp6       0      0 [::]:ssh                [::]:*                  LISTEN      -
tcp6       0      0 localhost:37979         [::]:*                  LISTEN      7021/java
tcp6       0      0 [::]:37605              [::]:*                  LISTEN      7021/java
tcp6       0      0 [::]:rmiregistry        [::]:*                  LISTEN      7021/java
tcp6       0      0 ip6-localhost:ipp       [::]:*                  LISTEN      -
tcp6       0      0 ip6-localhost:43651     [::]:*                  LISTEN      2913/bazel(onos)
tcp6       0      0 [::]:9876               [::]:*                  LISTEN      7021/java
tcp6       0      0 [::]:8181               [::]:*                  LISTEN      7021/java
tcp6       0      0 [::]:8101               [::]:*                  LISTEN      7021/java
```

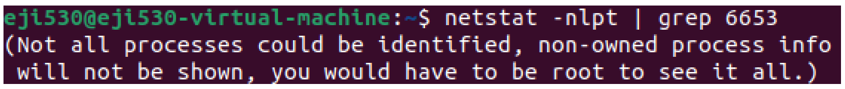
The first screenshot above shows the result when command “netstat -tlp”(t for TCP, l for listen, and p for process), and OpenFlow is activated in this scenario. The second screenshot shows the result when OpenFlow is deactivated. We can observe that two ports (6633 and 6653) are not shown in the second screenshot, so we can know that the controller listens to the OpenFlow connection request from the switch using port 6653. (6633 is for the earlier version of the OpenFlow protocol).

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

   Ans:

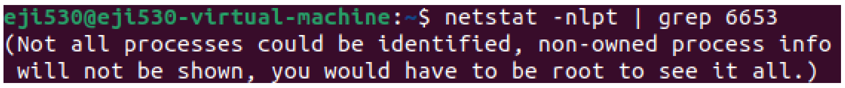
   The app OpenFlow Base Provider enables the controller to listen on the TCP port. To observe which APP enables the controller to listen on the TCP port, I deactivate all the OpenFlow related apps first, then activate them one by one, and use "netstat -nlpt | grep 6653" to see whether the port shows when each app is activated.

   The following is the result of command "netstat -nlpt | grep 6653" when I activate Optical Network Model:



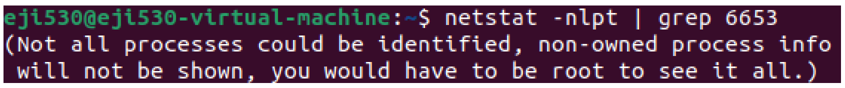
```
eji530@eji530-virtual-machine:~$ netstat -nlpt | grep 6653
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
```

   The following is the result when I activate Host Location Provider:



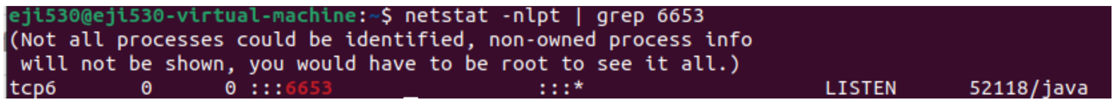
```
eji530@eji530-virtual-machine:~$ netstat -nlpt | grep 6653
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
```

   The following is the result when I activate LLDP Link Provider:



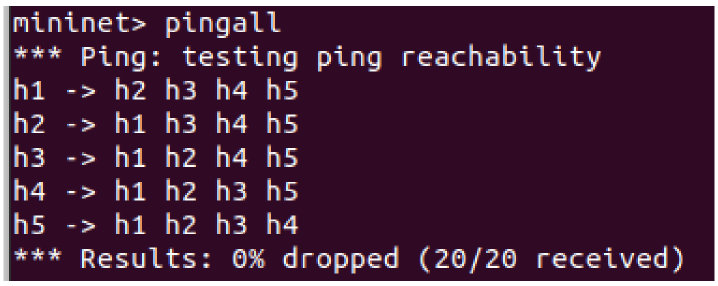
```
eji530@eji530-virtual-machine:~$ netstat -nlpt | grep 6653
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
```

   The following is the result when I activate OpenFlow Base Provider:

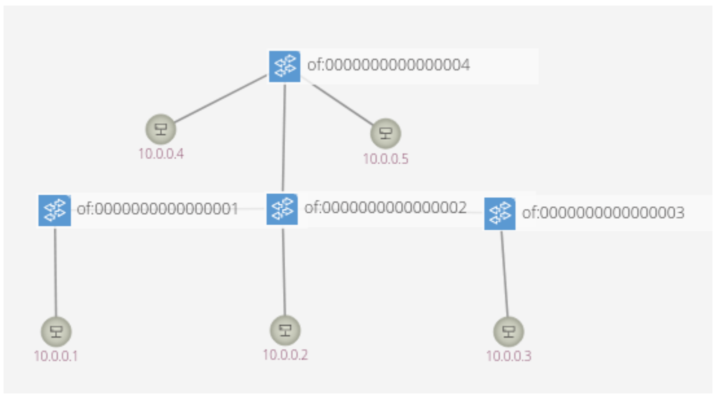


```
eji530@eji530-virtual-machine:~$ netstat -nlpt | grep 6653
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp6       0      0 :::6653                 :::*                    LISTEN      52118/java
```

Part2 take screenshots and explain what I've done

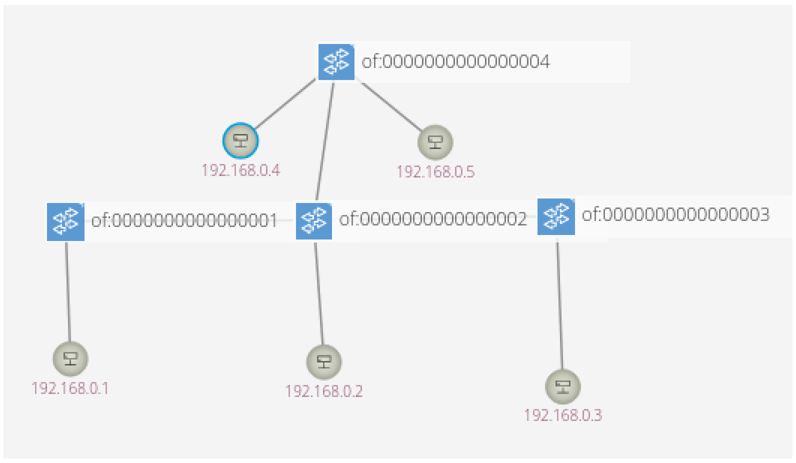


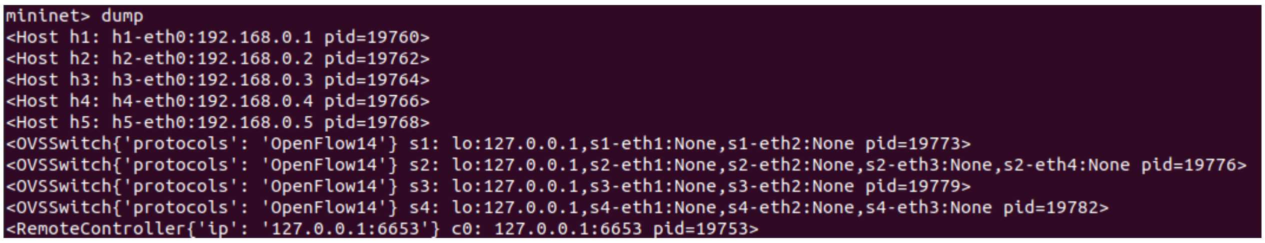
```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
```



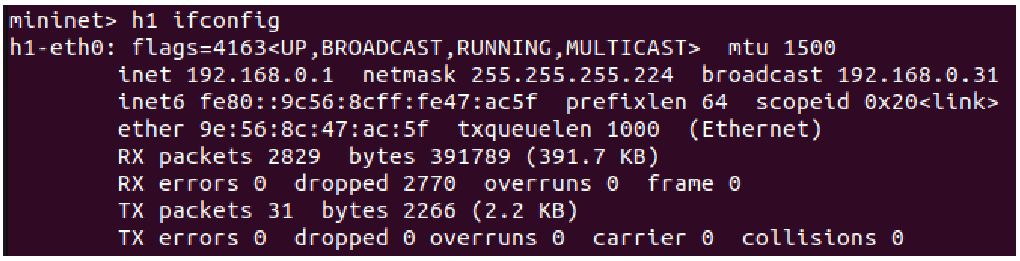
The first screenshot above shows the result when command “pingall” in mininet, and the second screenshot shows topology in ONOS GUI. To create this topology, I write a python script to define the 4 switches and 5 hosts, then use self.addLink to connect them together. Save the scripts and command the command TAs provided in the spec(p.27) to execute the script.

Part3 take screenshots and explain what I’ve done

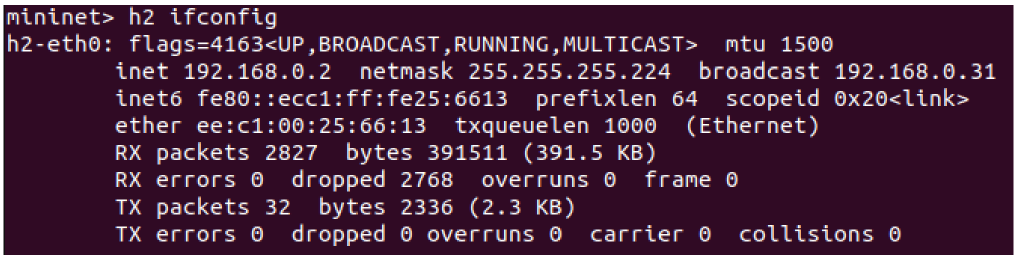




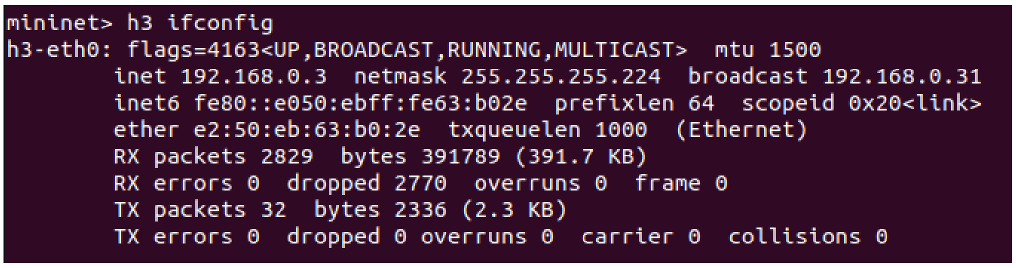
```
mininet> dump
<Host h1: h1-eth0:192.168.0.1 pid=19760>
<Host h2: h2-eth0:192.168.0.2 pid=19762>
<Host h3: h3-eth0:192.168.0.3 pid=19764>
<Host h4: h4-eth0:192.168.0.4 pid=19766>
<Host h5: h5-eth0:192.168.0.5 pid=19768>
<OVSSwitch{'protocols': 'OpenFlow14'} s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=19773>
<OVSSwitch{'protocols': 'OpenFlow14'} s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None,s2-eth4:None pid=19776>
<OVSSwitch{'protocols': 'OpenFlow14'} s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None pid=19779>
<OVSSwitch{'protocols': 'OpenFlow14'} s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None,s4-eth3:None pid=19782>
<RemoteController{'ip': '127.0.0.1:6653'} c0: 127.0.0.1:6653 pid=19753>
```



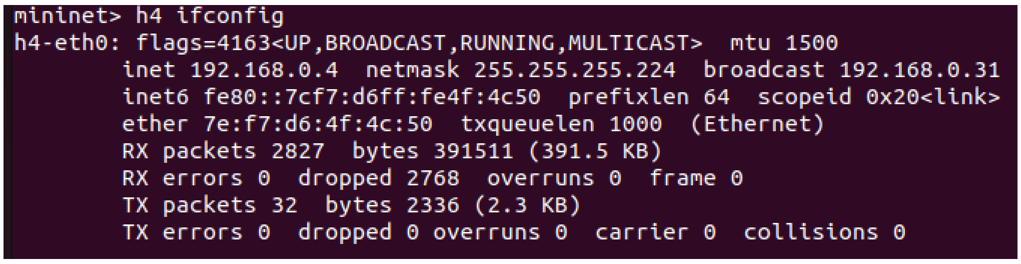
```
mininet> h1 ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.1  netmask 255.255.255.224  broadcast 192.168.0.31
        inet6 fe80::9c56:8cff:fe47:ac5f  prefixlen 64  scopeid 0x20<link>
        ether 9e:56:8c:47:ac:5f  txqueuelen 1000  (Ethernet)
        RX packets 2829  bytes 391789 (391.7 KB)
        RX errors 0  dropped 2770  overruns 0  frame 0
        TX packets 31  bytes 2266 (2.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```



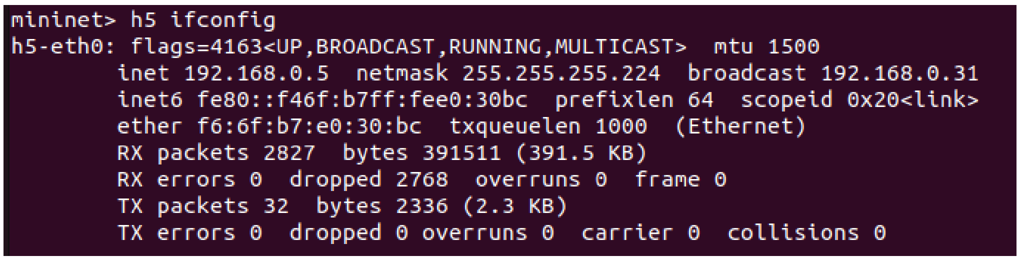
```
mininet> h2 ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.2  netmask 255.255.255.224  broadcast 192.168.0.31
        inet6 fe80::ecc1:ff:fe25:6613  prefixlen 64  scopeid 0x20<link>
        ether ee:c1:00:25:66:13  txqueuelen 1000  (Ethernet)
        RX packets 2827  bytes 391511 (391.5 KB)
        RX errors 0  dropped 2768  overruns 0  frame 0
        TX packets 32  bytes 2336 (2.3 KB)
        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::e050:ebff:fe63:b02e  prefixlen 64  scopeid 0x20<link>
        ether e2:50:eb:63:b0:2e  txqueuelen 1000  (Ethernet)
        RX packets 2829  bytes 391789 (391.7 KB)
        RX errors 0  dropped 2770  overruns 0  frame 0
        TX packets 32  bytes 2336 (2.3 KB)
        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::7cf7:d6ff:fe4f:4c50  prefixlen 64  scopeid 0x20<link>
        ether 7e:f7:d6:4f:4c:50  txqueuelen 1000  (Ethernet)
        RX packets 2827  bytes 391511 (391.5 KB)
        RX errors 0  dropped 2768  overruns 0  frame 0
        TX packets 32  bytes 2336 (2.3 KB)
        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::f46f:b7ff:fee0:30bc  prefixlen 64  scopeid 0x20<link>
        ether f6:6f:b7:e0:30:bc  txqueuelen 1000  (Ethernet)
        RX packets 2827  bytes 391511 (391.5 KB)
        RX errors 0  dropped 2768  overruns 0  frame 0
        TX packets 32  bytes 2336 (2.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

The first screenshot shows the topology, the second screenshot shows the result when command "dump" in mininet. And screenshots 3-7 show the result when command "ifconfig" for all host. To create this topology, use a python script which is similar to part2, and when adding hosts, specify its IP address simultaneously, such as, "h1 = self.addHost('h1', ip='192.168.0.1/27')" .

Part4 what I've learned / solved

In this lab, I have learned the basic usage of ONOS and mininet and build a simple network topology on my own. The following are two problems I met:

1. When running bazel run onos-local – clean debug, I found that its very time consuming, and it turns out that it was because the number of CPU I configure on this virtual machine is too small. After I adjust the CPU number,

this command ran much faster.

2. When answering part1 question 4, I want to deactivate apps in succession to observe which app enables the controller to listen on the TCP port. However, I found that the apps have dependencies, when I deactivate one app (eg: LLDP Location Provider), then all the apps related to org.onosproject.openflow will be deactivated also. To solve this problem, I deactivate all the apps first, then activate them one by one to observe the result instead.