CS3611 Computer Networks (Spring 2023)
Lab 4: Overlay Network and VXLAN

Xiangyuan Xue (521030910387)

1. For Step 1, we add a default router for the bridge. On VM 1, we run the following command.

   ```
   sudo route add default gw 192.168.56.2
   ```

   On VM 2, we run the following command.

   ```
   sudo route add default gw 192.168.56.2
   ```

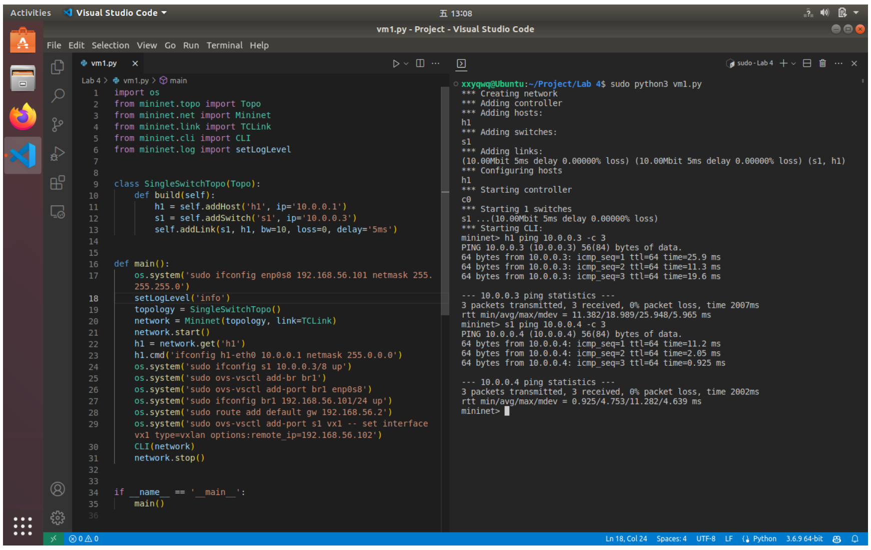
   For Step 2, we build the VXLAN tunnel and set the remote IP address. On VM 1, we run the following command.

   ```
   sudo ovs-vsctl add-port s1 vx1 -- set interface vx1 type=vxlan options
       :remote_ip=192.168.56.102
   ```

   On VM 2, we run the following command.

   ```
   sudo ovs-vsctl add-port s2 vx2 -- set interface vx2 type=vxlan options
       :remote_ip=192.168.56.101
   ```

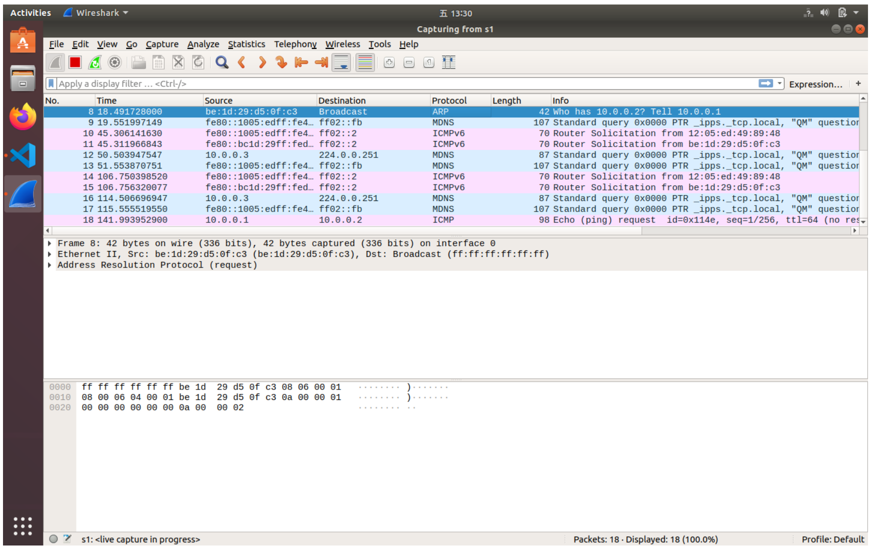
   Then we can ping 10.0.0.2/10.0.0.4 from 10.0.0.1/10.0.0.3 as the figure below.

   
   Activities
Visual Studio Code
五 13:08
vm1.py - Project - Visual Studio Code
File Edit Selection View Go Run Terminal Help
vm1.py
Lab 4 > vm1.py > main
sudo - Lab 4
Ln 18, Col 24 Spaces: 4 UTF-8 LF Python 3.6.9 64-bit

   ```python
   import os
   from mininet.topo import Topo
   from mininet.net import Mininet
   from mininet.link import TCLink
   from mininet.cli import CLI
   from mininet.log import setLogLevel


   class SingleSwitchTopo(Topo):
       def build(self):
           h1 = self.addHost('h1', ip='10.0.0.1')
           s1 = self.addSwitch('s1', ip='10.0.0.3')
           self.addLink(s1, h1, bw=10, loss=0, delay='5ms')


   def main():
       os.system('sudo ifconfig enp0s8 192.168.56.101 netmask 255.255.255.0')
       setLogLevel('info')
       topology = SingleSwitchTopo()
       network = Mininet(topology, link=TCLink)
       network.start()
       h1 = network.get('h1')
       h1.cmd('ifconfig h1-eth0 10.0.0.1 netmask 255.0.0.0')
       os.system('sudo ifconfig s1 10.0.0.3/8 up')
       os.system('sudo ovs-vsctl add-br br1')
       os.system('sudo ovs-vsctl add-port br1 enp0s8')
       os.system('sudo ifconfig br1 192.168.56.101/24 up')
       os.system('sudo route add default gw 192.168.56.2')
       os.system('sudo ovs-vsctl add-port s1 vx1 -- set interface vx1 type=vxlan options:remote_ip=192.168.56.102')
       CLI(network)
       network.stop()


   if __name__ == '__main__':
       main()
   ```

   ```
   xxyqwq@Ubuntu:~/Project/Lab 4$ sudo python3 vm1.py
   *** Creating network
   *** Adding controller
   *** Adding hosts:
   h1
   *** Adding switches:
   s1
   *** Adding links:
   (10.00Mbit 5ms delay 0.00000% loss) (10.00Mbit 5ms delay 0.00000% loss) (s1, h1)
   *** Configuring hosts
   h1
   *** Starting controller
   c0
   *** Starting 1 switches
   s1 ...(10.00Mbit 5ms delay 0.00000% loss)
   *** Starting CLI:
   mininet> h1 ping 10.0.0.3 -c 3
   PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
   64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=25.9 ms
   64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=11.3 ms
   64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=19.6 ms

   --- 10.0.0.3 ping statistics ---
   3 packets transmitted, 3 received, 0% packet loss, time 2007ms
   rtt min/avg/max/mdev = 11.382/18.989/25.948/5.965 ms
   mininet> s1 ping 10.0.0.4 -c 3
   PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
   64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=11.2 ms
   64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=2.05 ms
   64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.925 ms

   --- 10.0.0.4 ping statistics ---
   3 packets transmitted, 3 received, 0% packet loss, time 2002ms
   rtt min/avg/max/mdev = 0.925/4.753/11.282/4.639 ms
   mininet>
   ```

2. Use Wireshark to monitor the interfaces s1 and enp0s8 on VM 1. The results are shown in the following figures.



Capturing from s1

Apply a display filter … <Ctrl-/>

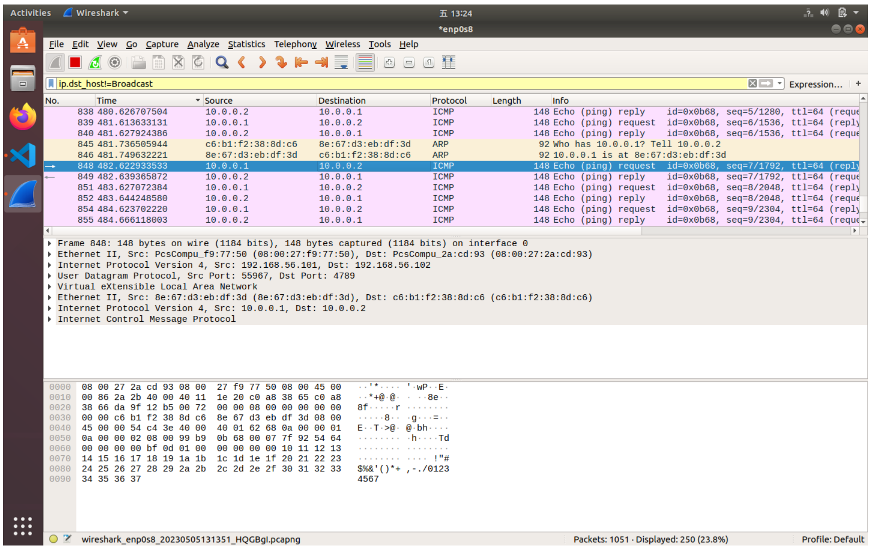
```
No.   Time            Source                 Destination    Protocol  Length  Info
    8 18.491728000    be:1d:29:d5:0f:c3      Broadcast      ARP           42 Who has 10.0.0.2? Tell 10.0.0.1
    9 19.551997149    fe80::1005:edff:fe4…   ff02::fb       MDNS         107 Standard query 0x0000 PTR _ipps._tcp.local, "QM" questior
   10 45.306141630    fe80::1005:edff:fe4…   ff02::2        ICMPv6        70 Router Solicitation from 12:05:ed:49:89:48
   11 45.311966843    fe80::bc1d:29ff:fed…   ff02::2        ICMPv6        70 Router Solicitation from be:1d:29:d5:0f:c3
   12 50.503947547    10.0.0.3               224.0.0.251    MDNS          87 Standard query 0x0000 PTR _ipps._tcp.local, "QM" questior
   13 51.553870751    fe80::1005:edff:fe4…   ff02::fb       MDNS         107 Standard query 0x0000 PTR _ipps._tcp.local, "QM" questior
   14 106.750398520   fe80::1005:edff:fe4…   ff02::2        ICMPv6        70 Router Solicitation from 12:05:ed:49:89:48
   15 106.756320077   fe80::bc1d:29ff:fed…   ff02::2        ICMPv6        70 Router Solicitation from be:1d:29:d5:0f:c3
   16 114.506696947   10.0.0.3               224.0.0.251    MDNS          87 Standard query 0x0000 PTR _ipps._tcp.local, "QM" questior
   17 115.555519550   fe80::1005:edff:fe4…   ff02::fb       MDNS         107 Standard query 0x0000 PTR _ipps._tcp.local, "QM" questior
   18 141.993952900   10.0.0.1               10.0.0.2       ICMP          98 Echo (ping) request  id=0x114e, seq=1/256, ttl=64 (no res
```

```
▸ Frame 8: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▸ Ethernet II, Src: be:1d:29:d5:0f:c3 (be:1d:29:d5:0f:c3), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ Address Resolution Protocol (request)
```

```
0000  ff ff ff ff ff ff be 1d  29 d5 0f c3 08 06 00 01   ········ )·······
0010  08 00 06 04 00 01 be 1d  29 d5 0f c3 0a 00 00 01   ········ )·······
0020  00 00 00 00 00 00 0a 00  00 02                     ········ ··
```

s1: <live capture in progress> Packets: 18 · Displayed: 18 (100.0%) Profile: Default

Activities
Wireshark
五 13:30
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression…



*enp0s8

ip.dst_host!=Broadcast

```
No.   Time            Source                 Destination            Protocol  Length  Info
  838 480.626707504   10.0.0.2               10.0.0.1               ICMP         148 Echo (ping) reply    id=0x0b68, seq=5/1280, ttl=64 (reque
  839 481.613633131   10.0.0.1               10.0.0.2               ICMP         148 Echo (ping) request  id=0x0b68, seq=6/1536, ttl=64 (reply
  840 481.627924386   10.0.0.2               10.0.0.1               ICMP         148 Echo (ping) reply    id=0x0b68, seq=6/1536, ttl=64 (reque
  845 481.736505944   c6:b1:f2:38:8d:c6      8e:67:d3:eb:df:3d      ARP           92 Who has 10.0.0.1? Tell 10.0.0.2
  846 481.749632221   8e:67:d3:eb:df:3d      c6:b1:f2:38:8d:c6      ARP           92 10.0.0.1 is at 8e:67:d3:eb:df:3d
→ 848 482.622933533   10.0.0.1               10.0.0.2               ICMP         148 Echo (ping) request  id=0x0b68, seq=7/1792, ttl=64 (reply
← 849 482.639365872   10.0.0.2               10.0.0.1               ICMP         148 Echo (ping) reply    id=0x0b68, seq=7/1792, ttl=64 (reque
  851 483.627072384   10.0.0.1               10.0.0.2               ICMP         148 Echo (ping) request  id=0x0b68, seq=8/2048, ttl=64 (reply
  852 483.644248580   10.0.0.2               10.0.0.1               ICMP         148 Echo (ping) reply    id=0x0b68, seq=8/2048, ttl=64 (reque
  854 484.623702220   10.0.0.1               10.0.0.2               ICMP         148 Echo (ping) request  id=0x0b68, seq=9/2304, ttl=64 (reply
  855 484.666118003   10.0.0.2               10.0.0.1               ICMP         148 Echo (ping) reply    id=0x0b68, seq=9/2304, ttl=64 (reque
```

```
▸ Frame 848: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_f9:77:50 (08:00:27:f9:77:50), Dst: PcsCompu_2a:cd:93 (08:00:27:2a:cd:93)
▸ Internet Protocol Version 4, Src: 192.168.56.101, Dst: 192.168.56.102
▸ User Datagram Protocol, Src Port: 55967, Dst Port: 4789
▸ Virtual eXtensible Local Area Network
▸ Ethernet II, Src: 8e:67:d3:eb:df:3d (8e:67:d3:eb:df:3d), Dst: c6:b1:f2:38:8d:c6 (c6:b1:f2:38:8d:c6)
▸ Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.2
▸ Internet Control Message Protocol
```

```
0000  08 00 27 2a cd 93 08 00  27 f9 77 50 08 00 45 00   ··'*···· '·wP··E·
0010  00 86 2a 2b 40 00 40 11  1e 20 c0 a8 38 65 c0 a8   ··*+@·@·  ··8e··
0020  38 66 da 9f 12 b5 00 72  00 00 08 00 00 00 00 00   8f·····r ········
0030  00 00 c6 b1 f2 38 8d c6  8e 67 d3 eb df 3d 08 00   ·····8·· ·g···=··
0040  45 00 00 54 c4 3e 40 00  40 01 62 68 0a 00 00 01   E··T·>@· @·bh····
0050  0a 00 00 02 08 00 99 b9  0b 68 00 07 7f 92 54 64   ········ ·h····Td
0060  00 00 00 00 bf 0d 01 00  00 00 00 00 10 11 12 13   ········ ········
0070  14 15 16 17 18 19 1a 1b  1c 1d 1e 1f 20 21 22 23   ········ ···· !"#
0080  24 25 26 27 28 29 2a 2b  2c 2d 2e 2f 30 31 32 33   $%&'()*+ ,-./0123
0090  34 35 36 37                                        4567
```

wireshark_enp0s8_20230505131351_HQGBgI.pcapng Packets: 1051 · Displayed: 250 (23.8%) Profile: Default

Activities
Wireshark
五 13:24
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression…

Explicitly, Address Resolution Protocol (ARP) and Internet Control Message Protocol (ICMP) are used in this procedure. In addition, Internet Protocol version 4 (IPv4) is implicitly and User Datagram Protocol (UDP) is encapsulated inside the VXLAN packet. Detailed descriptions are as follows.

- ARP: *ARP is a communication protocol used for discovering the link layer address, such as a MAC address, associated with a given internet layer address, typically an IPv4 address. This mapping is a critical function in the Internet protocol suite. ARP was defined in 1982 by RFC 826, which is Internet Standard STD 37.*[1]
- ICMP: *ICMP is a supporting protocol in the Internet protocol suite. It is used by network devices, including routers, to send error messages and operational information indicating success or failure when communicating with another IP address, for example, an error is indicated when a requested service is not available or that a host or router could not be reached.*[2]

- IPv4: *IPv4 is the fourth version of the Internet Protocol (IP). It is one of the core protocols of standards-based internetworking methods in the Internet and other packet-switched networks. IPv4 was the first version deployed for production on SATNET in 1982 and on the ARPANET in January 1983.*[3]
- UDP: *UDP is one of the core communication protocols of the Internet protocol suite used to send messages (transported as datagrams in packets) to other hosts on an Internet Protocol (IP) network. Within an IP network, UDP does not require prior communication to set up communication channels or data paths.*[4]

The workflow of VXLAN is specified as follows: The original packet is passed to the VXLAN Tunnel Endpoints (VTEP). The VTEP adds the VXLAN header and the UDP header to the original packet and sends the packet to the destination VTEP. After receiving the packet, the destination VTEP removes the UDP header and sends the original packet to the destination host according to the VXLAN Network Identifier (VNI) in the VXLAN header.

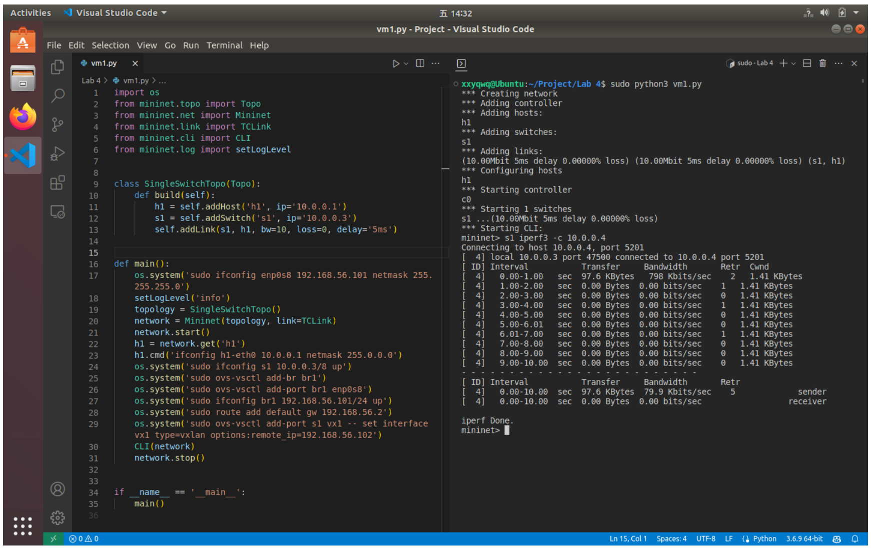
3. Start iperf3 server on VM 2. We run the following command on s2.

```
iperf3 -s
```

Start iperf3 client on VM 1. We run the following command on s1.

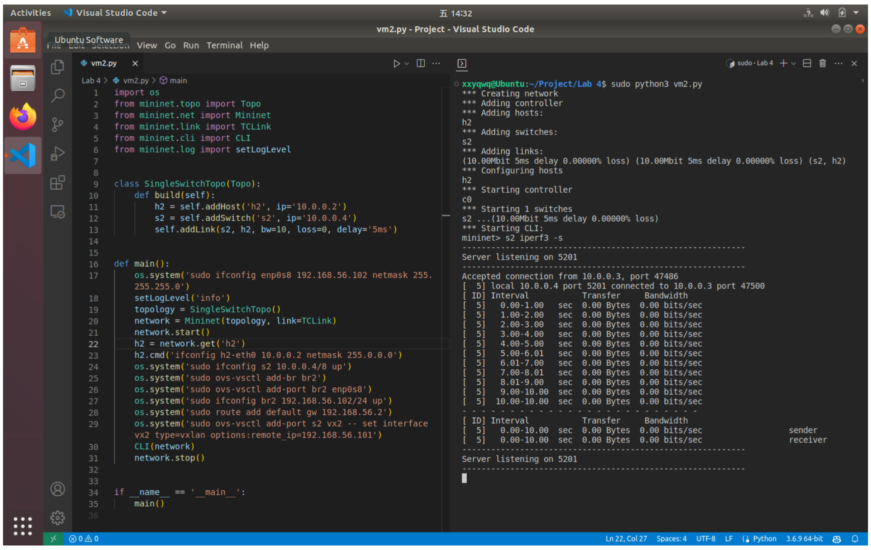
```
iperf3 -c 10.0.0.4
```

The results are shown in the following figures.


Activities
Visual Studio Code
五 14:32
vm1.py - Project - Visual Studio Code
File Edit Selection View Go Run Terminal Help
vm1.py
Lab 4 > vm1.py > ...
sudo - Lab 4
Ln 15, Col 1 Spaces: 4 UTF-8 LF Python 3.6.9 64-bit

```python
import os
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.link import TCLink
from mininet.cli import CLI
from mininet.log import setLogLevel


class SingleSwitchTopo(Topo):
    def build(self):
        h1 = self.addHost('h1', ip='10.0.0.1')
        s1 = self.addSwitch('s1', ip='10.0.0.3')
        self.addLink(s1, h1, bw=10, loss=0, delay='5ms')


def main():
    os.system('sudo ifconfig enp0s8 192.168.56.101 netmask 255.255.255.0')
    setLogLevel('info')
    topology = SingleSwitchTopo()
    network = Mininet(topology, link=TCLink)
    network.start()
    h1 = network.get('h1')
    h1.cmd('ifconfig h1-eth0 10.0.0.1 netmask 255.0.0.0')
    os.system('sudo ifconfig s1 10.0.0.3/8 up')
    os.system('sudo ovs-vsctl add-br br1')
    os.system('sudo ovs-vsctl add-port br1 enp0s8')
    os.system('sudo ifconfig br1 192.168.56.101/24 up')
    os.system('sudo route add default gw 192.168.56.2')
    os.system('sudo ovs-vsctl add-port s1 vx1 -- set interface vx1 type=vxlan options:remote_ip=192.168.56.102')
    CLI(network)
    network.stop()


if __name__ == '__main__':
    main()
```

```
xxyqwq@Ubuntu:~/Project/Lab 4$ sudo python3 vm1.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1
*** Adding switches:
s1
*** Adding links:
(10.00Mbit 5ms delay 0.00000% loss) (10.00Mbit 5ms delay 0.00000% loss) (s1, h1)
*** Configuring hosts
h1
*** Starting controller
c0
*** Starting 1 switches
s1 ...(10.00Mbit 5ms delay 0.00000% loss)
*** Starting CLI:
mininet> s1 iperf3 -c 10.0.0.4
Connecting to host 10.0.0.4, port 5201
[  4] local 10.0.0.3 port 47500 connected to 10.0.0.4 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec  97.6 KBytes   798 Kbits/sec    2   1.41 KBytes
[  4]   1.00-2.00   sec  0.00 Bytes  0.00 bits/sec    1   1.41 KBytes
[  4]   2.00-3.00   sec  0.00 Bytes  0.00 bits/sec    0   1.41 KBytes
[  4]   3.00-4.00   sec  0.00 Bytes  0.00 bits/sec    1   1.41 KBytes
[  4]   4.00-5.00   sec  0.00 Bytes  0.00 bits/sec    0   1.41 KBytes
[  4]   5.00-6.01   sec  0.00 Bytes  0.00 bits/sec    0   1.41 KBytes
[  4]   6.01-7.00   sec  0.00 Bytes  0.00 bits/sec    1   1.41 KBytes
[  4]   7.00-8.00   sec  0.00 Bytes  0.00 bits/sec    0   1.41 KBytes
[  4]   8.00-9.00   sec  0.00 Bytes  0.00 bits/sec    0   1.41 KBytes
[  4]   9.00-10.00  sec  0.00 Bytes  0.00 bits/sec    0   1.41 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec  97.6 KBytes  79.9 Kbits/sec    5             sender
[  4]   0.00-10.00  sec  0.00 Bytes  0.00 bits/sec                  receiver

iperf Done.
mininet> 
```



Activities
Visual Study Code
五 14:32
vm2.py - Project - Visual Studio Code
Ubuntu Software
File Edit Selection View Go Run Terminal Help
vm2.py
Lab 4 > vm2.py > main
sudo - Lab 4
Ln 22, Col 27 Spaces: 4 UTF-8 LF Python 3.6.9 64-bit

```python
import os
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.link import TCLink
from mininet.cli import CLI
from mininet.log import setLogLevel


class SingleSwitchTopo(Topo):
    def build(self):
        h2 = self.addHost('h2', ip='10.0.0.2')
        s2 = self.addSwitch('s2', ip='10.0.0.4')
        self.addLink(s2, h2, bw=10, loss=0, delay='5ms')


def main():
    os.system('sudo ifconfig enp0s8 192.168.56.102 netmask 255.255.255.0')
    setLogLevel('info')
    topology = SingleSwitchTopo()
    network = Mininet(topology, link=TCLink)
    network.start()
    h2 = network.get('h2')
    h2.cmd('ifconfig h2-eth0 10.0.0.2 netmask 255.0.0.0')
    os.system('sudo ifconfig s2 10.0.0.4/8 up')
    os.system('sudo ovs-vsctl add-br br2')
    os.system('sudo ovs-vsctl add-port br2 enp0s8')
    os.system('sudo ifconfig br2 192.168.56.102/24 up')
    os.system('sudo route add default gw 192.168.56.2')
    os.system('sudo ovs-vsctl add-port s2 vx2 -- set interface vx2 type=vxlan options:remote_ip=192.168.56.101')
    CLI(network)
    network.stop()


if __name__ == '__main__':
    main()
```

```
xxyqwq@Ubuntu:~/Project/Lab 4$ sudo python3 vm2.py
*** Creating network
*** Adding controller
*** Adding hosts:
h2
*** Adding switches:
s2
*** Adding links:
(10.00Mbit 5ms delay 0.00000% loss) (10.00Mbit 5ms delay 0.00000% loss) (s2, h2)
*** Configuring hosts
h2
*** Starting controller
c0
*** Starting 1 switches
s2 ...(10.00Mbit 5ms delay 0.00000% loss)
*** Starting CLI:
mininet> s2 iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 10.0.0.3, port 47486
[  5] local 10.0.0.4 port 5201 connected to 10.0.0.3 port 47500
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-1.00   sec  0.00 Bytes  0.00 bits/sec
[  5]   1.00-2.00   sec  0.00 Bytes  0.00 bits/sec
[  5]   2.00-3.00   sec  0.00 Bytes  0.00 bits/sec
[  5]   3.00-4.00   sec  0.00 Bytes  0.00 bits/sec
[  5]   4.00-5.00   sec  0.00 Bytes  0.00 bits/sec
[  5]   5.00-6.01   sec  0.00 Bytes  0.00 bits/sec
[  5]   6.01-7.00   sec  0.00 Bytes  0.00 bits/sec
[  5]   7.00-8.01   sec  0.00 Bytes  0.00 bits/sec
[  5]   8.01-9.00   sec  0.00 Bytes  0.00 bits/sec
[  5]   9.00-10.00  sec  0.00 Bytes  0.00 bits/sec
[  5]  10.00-10.00  sec  0.00 Bytes  0.00 bits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-10.00  sec  0.00 Bytes  0.00 bits/sec                  sender
[  5]   0.00-10.00  sec  0.00 Bytes  0.00 bits/sec                  receiver
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

We can see that the bandwidth between 192.168.56.101 and 192.168.56.102 is approximately 798Kbits/sec at the beginning, which is extremely low. Later the bandwidth drops to zero and all the packets are lost. This problem arises from the unreasonable Maximum Transmission Unit (MTU) setting. Run the following command on VM 1.

```
ifconfig br1
```

We can see that the MTU size of the bridge is 1500Bytes. However, the packet is encapsulated in the VXLAN header and the UDP header, which enlarges the packet size. Since the packet size exceeds the MTU limitation, it will be split into several packets. However, the VTEP does not support the packet fragmentation, so the bandwidth drops shaply and other packets are lost.[5]

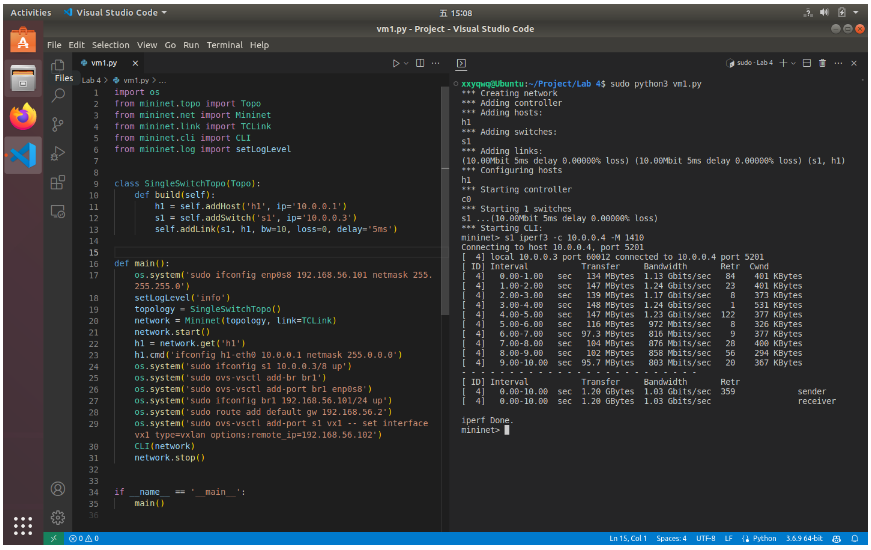
4. (1) Now we try to determine the reasonable MTU size. The MTU size of the bridge is 1500Bytes. The VXLAN header requires 50Bytes. The UDP header requires 20Bytes, in which 12Bytes are used for checksum. Another 20Bytes are required for the IPv4 header. Therefore, the reasonable MTU size is specified as

$$1500\text{Bytes} - 50\text{Bytes} - 20\text{Bytes} - 20\text{Bytes} = 1410\text{Bytes}$$

With the reasonable MTU size, we run the following command on s1.

```
iperf3 -c 10.0.0.4 -M 1410
```

The results are shown in the following figure.



Activities
Visual Studio Code
五 15:08
vm1.py - Project - Visual Studio Code
File Edit Selection View Go Run Terminal Help
vm1.py
Files
Lab 4 > vm1.py > ...
sudo - Lab 4
Ln 15, Col 1 Spaces: 4 UTF-8 LF Python 3.6.9 64-bit

```python
import os
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.link import TCLink
from mininet.cli import CLI
from mininet.log import setLogLevel


class SingleSwitchTopo(Topo):
    def build(self):
        h1 = self.addHost('h1', ip='10.0.0.1')
        s1 = self.addSwitch('s1', ip='10.0.0.3')
        self.addLink(s1, h1, bw=10, loss=0, delay='5ms')


def main():
    os.system('sudo ifconfig enp0s8 192.168.56.101 netmask 255.
    255.255.0')
    setLogLevel('info')
    topology = SingleSwitchTopo()
    network = Mininet(topology, link=TCLink)
    network.start()
    h1 = network.get('h1')
    h1.cmd('ifconfig h1-eth0 10.0.0.1 netmask 255.0.0.0')
    os.system('sudo ifconfig s1 10.0.0.3/8 up')
    os.system('sudo ovs-vsctl add-br br1')
    os.system('sudo ovs-vsctl add-port br1 enp0s8')
    os.system('sudo ifconfig br1 192.168.56.101/24 up')
    os.system('sudo route add default gw 192.168.56.2')
    os.system('sudo ovs-vsctl add-port s1 vx1 -- set interface
    vx1 type=vxlan options:remote_ip=192.168.56.102')
    CLI(network)
    network.stop()


if __name__ == '__main__':
    main()
```

```
xxyqwq@Ubuntu:~/Project/Lab 4$ sudo python3 vm1.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1
*** Adding switches:
s1
*** Adding links:
(10.00Mbit 5ms delay 0.00000% loss) (10.00Mbit 5ms delay 0.00000% loss) (s1, h1)
*** Configuring hosts
h1
*** Starting controller
c0
*** Starting 1 switches
s1 ...(10.00Mbit 5ms delay 0.00000% loss)
*** Starting CLI:
mininet> s1 iperf3 -c 10.0.0.4 -M 1410
Connecting to host 10.0.0.4, port 5201
[  4] local 10.0.0.3 port 60012 connected to 10.0.0.4 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec   134 MBytes  1.13 Gbits/sec   84    401 KBytes
[  4]   1.00-2.00   sec   147 MBytes  1.24 Gbits/sec   23    401 KBytes
[  4]   2.00-3.00   sec   139 MBytes  1.17 Gbits/sec    8    373 KBytes
[  4]   3.00-4.00   sec   148 MBytes  1.24 Gbits/sec    1    531 KBytes
[  4]   4.00-5.00   sec   147 MBytes  1.23 Gbits/sec  122    377 KBytes
[  4]   5.00-6.00   sec   116 MBytes   972 Mbits/sec    8    326 KBytes
[  4]   6.00-7.00   sec  97.3 MBytes   816 Mbits/sec    9    377 KBytes
[  4]   7.00-8.00   sec   104 MBytes   876 Mbits/sec   28    400 KBytes
[  4]   8.00-9.00   sec   102 MBytes   858 Mbits/sec   56    294 KBytes
[  4]   9.00-10.00  sec  95.7 MBytes   803 Mbits/sec   20    367 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec  1.20 GBytes  1.03 Gbits/sec  359             sender
[  4]   0.00-10.00  sec  1.20 GBytes  1.03 Gbits/sec                  receiver

iperf Done.
mininet> 
```

We can see that the bandwidth between 192.168.56.101 and 192.168.56.102 becomes 1.03Gbits/sec, which is a great improvement compared with the previous results.

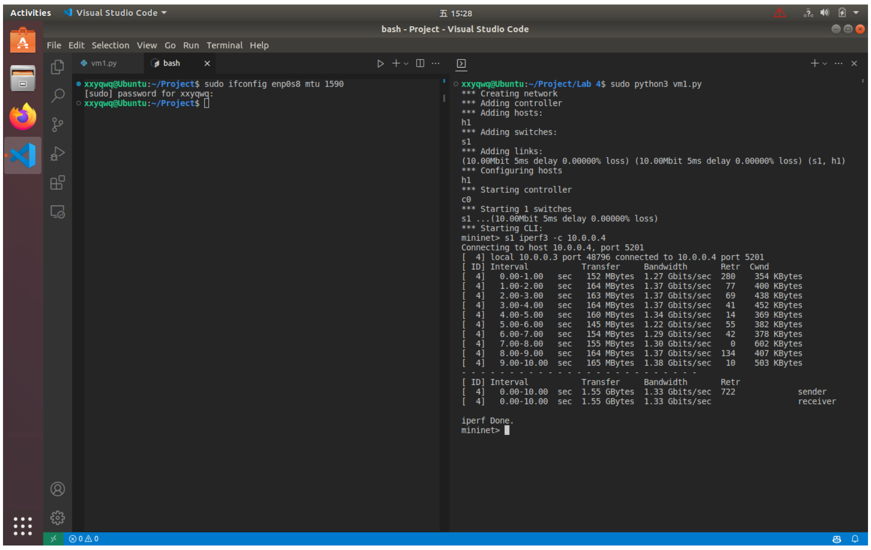
(2) Similarly, for the two enp0s8 interfaces, the reasonable MTU size is specified as

$$1500\text{Bytes} + 50\text{Bytes} + 20\text{Bytes} + 20\text{Bytes} = 1590\text{Bytes}$$

To modify the MTU size, we run the following command on both VM 1 and VM 2.

```
sudo ifconfig enp0s8 mtu 1590
```

Then we test the bandwidth again. The results are shown in the following figure.



Activities
Visual Studio Code
五 15:28
bash - Project - Visual Studio Code
File Edit Selection View Go Run Terminal Help
vm1.py
bash

```
xxyqwq@Ubuntu:~/Project$ sudo ifconfig enp0s8 mtu 1590
[sudo] password for xxyqwq:
xxyqwq@Ubuntu:~/Project$ 
```

```
xxyqwq@Ubuntu:~/Project/Lab 4$ sudo python3 vm1.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1
*** Adding switches:
s1
*** Adding links:
(10.00Mbit 5ms delay 0.00000% loss) (10.00Mbit 5ms delay 0.00000% loss) (s1, h1)
*** Configuring hosts
h1
*** Starting controller
c0
*** Starting 1 switches
s1 ...(10.00Mbit 5ms delay 0.00000% loss)
*** Starting CLI:
mininet> s1 iperf3 -c 10.0.0.4
Connecting to host 10.0.0.4, port 5201
[  4] local 10.0.0.3 port 48796 connected to 10.0.0.4 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec   152 MBytes  1.27 Gbits/sec  280    354 KBytes
[  4]   1.00-2.00   sec   164 MBytes  1.37 Gbits/sec   77    400 KBytes
[  4]   2.00-3.00   sec   163 MBytes  1.37 Gbits/sec   69    438 KBytes
[  4]   3.00-4.00   sec   164 MBytes  1.37 Gbits/sec   41    452 KBytes
[  4]   4.00-5.00   sec   160 MBytes  1.34 Gbits/sec   14    369 KBytes
[  4]   5.00-6.00   sec   145 MBytes  1.22 Gbits/sec   55    382 KBytes
[  4]   6.00-7.00   sec   154 MBytes  1.29 Gbits/sec   42    378 KBytes
[  4]   7.00-8.00   sec   155 MBytes  1.30 Gbits/sec    0    602 KBytes
[  4]   8.00-9.00   sec   164 MBytes  1.37 Gbits/sec  134    407 KBytes
[  4]   9.00-10.00  sec   165 MBytes  1.38 Gbits/sec   10    503 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec  1.55 GBytes  1.33 Gbits/sec  722             sender
[  4]   0.00-10.00  sec  1.55 GBytes  1.33 Gbits/sec                  receiver

iperf Done.
mininet> 
```

We can see that the bandwidth between 192.168.56.101 and 192.168.56.102 becomes 1.33Gbits/sec, which is a also great improvement.

References

[1] Wikipedia. Address Resolution Protocol. https://en.wikipedia.org/wiki/Address_Resolution_Protocol.

[2] Wikipedia. Internet Control Message Protocol. https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol.

[3] Wikipedia. Internet Protocol version 4. https://en.wikipedia.org/wiki/Internet_Protocol_version_4.

[4] Wikipedia. User Datagram Protocol. https://en.wikipedia.org/wiki/User_Datagram_Protocol.

[5] Wikipedia. Maximum Transmission Unit. https://en.wikipedia.org/wiki/Maximum_transmission_unit.