Lab1 ONOS and Mininet Installation

> Student ID: 313551097
> Student Name: 鄭淮薰

Part1: Answer Questions

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

When ONOS activate "org.onosproject.openflow," it activates the following APPs:

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

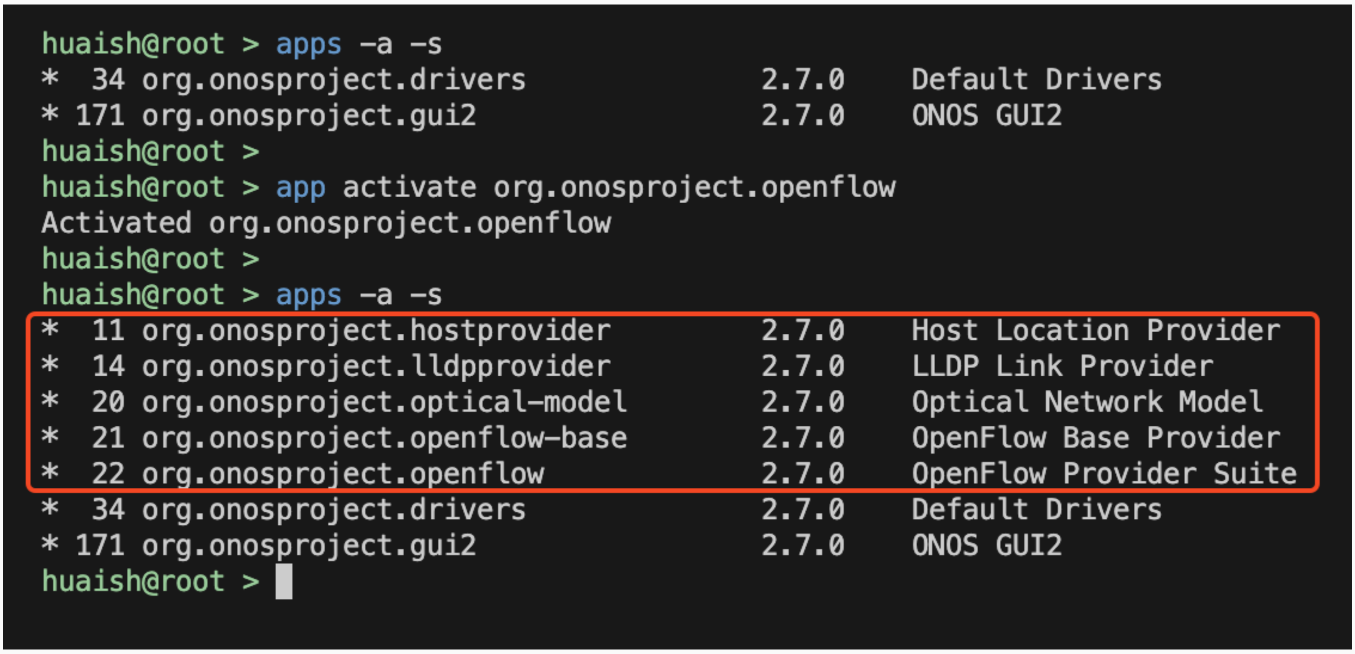
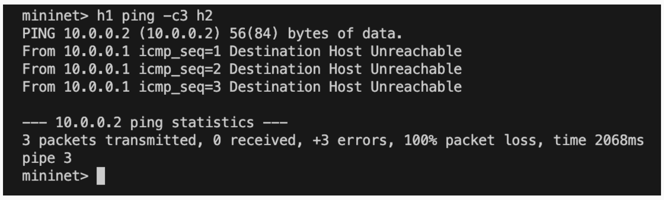


Figure 1: Activated APPs

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

Answer: No, H1 cannot ping H2 successfully.

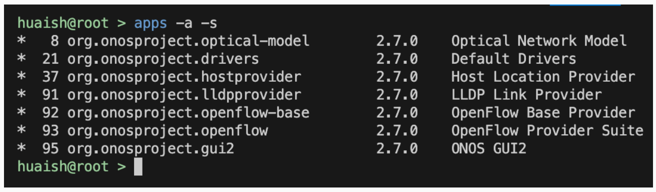
As shown in Figure 2, the ping fails because no flow rules are set on the data plane, so the switch doesn't know how to forward packets between H1 and H2. While ONOS has the Reactive Forwarding app (`org.onosproject.fwd`) to handle this automatically, it's turned off by default, as confirmed in Figure 3. That's why the ping doesn't go through.



```
mininet> h1 ping -c3 h2
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

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2068ms
pipe 3
mininet>
```

Figure 2: Ping command result



```
huaish@root > apps -a -s
*   8 org.onosproject.optical-model        2.7.0    Optical Network Model
*  21 org.onosproject.drivers              2.7.0    Default Drivers
*  37 org.onosproject.hostprovider         2.7.0    Host Location Provider
*  91 org.onosproject.lldpprovider         2.7.0    LLDP Link Provider
*  92 org.onosproject.openflow-base        2.7.0    OpenFlow Base Provider
*  93 org.onosproject.openflow             2.7.0    OpenFlow Provider Suite
*  95 org.onosproject.gui2                 2.7.0    ONOS GUI2
huaish@root >
```

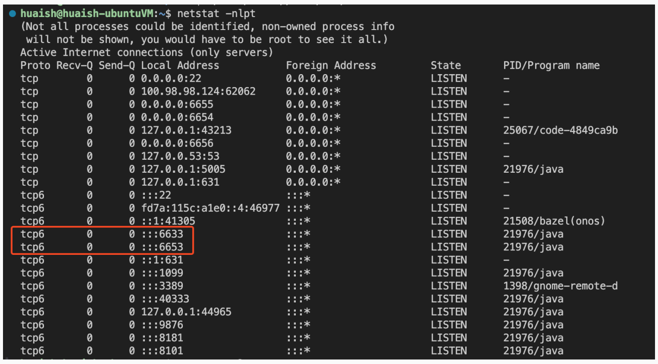
Figure 3: Default activated APPs

> Reference: there are no flows installed on the data-plane, which forward the traffic appropriately. -- **Basic ONOS tutorial**

3. Which TCP port does the controller listen to the OpenFlow connection request from the switch?

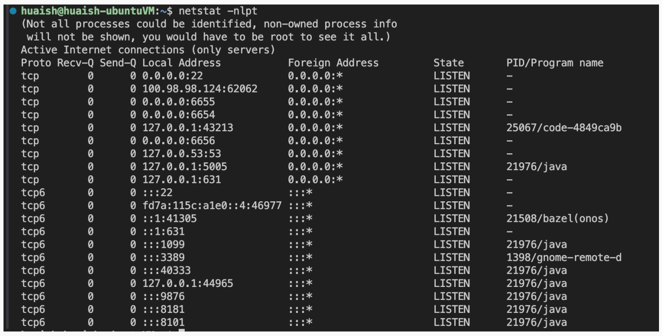
Answer: 6653

In Figures 4.1 and 4.2, I compared the port states before and after starting the OpenFlow app and found that ports `6633` and `6653` are open only when the OpenFlow app is activated. This indicates that both ports may be used for OpenFlow connections.



```
huaish@huaish-ubuntuVM:~$ netstat -nlpt
(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 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 100.98.98.124:62062     0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:6655            0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:6654            0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:43213         0.0.0.0:*               LISTEN      25067/code-4849ca9b
tcp        0      0 0.0.0.0:6656            0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:5005          0.0.0.0:*               LISTEN      21976/java
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
tcp6       0      0 fd7a:115c:a1e0::4:46977 :::*                    LISTEN      -
tcp6       0      0 ::1:41305               :::*                    LISTEN      21508/bazel(onos)
tcp6       0      0 :::6633                 :::*                    LISTEN      21976/java
tcp6       0      0 :::6653                 :::*                    LISTEN      21976/java
tcp6       0      0 ::1:631                 :::*                    LISTEN      -
tcp6       0      0 :::1099                 :::*                    LISTEN      21976/java
tcp6       0      0 :::3389                 :::*                    LISTEN      1398/gnome-remote-d
tcp6       0      0 :::40333                :::*                    LISTEN      21976/java
tcp6       0      0 127.0.0.1:44965         :::*                    LISTEN      21976/java
tcp6       0      0 :::9876                 :::*                    LISTEN      21976/java
tcp6       0      0 :::8181                 :::*                    LISTEN      21976/java
tcp6       0      0 :::8101                 :::*                    LISTEN      21976/java
```

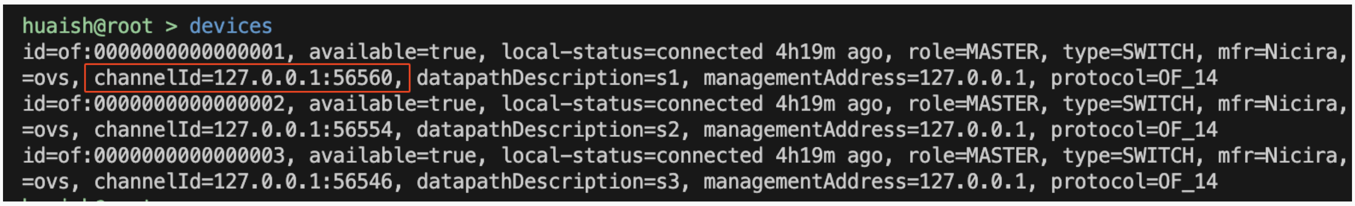
Figure 4.1: Port listening state (activating OpenFlow app)



```
huaish@huaish-ubuntuVM:~$ netstat -nlpt
(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 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 100.98.98.124:62062     0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:6655            0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:6654            0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:43213         0.0.0.0:*               LISTEN      25067/code-4849ca9b
tcp        0      0 0.0.0.0:6656            0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:5005          0.0.0.0:*               LISTEN      21976/java
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
tcp6       0      0 fd7a:115c:a1e0::4:46977 :::*                    LISTEN      -
tcp6       0      0 ::1:41305               :::*                    LISTEN      21508/bazel(onos)
tcp6       0      0 ::1:631                 :::*                    LISTEN      -
tcp6       0      0 :::1099                 :::*                    LISTEN      21976/java
tcp6       0      0 :::3389                 :::*                    LISTEN      1398/gnome-remote-d
tcp6       0      0 :::40333                :::*                    LISTEN      21976/java
tcp6       0      0 127.0.0.1:44965         :::*                    LISTEN      21976/java
tcp6       0      0 :::9876                 :::*                    LISTEN      21976/java
tcp6       0      0 :::8181                 :::*                    LISTEN      21976/java
tcp6       0      0 :::8101                 :::*                    LISTEN      21976/java
```

Figure 4.2: Port listening state (deactivating OpenFlow app)

Figure 5 shows that the switch *s1* communicates with the controller via port `56560`.



```
huaish@root > devices
id=of:0000000000000001, available=true, local-status=connected 4h19m ago, role=MASTER, type=SWITCH, mfr=Nicira,
=ovs, channelId=127.0.0.1:56560, datapathDescription=s1, managementAddress=127.0.0.1, protocol=OF_14
id=of:0000000000000002, available=true, local-status=connected 4h19m ago, role=MASTER, type=SWITCH, mfr=Nicira,
=ovs, channelId=127.0.0.1:56554, datapathDescription=s2, managementAddress=127.0.0.1, protocol=OF_14
id=of:0000000000000003, available=true, local-status=connected 4h19m ago, role=MASTER, type=SWITCH, mfr=Nicira,
=ovs, channelId=127.0.0.1:56546, datapathDescription=s3, managementAddress=127.0.0.1, protocol=OF_14
```

Figure 5: Switch *s1* channelId

As shown in Figure 6, the tshark capture result confirms that the controller listens on port `6653` for OpenFlow connections.

```
huaish@huaish-ubuntuVM:~$ sudo tshark -i any -f "port 56560" -c 10
Running as user "root" and group "root". This could be dangerous.
Capturing on 'any'
 ** (tshark:35253) 22:42:21.542661 [Main MESSAGE] -- Capture started.
 ** (tshark:35253) 22:42:21.542779 [Main MESSAGE] -- File: "/tmp/wireshark_anyCU2XT2.pcapng"
    1 0.000000000    127.0.0.1 → 127.0.0.1    OpenFlow 132 Type: OFPT_MULTIPART_REQUEST
    2 0.000165547    127.0.0.1 → 127.0.0.1    OpenFlow 84 Type: OFPT_MULTIPART_REPLY
    3 0.000193424    127.0.0.1 → 127.0.0.1    TCP 68 6653 → 56560 [ACK] Seq=65 Ack=17 Win=86 Len=0 TSval=3083074907 TSecr=3083074907
    4 0.000206794    127.0.0.1 → 127.0.0.1    OpenFlow 84 Type: OFPT_MULTIPART_REPLY
    5 0.000209600    127.0.0.1 → 127.0.0.1    TCP 68 6653 → 56560 [ACK] Seq=65 Ack=33 Win=86 Len=0 TSval=3083074907 TSecr=3083074907
    6 0.000218835    127.0.0.1 → 127.0.0.1    OpenFlow 84 Type: OFPT_MULTIPART_REPLY
    7 0.000220419    127.0.0.1 → 127.0.0.1    TCP 68 6653 → 56560 [ACK] Seq=65 Ack=49 Win=86 Len=0 TSval=3083074907 TSecr=3083074907
    8 0.400488562    127.0.0.1 → 127.0.0.1    OpenFlow 784 Type: OFPT_PACKET_OUT
    9 0.400959206    127.0.0.1 → 127.0.0.1    OpenFlow 249 Type: OFPT_PACKET_IN
   10 0.400974245    127.0.0.1 → 127.0.0.1    TCP 68 6653 → 56560 [ACK] Seq=781 Ack=230 Win=86 Len=0 TSval=3083075308 TSecr=3083075308
10 packets captured
```

Figure 6: tshark capture result

> Reference: Controllers should listen on TCP port 6653 for switches that want to set up a connection. Earlier versions of the OpenFlow protocol unofficially used port 6633 -- **OpenFlow - Wikipedia**

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

Answer: OpenFlow Base Provider (org.onosproject.openflow-base)

1. When no APPs are deactivated (both **openflow-base** and **openflow** are activated), the controller listens on port `6653` for OpenFlow connections, as shown the top of Figure 7 (part 1).
2. After deactivating the **openflow-base** app (both **openflow-base** and its dependencies are deactivated), we can see that port `6653` is no longer listening, as shown in the middle of Figure 7 (part 2).
3. When we activate the **openflow-base** app again (only **openflow-base** and its dependencies is activated), the controller listens on port `6653` again, as shown at the middle of Figure 7 (part 3).
4. When we deactivate the **openflow-base** but activate the **org.onosproject.optical-model** app which is the dependency of **openflow-base**, we can see that port `6653` is closed, as shown at the bottom of Figure 7 (part 4).

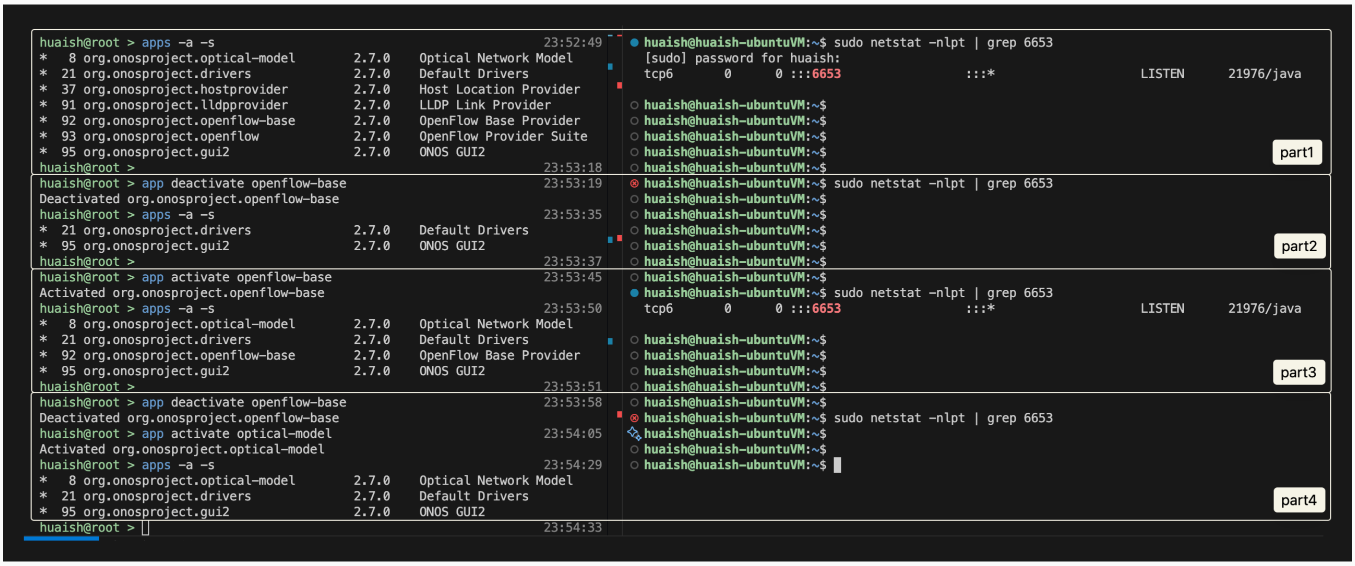
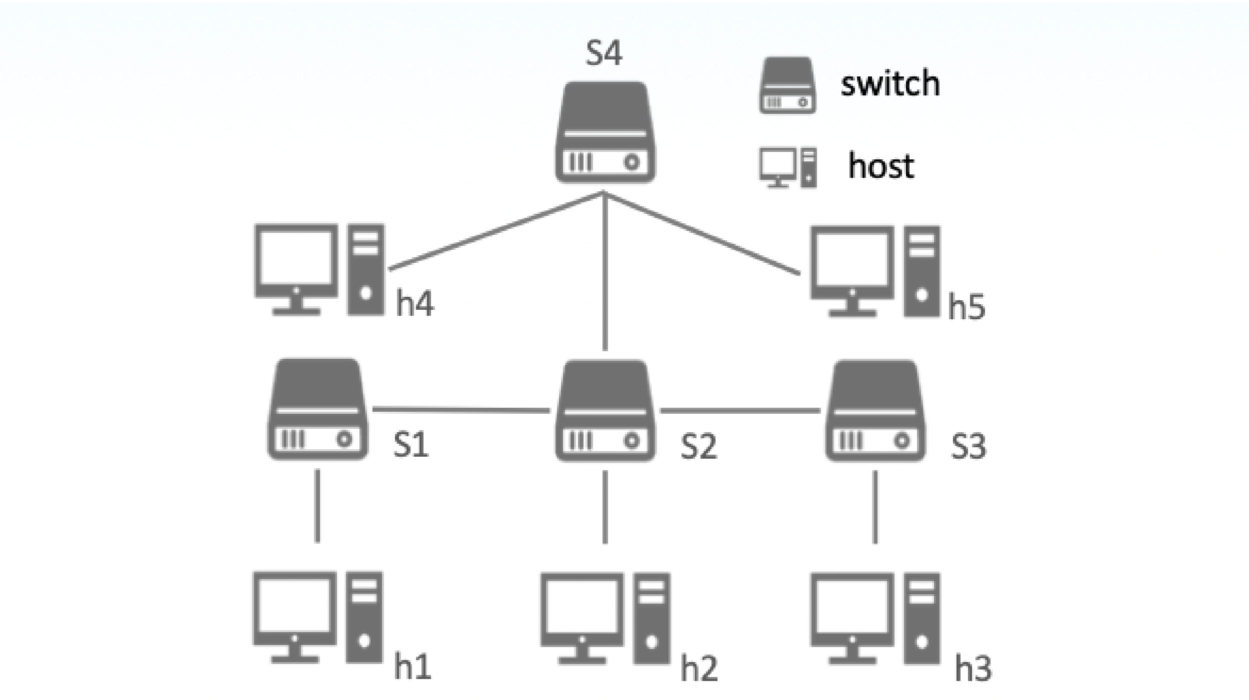


Figure 7: Deactivate/Activate APPs and check port status

From the above observations, we can conclude that the **openflow-base** app enables the controller to listen on the TCP port.

Part2: Create a custom Topology



I created above custom topology with the following steps:

1. Add 5 hosts: `h1`, `h2`, `h3`, `h4`, `h5`
2. Add 4 switches: `S1`, `S2`, `S3`, `S4`
3. Add links between hosts and switches

The implementation of the custom topology is written in the Python script `lab1_part2_313551097.py`.

Result:

Figure 8 shows the custom topology on GUI after running the script.

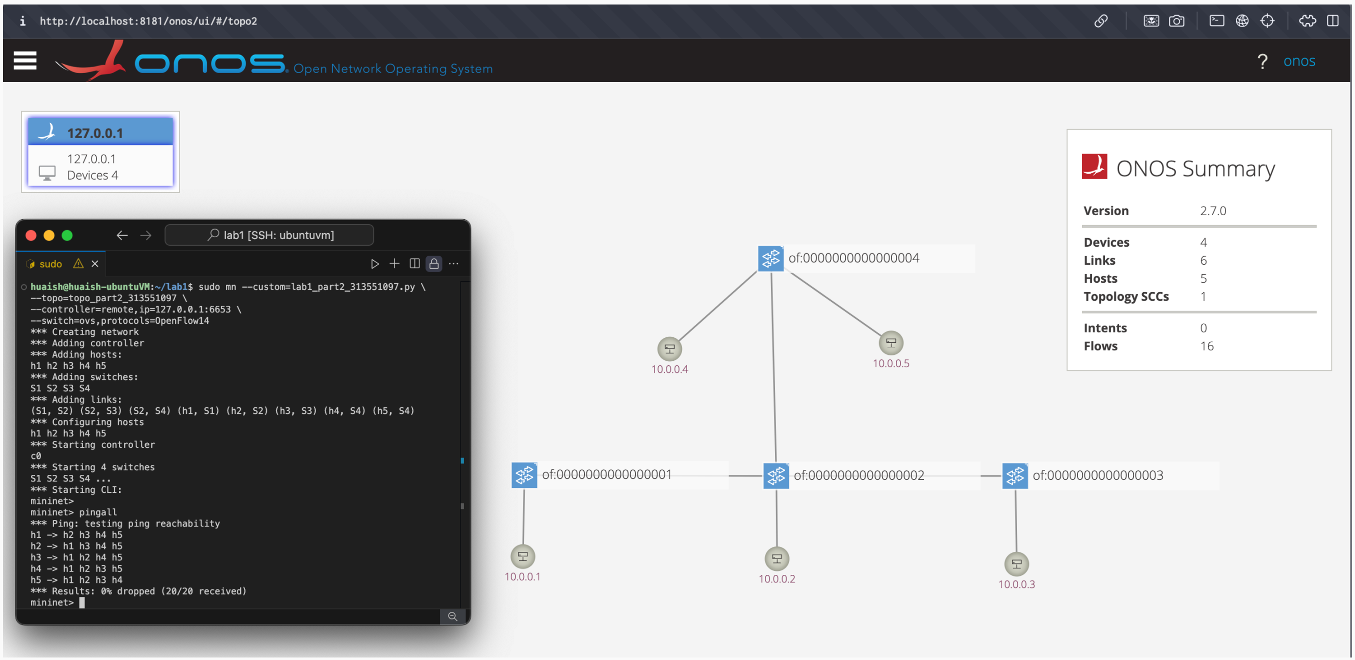


Figure 8: Custom Topology on GUI

Part3: Statically assign Hosts IP Address in Mininet

I manually assigned IP addresses as follows:

- IP addresses: 192.168.0.0/27

Host	IP Address
h1	192.168.0.1
h2	192.168.0.2
h3	192.168.0.3
h4	192.168.0.4
h5	192.168.0.5

- netmask: 255.255.255.244

To manually assign IP addresses to each host, we can assign the IP address when creating the host. For example:

```
h1 = net.addHost('h1', ip='192.168.0.1/27')
h2 = net.addHost('h2', ip='192.168.0.2/27')
```

In this case, `/27` represents the netmask, which means the first 27 bits are network bits, and the remaining 5 bits are host bits. The binary representation of the netmask is `11111111.11111111.11111111.11100000`, which corresponds to `255.255.255.224` in decimal.

Thus, the subnet mask for `/27` is **`255.255.255.224`**.

The implementation of the custom topology with static IP addresses is written in the Python script `lab1_part3_313551097.py`.

Result:

Figure 9 shows the result of running the script.

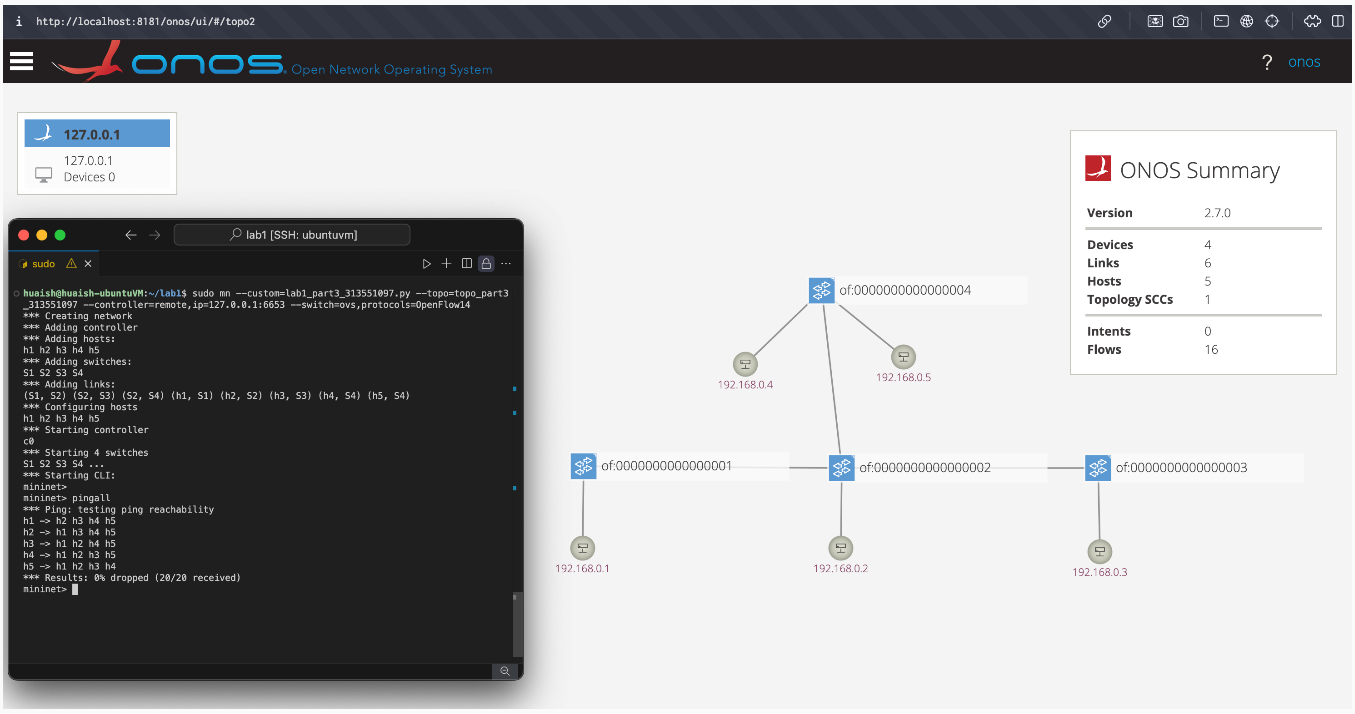


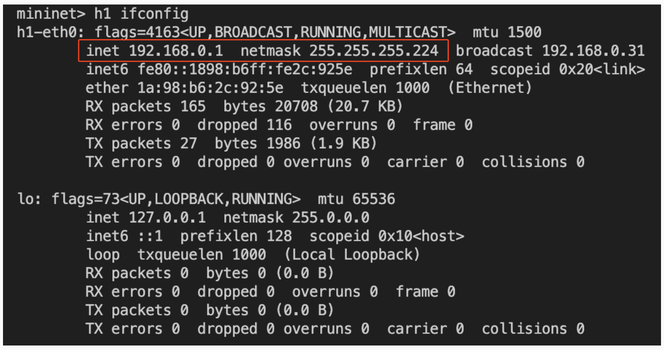
Figure 9: Result of running script `lab1_part3_313551097.py`

Figure 10 shows the result of running the `dump` command.

```
mininet> dump
<Host h1: h1-eth0:192.168.0.1 pid=47894>
<Host h2: h2-eth0:192.168.0.2 pid=47896>
<Host h3: h3-eth0:192.168.0.3 pid=47898>
<Host h4: h4-eth0:192.168.0.4 pid=47900>
<Host h5: h5-eth0:192.168.0.5 pid=47902>
<OVSSwitch{'protocols': 'OpenFlow14'} S1: lo:127.0.0.1,S1-eth1:None,S1-eth2:None pid=47907>
<OVSSwitch{'protocols': 'OpenFlow14'} S2: lo:127.0.0.1,S2-eth1:None,S2-eth2:None,S2-eth3:None,S2-eth4:None pid=47910>
<OVSSwitch{'protocols': 'OpenFlow14'} S3: lo:127.0.0.1,S3-eth1:None,S3-eth2:None pid=47913>
<OVSSwitch{'protocols': 'OpenFlow14'} S4: lo:127.0.0.1,S4-eth1:None,S4-eth2:None,S4-eth3:None pid=47916>
<RemoteController{'ip': '127.0.0.1:6653'} c0: 127.0.0.1:6653 pid=47887>
mininet>
```

Figure 10: Result of running the `dump` command

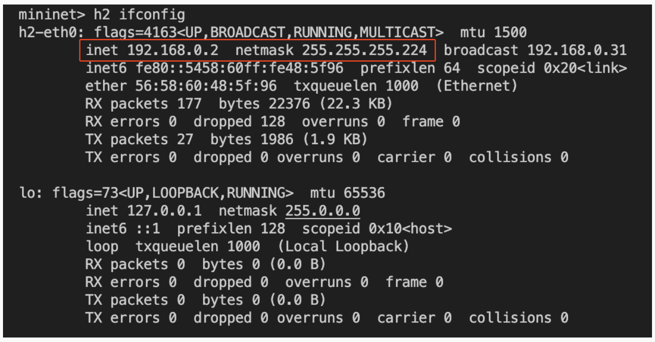
Figure 11.1 to 11.5 show the result of running the `ifconfig` command on each host.



```
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::1898:b6ff:fe2c:925e  prefixlen 64  scopeid 0x20<link>
        ether 1a:98:b6:2c:92:5e  txqueuelen 1000  (Ethernet)
        RX packets 165  bytes 20708 (20.7 KB)
        RX errors 0  dropped 116  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
```

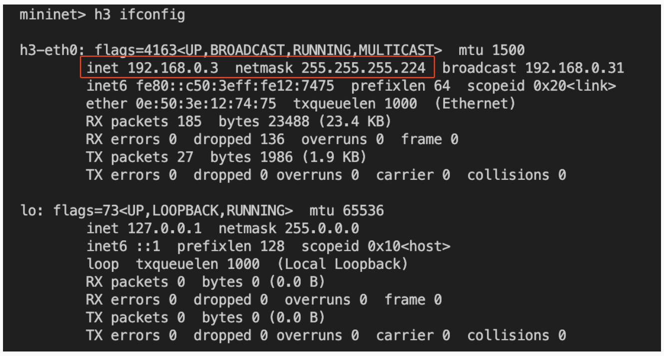
Figure 11.1: h1 ifconfig



```
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::5458:60ff:fe48:5f96  prefixlen 64  scopeid 0x20<link>
        ether 56:58:60:48:5f:96  txqueuelen 1000  (Ethernet)
        RX packets 177  bytes 22376 (22.3 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
```

Figure 11.2: h2 ifconfig

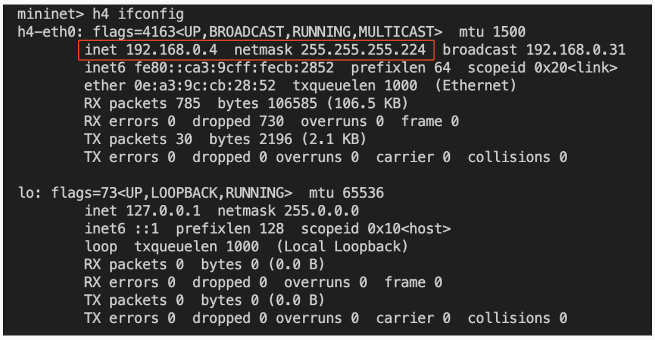


```
mininet> h3 ifconfig

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::c50:3eff:fe12:7475  prefixlen 64  scopeid 0x20<link>
        ether 0e:50:3e:12:74:75  txqueuelen 1000  (Ethernet)
        RX packets 185  bytes 23488 (23.4 KB)
        RX errors 0  dropped 136  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
```

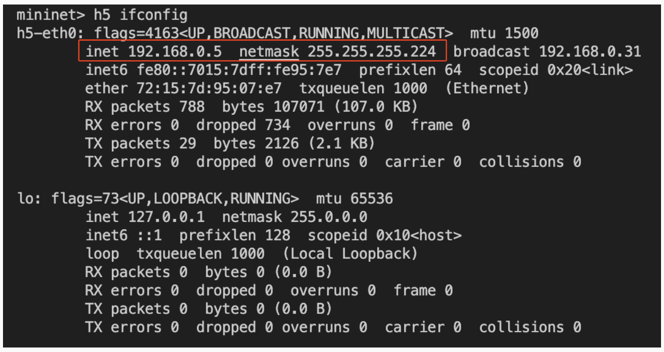
Figure 11.3: h3 ifconfig



```
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::ca3:9cff:fecb:2852  prefixlen 64  scopeid 0x20<link>
        ether 0e:a3:9c:cb:28:52  txqueuelen 1000  (Ethernet)
        RX packets 785  bytes 106585 (106.5 KB)
        RX errors 0  dropped 730  overruns 0  frame 0
        TX packets 30  bytes 2196 (2.1 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 11.4: h4 ifconfig



```
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::7015:7dff:fe95:7e7  prefixlen 64  scopeid 0x20<link>
        ether 72:15:7d:95:07:e7  txqueuelen 1000  (Ethernet)
        RX packets 788  bytes 107071 (107.0 KB)
        RX errors 0  dropped 734  overruns 0  frame 0
        TX packets 29  bytes 2126 (2.1 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 11.5: h5 ifconfig

What you've learned or solved

Lab 1 provided a solid understanding of ONOS and Mininet, covering the relationship between controllers and switches, the role of ONOS apps, and how to create custom topologies. Through hands-on practice, I deepened my knowledge of network technologies, particularly in observing controller-switch interactions and interpreting network connections. Writing Python scripts for custom topologies enhanced my familiarity with Mininet and improved my understanding of IP address allocation in SDN environments. This lab effectively bridged theory and practice, equipping me with skills for configuring network topologies, laying a strong foundation for future network development.