Overlay Network and VXLAN

Xin Xu 519021910726

Preparation

The lab environment is ubuntu 20.04, which is used as vm1. And I create another virtual machine inside vm1, which is vm2.

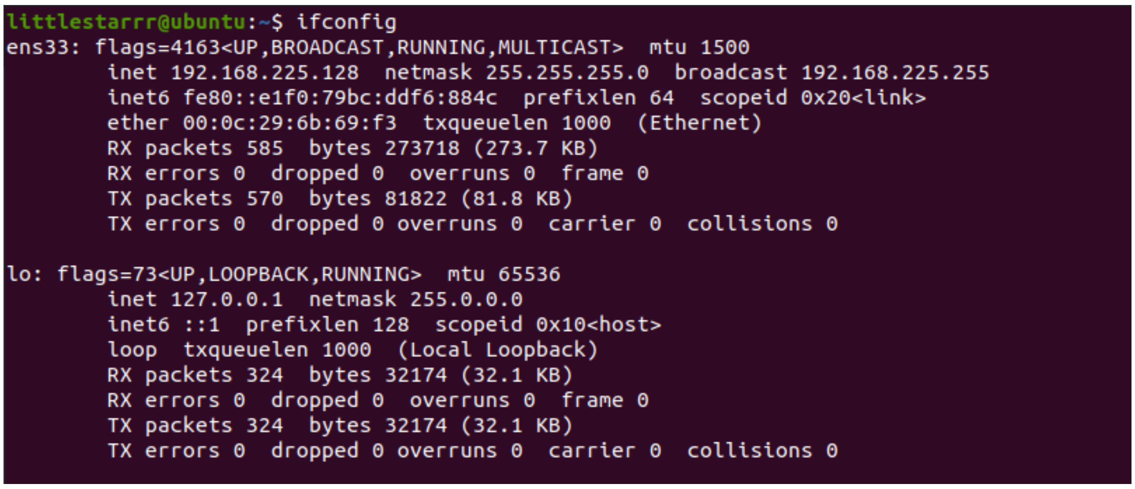
Use `ifconfig` command to check the network information of vm1 and vm2. Chances are that vm1 and vm2 are not in the same subnet, which means we can't `ping` vm2 from vm1. So, I use bridge and tap interface to enable `ping` between these two virtual machines.

The commands I used are as follows:

- Create the bridge:

  `sudo brctl addbr br0`

- Get the NIC of the host:

```
littlestarrr@ubuntu:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.225.128  netmask 255.255.255.0  broadcast 192.168.225.255
        inet6 fe80::e1f0:79bc:ddf6:884c  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:6b:69:f3  txqueuelen 1000  (Ethernet)
        RX packets 585  bytes 273718 (273.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 570  bytes 81822 (81.8 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 324  bytes 32174 (32.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 324  bytes 32174 (32.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

  So the NIC of the host machine is `ens33`.

- Clear IP of `ens33` (the NIC of vm1):

  `sudo ip add flush dev ens33`

- Add `ens33` to bridge:

  `sudo brctl addif br0 ens33`

- Create TAP interface(littlestarrr is the name of vm2):

  `sudo tunctl -t tap0 -u littlestarrr`

- Add `tap0` to bridge:

  `sudo brctl addif br0 tap0`

- Make sure that everything is up:

```
sudo ifconfig ens33 up
sudo ifconfig tap0 up
sudo ifconfig br0 up
```

- Give IP address to `br0`:

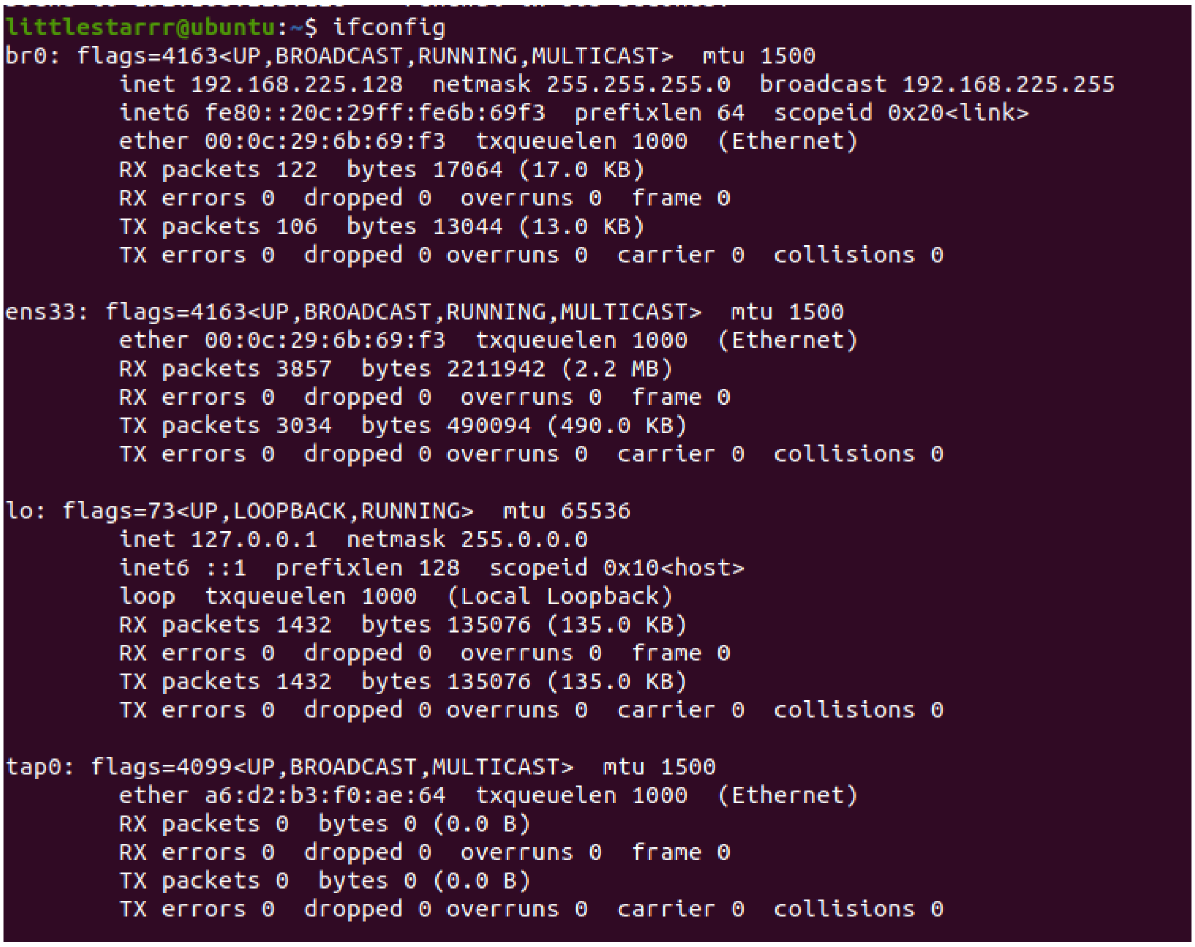
`sudo dhclient -v br0`

- Create vm2 with tap0:

  `qemu-system-x86_64 ubuntu.img -m 2048 -enable-kvm -netdev tap,id=mynet0,ifname=tap0,script=no,downscript=no -device e1000,netdev=mynet0,mac=52:55:00:d1:55:01`

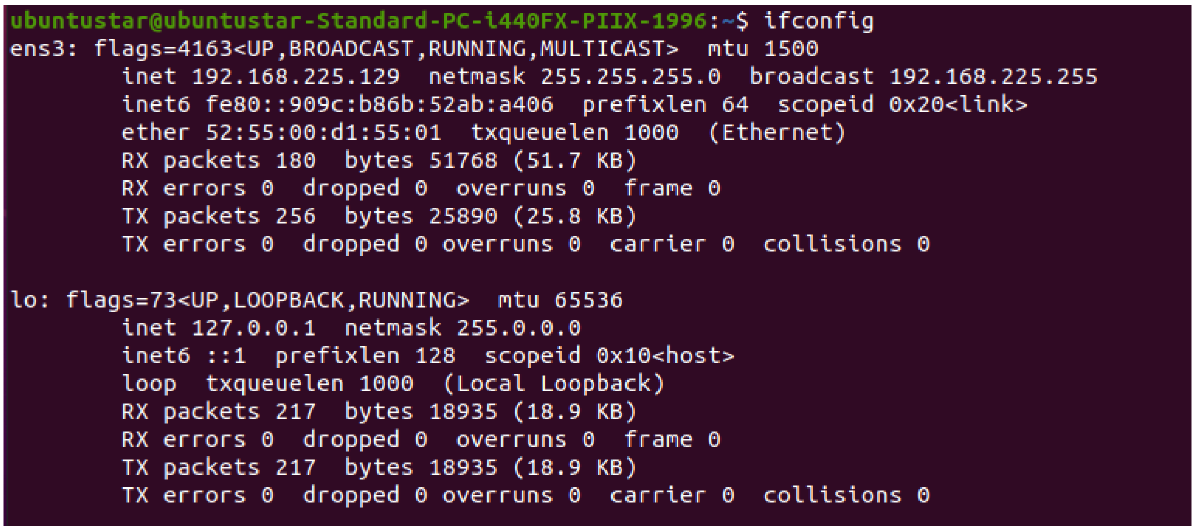
Now, vm1 and vm2 are in the same subnet:

vm1:



```
littlestarrr@ubuntu:~$ ifconfig
br0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.225.128  netmask 255.255.255.0  broadcast 192.168.225.255
        inet6 fe80::20c:29ff:fe6b:69f3  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:6b:69:f3  txqueuelen 1000  (Ethernet)
        RX packets 122  bytes 17064 (17.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 106  bytes 13044 (13.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether 00:0c:29:6b:69:f3  txqueuelen 1000  (Ethernet)
        RX packets 3857  bytes 2211942 (2.2 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3034  bytes 490094 (490.0 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 1432  bytes 135076 (135.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1432  bytes 135076 (135.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

tap0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether a6:d2:b3:f0:ae:64  txqueuelen 1000  (Ethernet)
        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
```

vm2:



```
ubuntustar@ubuntustar-Standard-PC-i440FX-PIIX-1996:~$ ifconfig
ens3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.225.129  netmask 255.255.255.0  broadcast 192.168.225.255
        inet6 fe80::909c:b86b:52ab:a406  prefixlen 64  scopeid 0x20<link>
        ether 52:55:00:d1:55:01  txqueuelen 1000  (Ethernet)
        RX packets 180  bytes 51768 (51.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 256  bytes 25890 (25.8 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 217  bytes 18935 (18.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 217  bytes 18935 (18.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

So that vm1 and vm2 are in the same subnet, which means vm1 can `ping` vm2 successfully.

Experiment with VXLAN

On vm1, start mininet and set IPs for both hosts:

```
sudo mn
h1 ifconfig h1-eth0 10.0.0.1 netmask 255.0.0.0
h2 ifconfig h2-eth0 10.0.0.2 netmask 255.0.0.0
```

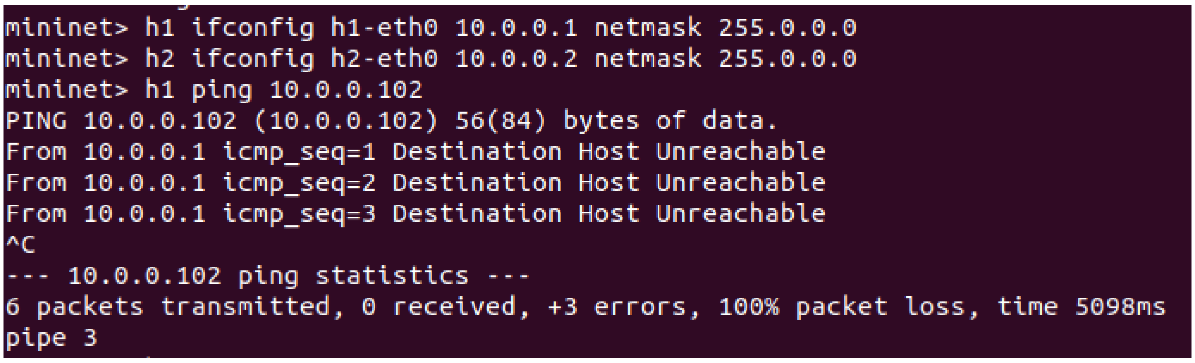
Open another terminal, and set the switch IP as:

`sudo ifconfig s1 10.0.0.101/8 up`

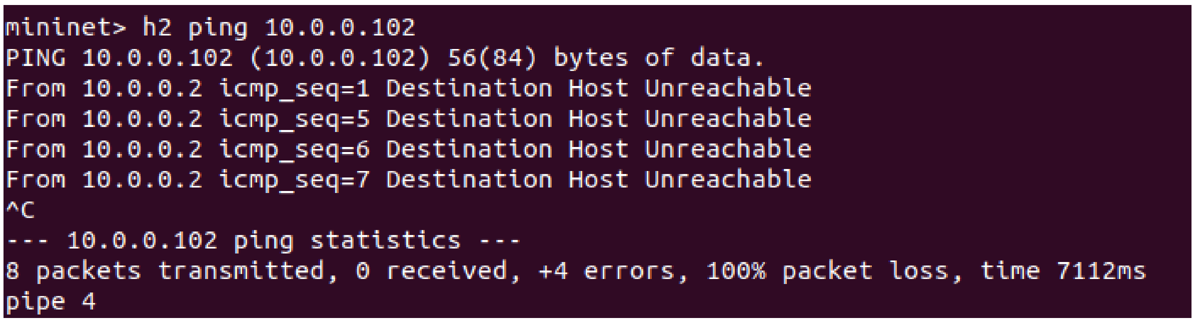
On vm2, create a new bridge and set its IP as follows:

```
sudo ovs-vsctl add-br s2
sudo ifconfig s2 10.0.0.102/8 up
```

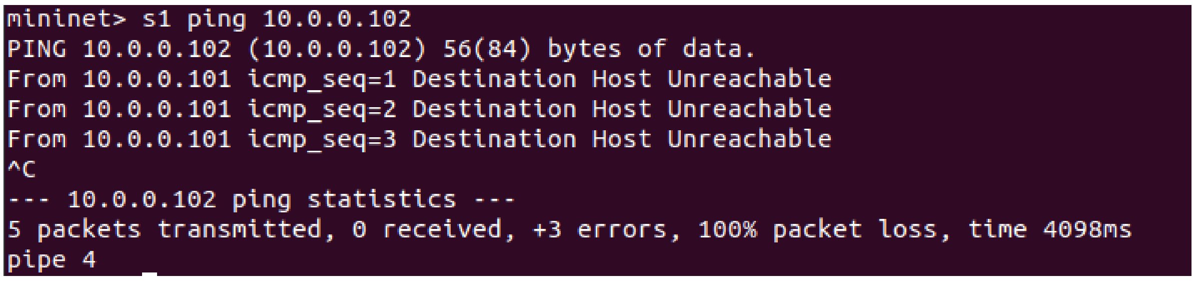
Now, if you ping 10.0.0.102 from 10.0.0.101/10.0.0.1/10.0.0.2, there will be no response:



```
mininet> h1 ifconfig h1-eth0 10.0.0.1 netmask 255.0.0.0
mininet> h2 ifconfig h2-eth0 10.0.0.2 netmask 255.0.0.0
mininet> h1 ping 10.0.0.102
PING 10.0.0.102 (10.0.0.102) 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
^C
--- 10.0.0.102 ping statistics ---
6 packets transmitted, 0 received, +3 errors, 100% packet loss, time 5098ms
pipe 3
```



```
mininet> h2 ping 10.0.0.102
PING 10.0.0.102 (10.0.0.102) 56(84) bytes of data.
From 10.0.0.2 icmp_seq=1 Destination Host Unreachable
From 10.0.0.2 icmp_seq=5 Destination Host Unreachable
From 10.0.0.2 icmp_seq=6 Destination Host Unreachable
From 10.0.0.2 icmp_seq=7 Destination Host Unreachable
^C
--- 10.0.0.102 ping statistics ---
8 packets transmitted, 0 received, +4 errors, 100% packet loss, time 7112ms
pipe 4
```



```
mininet> s1 ping 10.0.0.102
PING 10.0.0.102 (10.0.0.102) 56(84) bytes of data.
From 10.0.0.101 icmp_seq=1 Destination Host Unreachable
From 10.0.0.101 icmp_seq=2 Destination Host Unreachable
From 10.0.0.101 icmp_seq=3 Destination Host Unreachable
^C
--- 10.0.0.102 ping statistics ---
5 packets transmitted, 0 received, +3 errors, 100% packet loss, time 4098ms
pipe 4
```

To tackle this problem, we can create a vxlan port in the bridges of s1 and s2:

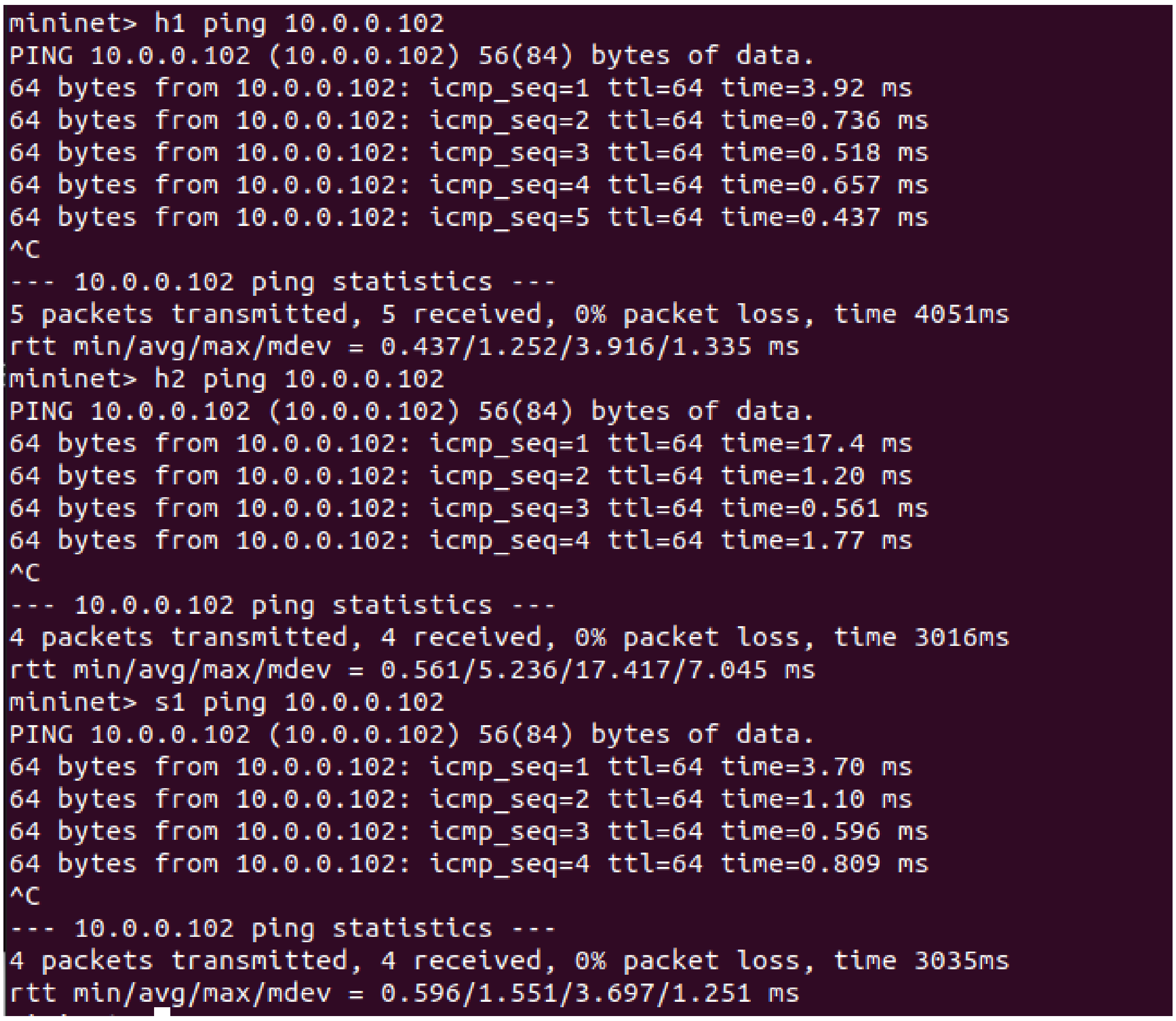
On vm1:

`sudo ovs-vsctl add-port s1 vxlan0 -- set interface vxlan0 type=vxlan options:remote_ip=192.168.225.129`

On vm2:

`sudo ovs-vsctl add-port s1 vxlan0 -- set interface vxlan0 type=vxlan options:remote_ip=192.168.225.128`

Now, we can ping 10.0.0.102 from 10.0.0.101/10.0.0.1/10.0.0.2.

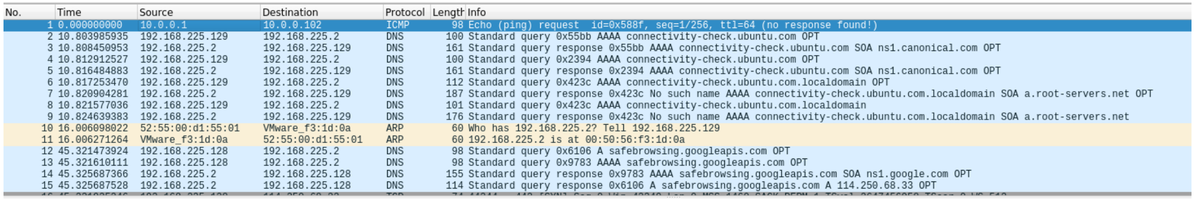


```
mininet> h1 ping 10.0.0.102
PING 10.0.0.102 (10.0.0.102) 56(84) bytes of data.
64 bytes from 10.0.0.102: icmp_seq=1 ttl=64 time=3.92 ms
64 bytes from 10.0.0.102: icmp_seq=2 ttl=64 time=0.736 ms
64 bytes from 10.0.0.102: icmp_seq=3 ttl=64 time=0.518 ms
64 bytes from 10.0.0.102: icmp_seq=4 ttl=64 time=0.657 ms
64 bytes from 10.0.0.102: icmp_seq=5 ttl=64 time=0.437 ms
^C
--- 10.0.0.102 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4051ms
rtt min/avg/max/mdev = 0.437/1.252/3.916/1.335 ms
mininet> h2 ping 10.0.0.102
PING 10.0.0.102 (10.0.0.102) 56(84) bytes of data.
64 bytes from 10.0.0.102: icmp_seq=1 ttl=64 time=17.4 ms
64 bytes from 10.0.0.102: icmp_seq=2 ttl=64 time=1.20 ms
64 bytes from 10.0.0.102: icmp_seq=3 ttl=64 time=0.561 ms
64 bytes from 10.0.0.102: icmp_seq=4 ttl=64 time=1.77 ms
^C
--- 10.0.0.102 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3016ms
rtt min/avg/max/mdev = 0.561/5.236/17.417/7.045 ms
mininet> s1 ping 10.0.0.102
PING 10.0.0.102 (10.0.0.102) 56(84) bytes of data.
64 bytes from 10.0.0.102: icmp_seq=1 ttl=64 time=3.70 ms
64 bytes from 10.0.0.102: icmp_seq=2 ttl=64 time=1.10 ms
64 bytes from 10.0.0.102: icmp_seq=3 ttl=64 time=0.596 ms
64 bytes from 10.0.0.102: icmp_seq=4 ttl=64 time=0.809 ms
^C
--- 10.0.0.102 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3035ms
rtt min/avg/max/mdev = 0.596/1.551/3.697/1.251 ms
```

Homework

Use Wireshark to monitor the interfaces

The result is:



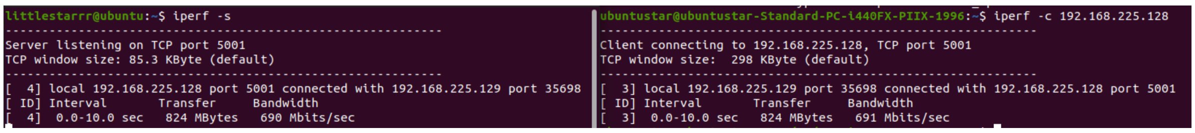
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.1	10.0.0.102	ICMP	98	Echo (ping) request id=0x588f, seq=1/256, ttl=64 (no response found!)
2	10.803985935	192.168.225.129	192.168.225.2	DNS	100	Standard query 0x55bb AAAA connectivity-check.ubuntu.com OPT
3	10.808450953	192.168.225.2	192.168.225.129	DNS	161	Standard query response 0x55bb AAAA connectivity-check.ubuntu.com SOA ns1.canonical.com OPT
4	10.812912527	192.168.225.129	192.168.225.2	DNS	100	Standard query 0x2394 AAAA connectivity-check.ubuntu.com OPT
5	10.816484883	192.168.225.2	192.168.225.129	DNS	161	Standard query response 0x2394 AAAA connectivity-check.ubuntu.com SOA ns1.canonical.com OPT
6	10.817253470	192.168.225.129	192.168.225.2	DNS	112	Standard query 0x423c AAAA connectivity-check.ubuntu.com.localdomain OPT
7	10.820904281	192.168.225.2	192.168.225.129	DNS	187	Standard query response 0x423c No such name AAAA connectivity-check.ubuntu.com.localdomain SOA a.root-servers.net OPT
8	10.821577036	192.168.225.129	192.168.225.2	DNS	101	Standard query 0x423c AAAA connectivity-check.ubuntu.com.localdomain
9	10.824639383	192.168.225.2	192.168.225.129	DNS	176	Standard query response 0x423c No such name AAAA connectivity-check.ubuntu.com.localdomain SOA a.root-servers.net
10	16.006098022	52:55:00:d1:55:01	VMware_f3:1d:0a	ARP	60	Who has 192.168.225.2? Tell 192.168.225.129
11	16.006271264	VMware_f3:1d:0a	52:55:00:d1:55:01	ARP	60	192.168.225.2 is at 00:50:56:f3:1d:0a
12	45.321473924	192.168.225.128	192.168.225.2	DNS	98	Standard query 0x6106 A safebrowsing.googleapis.com OPT
13	45.321610111	192.168.225.128	192.168.225.2	DNS	98	Standard query 0x9783 AAAA safebrowsing.googleapis.com OPT
14	45.325687366	192.168.225.2	192.168.225.128	DNS	155	Standard query response 0x9783 AAAA safebrowsing.googleapis.com SOA ns1.google.com OPT
15	45.325687528	192.168.225.2	192.168.225.128	DNS	114	Standard query response 0x6106 A safebrowsing.googleapis.com A 114.250.68.33 OPT

Just the first packet is sent from interface s1, all the rest is sent from `ens33` (the true NIC IP address of vm1). Because s2 is not in the same subnet of h1, h2 and s1, it cannot ping them naturally. The packets only can be sent through `vxlan0`, and the `vxlan0` from s2 connects to the IP address of `ens33`, so the rest packets are sent from `ens33`.

Use `iperf` to test bandwidth

Test the bandwidth between two vms

The result is:

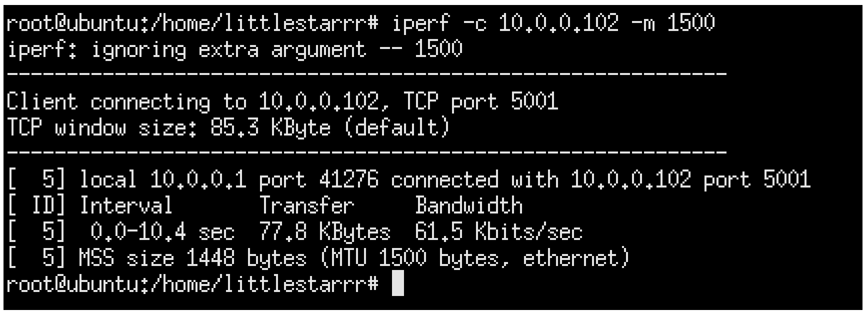


```
littlestarrr@ubuntu:~$ iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 192.168.225.128 port 5001 connected with 192.168.225.129 port 35698
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec   824 MBytes   690 Mbits/sec
```

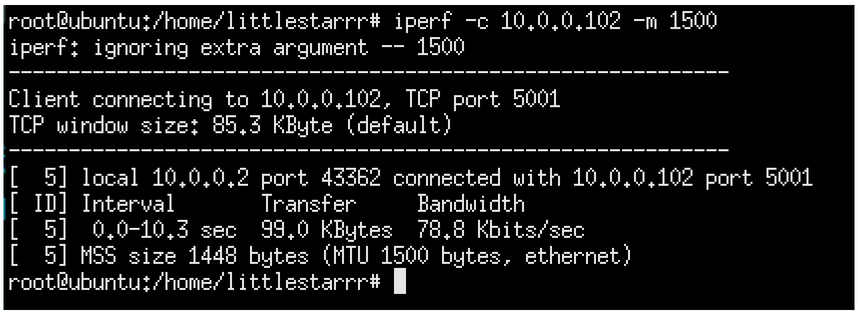
```
ubuntustar@ubuntustar-Standard-PC-i440FX-PIIX-1996:~$ iperf -c 192.168.225.128
------------------------------------------------------------
Client connecting to 192.168.225.128, TCP port 5001
TCP window size:  298 KByte (default)
------------------------------------------------------------
[  3] local 192.168.225.129 port 35698 connected with 192.168.225.128 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec   824 MBytes   691 Mbits/sec
```

Test the bandwidth between s2 and h1, h2 and s1

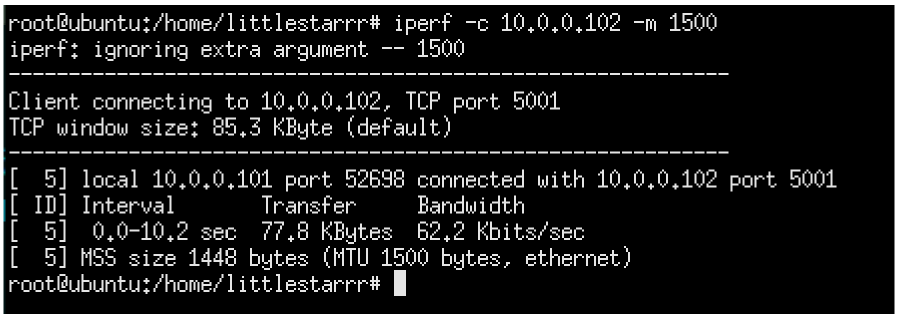
The result is:



```
root@ubuntu:/home/littlestarrr# iperf -c 10.0.0.102 -m 1500
iperf: ignoring extra argument -- 1500
------------------------------------------------------------
Client connecting to 10.0.0.102, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  5] local 10.0.0.1 port 41276 connected with 10.0.0.102 port 5001
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.4 sec  77.8 KBytes  61.5 Kbits/sec
[  5] MSS size 1448 bytes (MTU 1500 bytes, ethernet)
root@ubuntu:/home/littlestarrr#
```



```
root@ubuntu:/home/littlestarrr# iperf -c 10.0.0.102 -m 1500
iperf: ignoring extra argument -- 1500
------------------------------------------------------------
Client connecting to 10.0.0.102, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  5] local 10.0.0.2 port 43362 connected with 10.0.0.102 port 5001
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.3 sec  99.0 KBytes  78.8 Kbits/sec
[  5] MSS size 1448 bytes (MTU 1500 bytes, ethernet)
root@ubuntu:/home/littlestarrr#
```



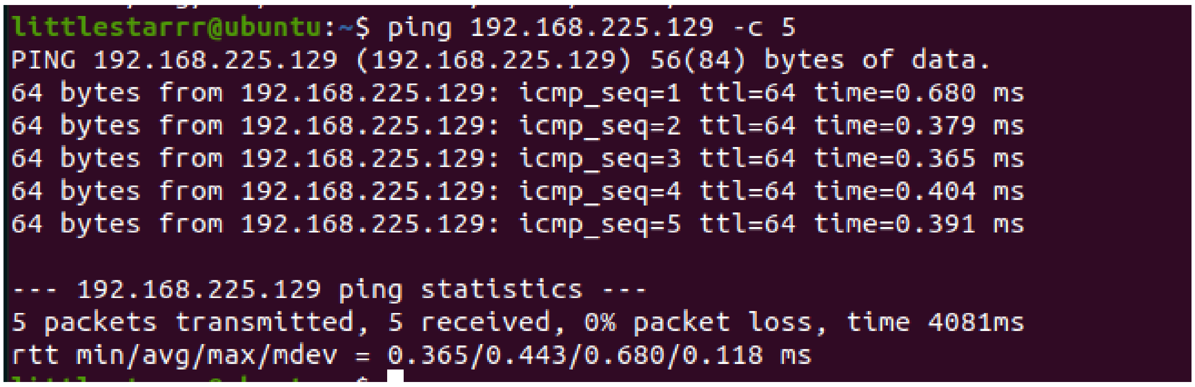
```
root@ubuntu:/home/littlestarrr# iperf -c 10.0.0.102 -m 1500
iperf: ignoring extra argument -- 1500
------------------------------------------------------------
Client connecting to 10.0.0.102, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  5] local 10.0.0.101 port 52698 connected with 10.0.0.102 port 5001
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.2 sec  77.8 KBytes  62.2 Kbits/sec
[  5] MSS size 1448 bytes (MTU 1500 bytes, ethernet)
root@ubuntu:/home/littlestarrr#
```

Because the maximum MTU of h1, h2, s1, s2 and `vxlan0` is 1500, so I fix the maximum size of packets in iperf. If there is no consistency in the size of MTU, switches may need more time to unpackaged, split and package packets. So that the bandwidth may be reduced for the waste of time. We can see that the bandwidth between s2 and h1, h2 and s1 is much less than the bandwidth between two vms. The reason is that packets transmitted by vxlan must be encapsulated, which needs more time to transmit one packet than normal transmition. And because vxlan will add a head to the original packet, it may exceeds the default 1514 byte MTU, which can result in less bandwidth.

Use `ping` to test the network latency

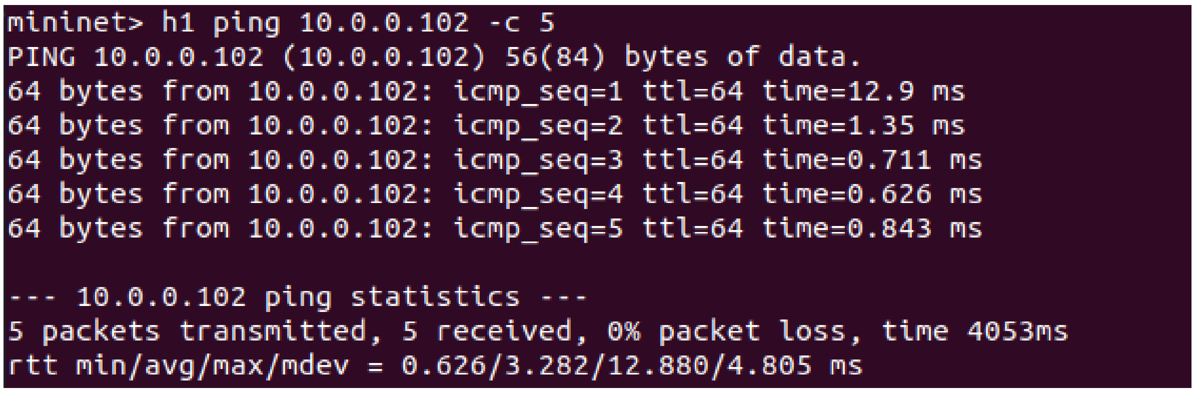
Test the latency between two vms

The result is:

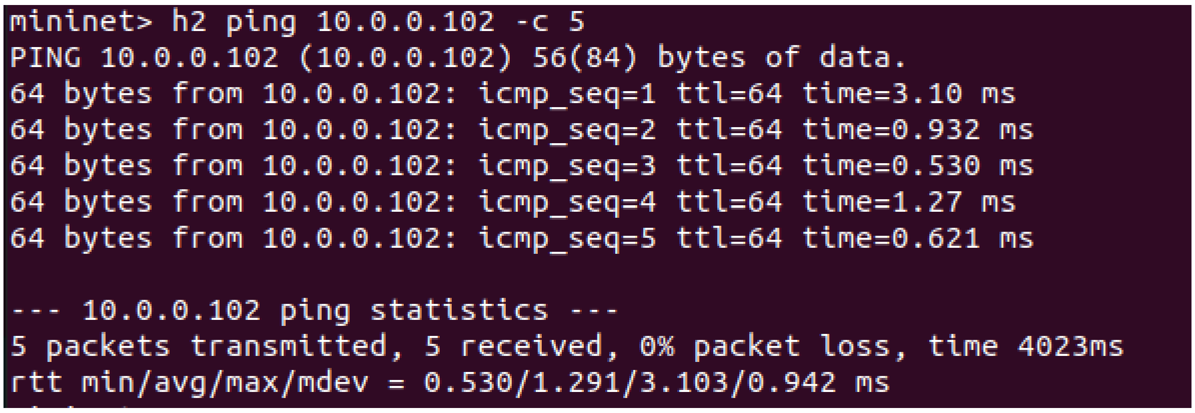


```
littlestarrr@ubuntu:~$ ping 192.168.225.129 -c 5
PING 192.168.225.129 (192.168.225.129) 56(84) bytes of data.
64 bytes from 192.168.225.129: icmp_seq=1 ttl=64 time=0.680 ms
64 bytes from 192.168.225.129: icmp_seq=2 ttl=64 time=0.379 ms
64 bytes from 192.168.225.129: icmp_seq=3 ttl=64 time=0.365 ms
64 bytes from 192.168.225.129: icmp_seq=4 ttl=64 time=0.404 ms
64 bytes from 192.168.225.129: icmp_seq=5 ttl=64 time=0.391 ms

--- 192.168.225.129 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4081ms
rtt min/avg/max/mdev = 0.365/0.443/0.680/0.118 ms
```

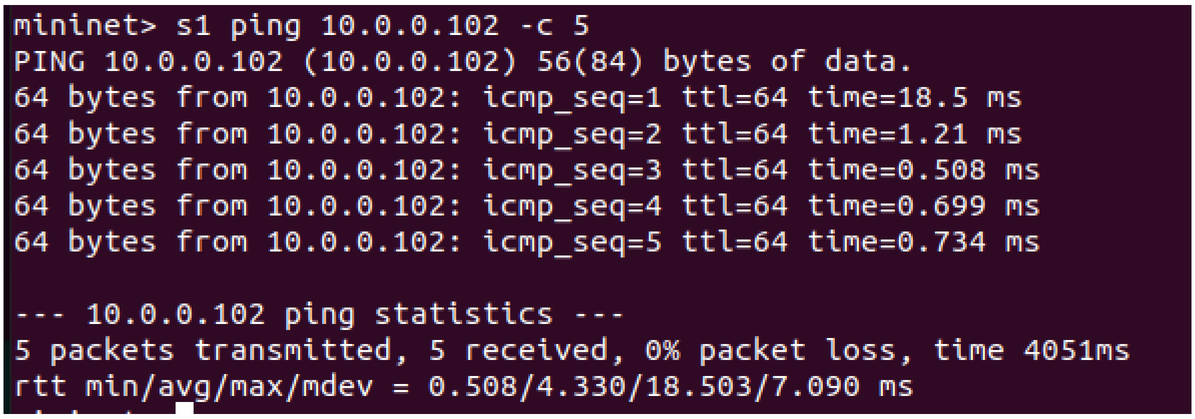
Test the latency between s2 and h1, h2 and s1



```
mininet> h1 ping 10.0.0.102 -c 5
PING 10.0.0.102 (10.0.0.102) 56(84) bytes of data.
64 bytes from 10.0.0.102: icmp_seq=1 ttl=64 time=12.9 ms
64 bytes from 10.0.0.102: icmp_seq=2 ttl=64 time=1.35 ms
64 bytes from 10.0.0.102: icmp_seq=3 ttl=64 time=0.711 ms
64 bytes from 10.0.0.102: icmp_seq=4 ttl=64 time=0.626 ms
64 bytes from 10.0.0.102: icmp_seq=5 ttl=64 time=0.843 ms

--- 10.0.0.102 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4053ms
rtt min/avg/max/mdev = 0.626/3.282/12.880/4.805 ms
```



```
mininet> h2 ping 10.0.0.102 -c 5
PING 10.0.0.102 (10.0.0.102) 56(84) bytes of data.
64 bytes from 10.0.0.102: icmp_seq=1 ttl=64 time=3.10 ms
64 bytes from 10.0.0.102: icmp_seq=2 ttl=64 time=0.932 ms
64 bytes from 10.0.0.102: icmp_seq=3 ttl=64 time=0.530 ms
64 bytes from 10.0.0.102: icmp_seq=4 ttl=64 time=1.27 ms
64 bytes from 10.0.0.102: icmp_seq=5 ttl=64 time=0.621 ms

--- 10.0.0.102 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4023ms
rtt min/avg/max/mdev = 0.530/1.291/3.103/0.942 ms
```



```
mininet> s1 ping 10.0.0.102 -c 5
PING 10.0.0.102 (10.0.0.102) 56(84) bytes of data.
64 bytes from 10.0.0.102: icmp_seq=1 ttl=64 time=18.5 ms
64 bytes from 10.0.0.102: icmp_seq=2 ttl=64 time=1.21 ms
64 bytes from 10.0.0.102: icmp_seq=3 ttl=64 time=0.508 ms
64 bytes from 10.0.0.102: icmp_seq=4 ttl=64 time=0.699 ms
64 bytes from 10.0.0.102: icmp_seq=5 ttl=64 time=0.734 ms

--- 10.0.0.102 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4051ms
rtt min/avg/max/mdev = 0.508/4.330/18.503/7.090 ms
```

As we can see, the time to transmit the first packet is much longer than others in vxlan. The reason is ARP request. At first h1 doesn't know the mac address of s2, corresponding VTEP must broadcast ARP packet. After s2 responds and VTEPs record the path between h1 and s2, h1 can send ICMP packet. After the first packet of ping process, the rest only needs to transmit ICMP packets. So the time is much shorter.

The time to directly `ping` two vms is much shorter than `ping` through vxlan. Because the two virtual machines are in the same subnet, there is an ARP cache in the first ping packet, which can avoid ARP request. And it doesn't need to encapsulate vxlan header, too.