

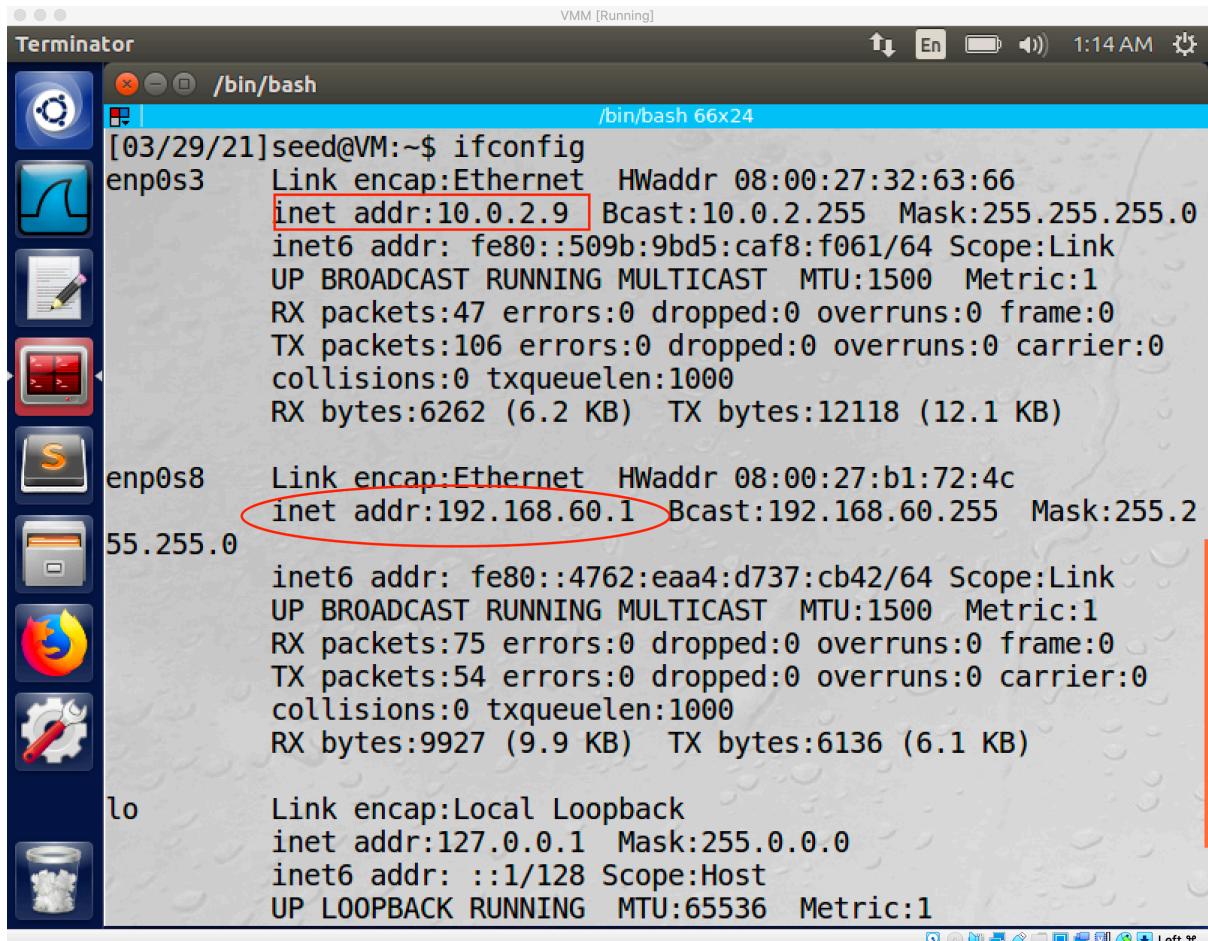
Client VM: IP: 10.0.2.7

Server VM: IP1: 10.0.2.9, IP2: 192.168.60.1

Host v: IP: 192.168.60.101

Task1: Network Setup

As for Server VM, two network adaptors are added, one has the IP address of 10.0.2.9, the other has the IP address of 192.168.60.1. It is because the server needs to be accessible by both client and the host v.



```
[03/29/21]seed@VM:~$ ifconfig
enp0s3      Link encap:Ethernet HWaddr 08:00:27:32:63:66
             inet addr:10.0.2.9 Bcast:10.0.2.255 Mask:255.255.255.0
             inet6 addr: fe80::509b:9bd5:caf8:f061/64 Scope:Link
                   UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
                   RX packets:47 errors:0 dropped:0 overruns:0 frame:0
                   TX packets:106 errors:0 dropped:0 overruns:0 carrier:0
                   collisions:0 txqueuelen:1000
                   RX bytes:6262 (6.2 KB) TX bytes:12118 (12.1 KB)

enp0s8      Link encap:Ethernet HWaddr 08:00:27:b1:72:4c
             inet addr:192.168.60.1 Bcast:192.168.60.255 Mask:255.255.255.0
             inet6 addr: fe80::4762:eaa4:d737:cb42/64 Scope:Link
                   UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
                   RX packets:75 errors:0 dropped:0 overruns:0 frame:0
                   TX packets:54 errors:0 dropped:0 overruns:0 carrier:0
                   collisions:0 txqueuelen:1000
                   RX bytes:9927 (9.9 KB) TX bytes:6136 (6.1 KB)

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
```

As for the host v, I create a new connection for it which set up the ipv4 address as 192.168.60.101 to make sure it within the same private network with the server, which means the client cannot access it.

```
bin/bash /bin/bash 70x24
[03/30/21]seed@VM:~$ ifconfig
enp0s3      Link encap:Ethernet HWaddr 08:00:27:66:b0:79
              inet addr:192.168.60.101 Bcast:192.168.60.255 Mask:255.255
              .255.0
                      inet6 addr: fe80::a80e:6b3f:e26b:f414/64 Scope:Link
                      UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
                      RX packets:0 errors:0 dropped:0 overruns:0 frame:0
                      TX packets:70 errors:0 dropped:0 overruns:0 carrier:0
                      collisions:0 txqueuelen:1000
                      RX bytes:0 (0.0 B) TX bytes:6995 (6.9 KB)

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:36 errors:0 dropped:0 overruns:0 frame:0
                      TX packets:36 errors:0 dropped:0 overruns:0 carrier:0
                      collisions:0 txqueuelen:1
                      RX bytes:10834 (10.8 KB) TX bytes:10834 (10.8 KB)

[03/30/21]seed@VM:~$
```

To test this setup, several tests have been done.

1. I ping the VPN server from the Host U.

VMB [Running] 3:34 AM

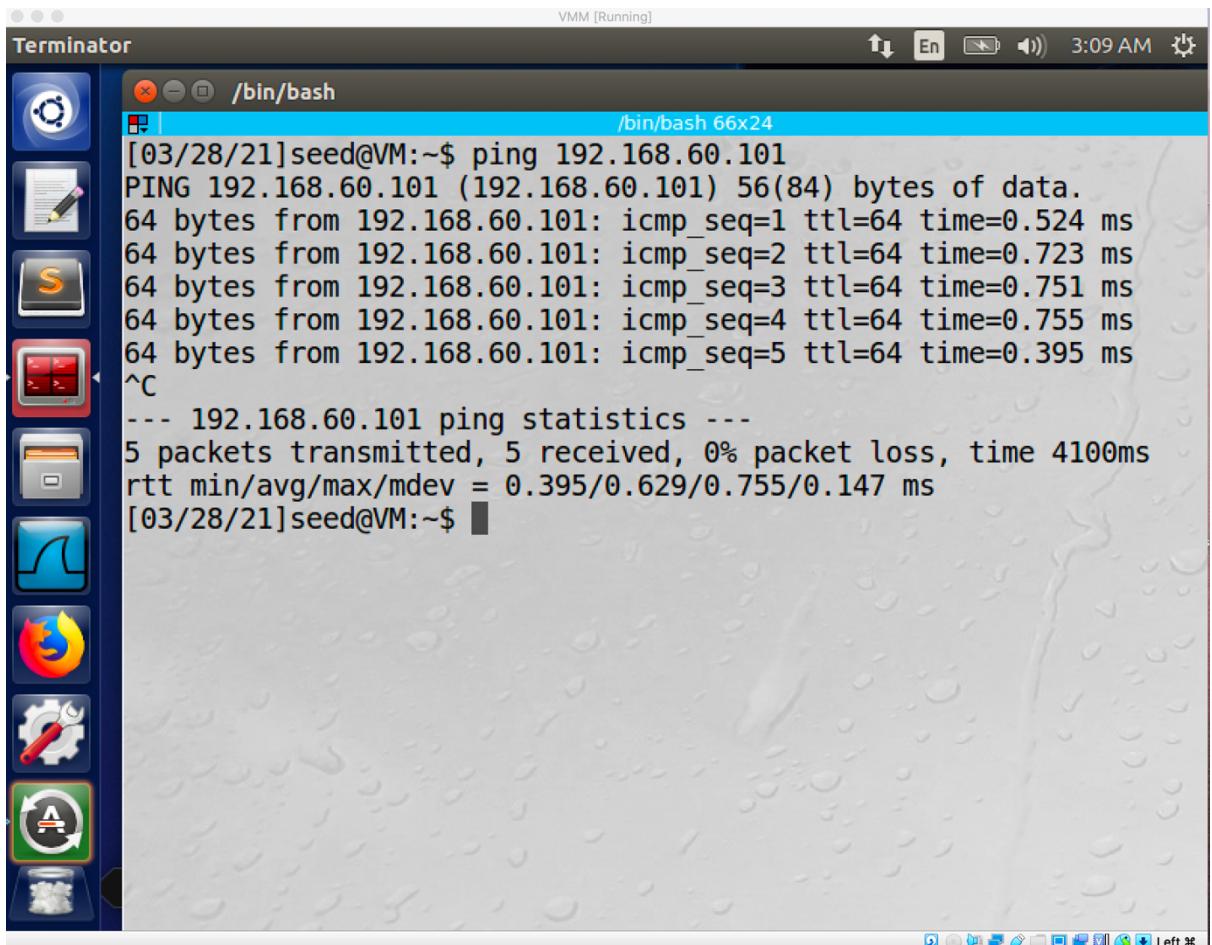
Terminator /bin/bash /bin/bash 70x24

```
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:294 errors:0 dropped:0 overruns:0 frame:0
      TX packets:294 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1
      RX bytes:51320 (51.3 KB) TX bytes:51320 (51.3 KB)

[03/28/21]seed@VM:~$ ping 912.168.60.1
ping: unknown host 912.168.60.1
[03/28/21]seed@VM:~$ ^C
[03/28/21]seed@VM:~$ ping 192.168.60.1
PING 192.168.60.1 (192.168.60.1) 56(84) bytes of data.
64 bytes from 192.168.60.1: icmp_seq=1 ttl=64 time=0.898 ms
64 bytes from 192.168.60.1: icmp_seq=2 ttl=64 time=0.345 ms
64 bytes from 192.168.60.1: icmp_seq=3 ttl=64 time=0.434 ms
64 bytes from 192.168.60.1: icmp_seq=4 ttl=64 time=0.480 ms
64 bytes from 192.168.60.1: icmp_seq=5 ttl=64 time=0.351 ms
64 bytes from 192.168.60.1: icmp_seq=6 ttl=64 time=0.503 ms
^C
--- 192.168.60.1 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5129ms
rtt min/avg/max/mdev = 0.345/0.501/0.898/0.189 ms
[03/28/21]seed@VM:~$
```

The ping operation is successful.

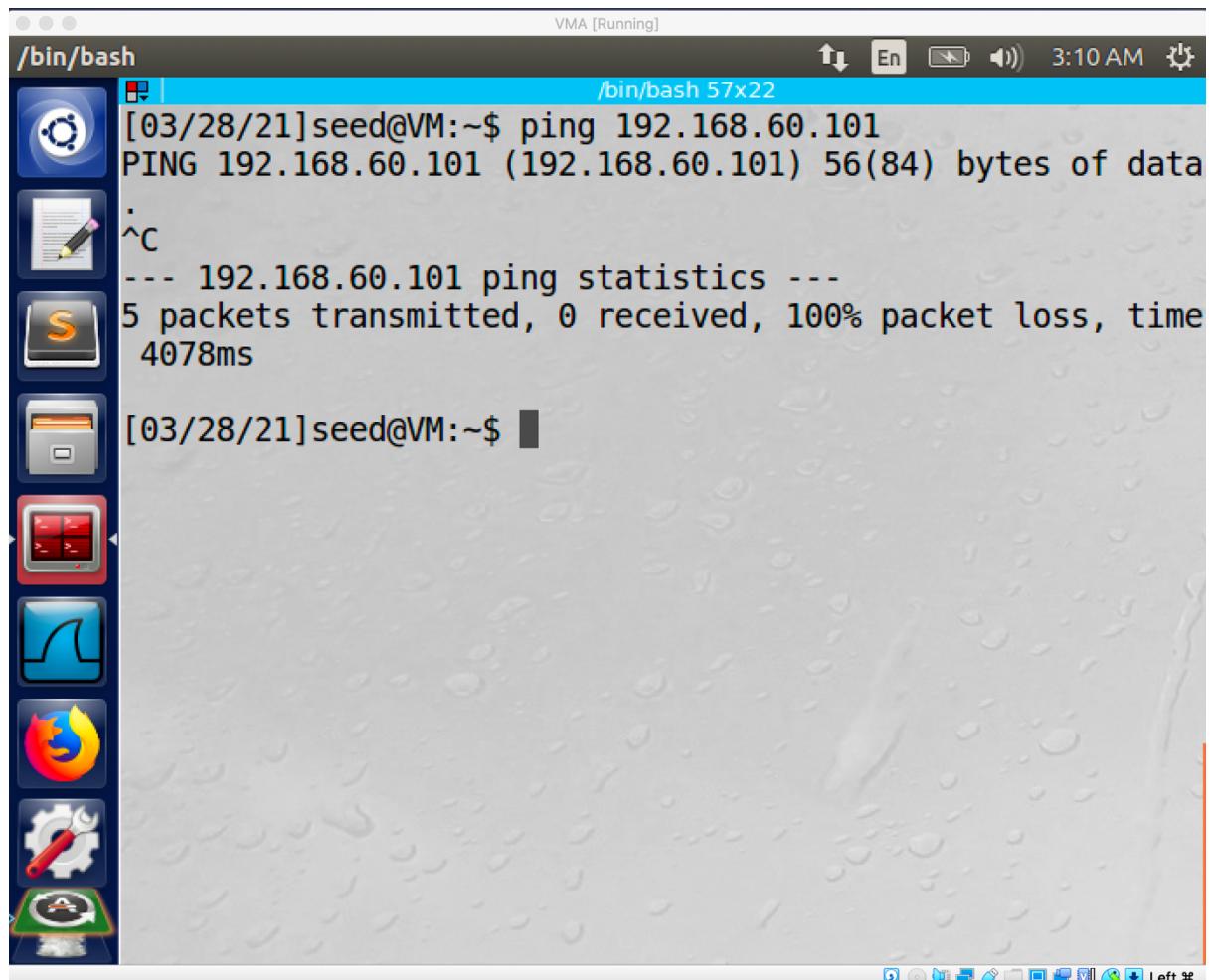
2. Ping host V from VPN server.



```
[03/28/21]seed@VM:~$ ping 192.168.60.101
PING 192.168.60.101 (192.168.60.101) 56(84) bytes of data.
64 bytes from 192.168.60.101: icmp_seq=1 ttl=64 time=0.524 ms
64 bytes from 192.168.60.101: icmp_seq=2 ttl=64 time=0.723 ms
64 bytes from 192.168.60.101: icmp_seq=3 ttl=64 time=0.751 ms
64 bytes from 192.168.60.101: icmp_seq=4 ttl=64 time=0.755 ms
64 bytes from 192.168.60.101: icmp_seq=5 ttl=64 time=0.395 ms
^C
--- 192.168.60.101 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4100ms
rtt min/avg/max/mdev = 0.395/0.629/0.755/0.147 ms
[03/28/21]seed@VM:~$
```

The ping operation is succeeded.

3. Ping host V from host U.



The ping operation is not successful.

Task2: Create and configure the TUN interface

- Name of the interface:

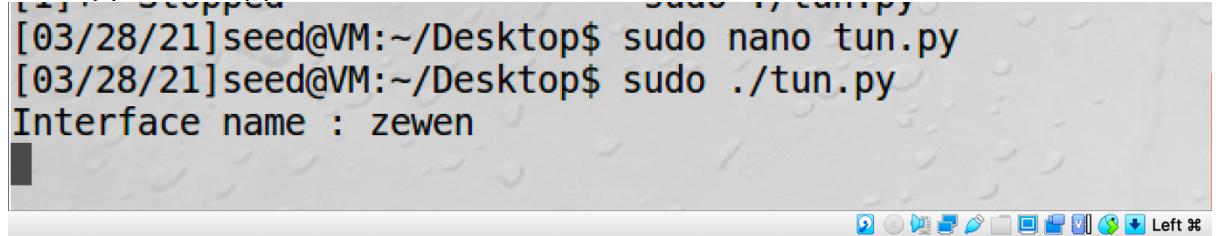
Firstly, I create a python file(tun.py) to set up a tun interface, then I modified the name of the tun interface to my last name.

```

lab6 > tun.py > ...
2
3     import fcntl
4     import struct
5     import os
6     import time
7     from scapy.all import *
8
9     TUNSETIFF = 0x400454ca
10    IFF_TUN = 0x0001
11    IFF_TAP = 0x0002
12    IFF_NO_PI = 0x1000
13    # create TUN interface
14    tun = os.open("/dev/net/tun", os.O_RDWR)
15    # ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
16    # change the name of the tun interface
17    ifr = struct.pack('16sH', b'zewen', IFF_TUN | IFF_NO_PI)
18    ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
19
20    # get the interface name
21    ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
22    print("Interface name : {}".format(ifname))
23
24    # while (True):
25    #     time.sleep(10)
26    while (True):
27        time.sleep(10)
28

```

Then I run this python file by using the command “chmod a+x tun.py” and command “sudo python tun.py”.



```
[03/28/21]seed@VM:~/Desktop$ sudo nano tun.py
[03/28/21]seed@VM:~/Desktop$ sudo ./tun.py
Interface name : zewen
```

As you can see the name of the interface has become my last name.

Secondly, I run the command “ip address” on a separate terminal.

```
/bin/bash 57x22
ll it by typing:
sudo apt install iprint
[03/28/21]seed@VM:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:d6:c1:35 brd ff:ff:ff:ff:ff:ff
        inet 10.0.2.7/24 brd 10.0.2.255 scope global dynamic
          enp0s3
            valid_lft 408sec preferred_lft 408sec
        inet6 fe80::bb01:4d92:757a:36c3/64 scope link
            valid_lft forever preferred_lft forever
3: tun0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 500
    link/none
[03/28/21]seed@VM:~$
```

Since there is no ip address bind to this interface yet, we only can see there is a new tun0 interface.

b. Set up The TUN interface

I use the command “ sudo ip addr add 192.168.53.99/24 dev zewen” and command “sudo ip link set dev tun0 up” to assign an IP address to the interface and bring up the interface. I also add two lines of code to the tun.py, then every time I run this python file, it will automatically assign the ip address to the interface then bring it up.

```

lab6 > tun.py > ...
2
3   import fcntl
4   import struct
5   import os
6   import time
7   from scapy.all import *
8
9   TUNSETIFF = 0x400454ca
10  IFF_TUN = 0x0001
11  IFF_TAP = 0x0002
12  IFF_NO_PI = 0x1000
13  # create TUN interface
14  tun = os.open("/dev/net/tun", os.O_RDWR)
15  # ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
16  # change the name of the tun interface
17  ifr = struct.pack('16sH', b'zewen', IFF_TUN | IFF_NO_PI)
18  ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
19
20  # get the interface name
21  ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
22  print("Interface name : {}".format(ifname))
23  os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
24  os.system("ip link set dev {} up".format(ifname))
25  while (True):
26      time.sleep(10)
27

```

Then I run the command “ ip address” again.

```

3: zewen: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN group default qlen 500
    link/none
        inet 192.168.53.99/24 scope global zewen
            valid_lft forever preferred_lft forever
        inet6 fe80::6761:26ce:f12:2f5d/64 scope link flags 0
            valid_lft forever preferred_lft forever

```

As you can see the IP address has been assigned to this interface.

- Read from the TUN interface.

I create a tun1.py to read the traffic.

```

2
3   import fcntl
4   import struct
5   import os
6   import time
7   from scapy.all import *
8
9   TUNSETIFF = 0x400454ca
10  IFF_TUN = 0x0001
11  IFF_TAP = 0x0002
12  IFF_NO_PI = 0x1000
13  # create TUN interface
14  tun = os.open("/dev/net/tun", os.O_RDWR)
15  ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
16  # change the name of the tun interface
17  ifr = struct.pack('16sH', b'zewen', IFF_TUN | IFF_NO_PI)
18  ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
19
20  # get the interface name
21  ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
22  print("Interface name : {}".format(ifname))
23  os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
24  os.system("ip link set dev {} up".format(ifname))
25  while (True):
26      packet = os.read(tun, 2048)
27      if True:
28          ip = IP(packet)
29          ip.show()

```

On Host U, I ping the 192.168.53.2, then the interface will show this packet information.

```
len      = 84
id       = 16197
flags    = DF
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0xfae
src      = 192.168.53.99
dst      = 192.168.53.2
\options \
###[ ICMP ]###
    type    = echo-request
    code    = 0
    chksum = 0xad1c
    id     = 0xe30
    seq    = 0x5
###[ Raw ]###
    load    = '\xb2P``:\xfa\x04\x00\x08\t\n\x0b\x0c
\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1
b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

- d. I create a tun2.py file by modifying the original tun.py file.

```

lab6 > tun2.py > ...
1  #!/usr/bin/python3
2
3  import fcntl
4  import struct
5  import os
6  import time
7  from scapy.all import *
8
9  TUNSETIFF = 0x400454ca
10 IFF_TUN = 0x0001
11 IFF_TAP = 0x0002
12 IFF_NO_PI = 0x1000
13 # create TUN interface
14 tun = os.open("/dev/net/tun", os.O_RDWR)
15 # ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
16 # change the name of the tun interface
17 ifr = struct.pack('16sH', b'zewen', IFF_TUN | IFF_NO_PI)
18 ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
19
20 # get the interface name
21 ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
22 print("Interface name : {}".format(ifname))
23 os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
24 os.system("ip link set dev {} up".format(ifname))
25 while (True):
26     packet = os.read(tun, 2048)
27     if True:
28         ip = IP(packet)
29         ip.show()
30         newip = IP(src='1.2.3.4', dst=ip.src)
31         newpkt = newip / ip.payload
32         os.write(tun, bytes(newpkt))
33

```

After the tun received a packet, I will modify the ip header by changing it to 1.2.3.4

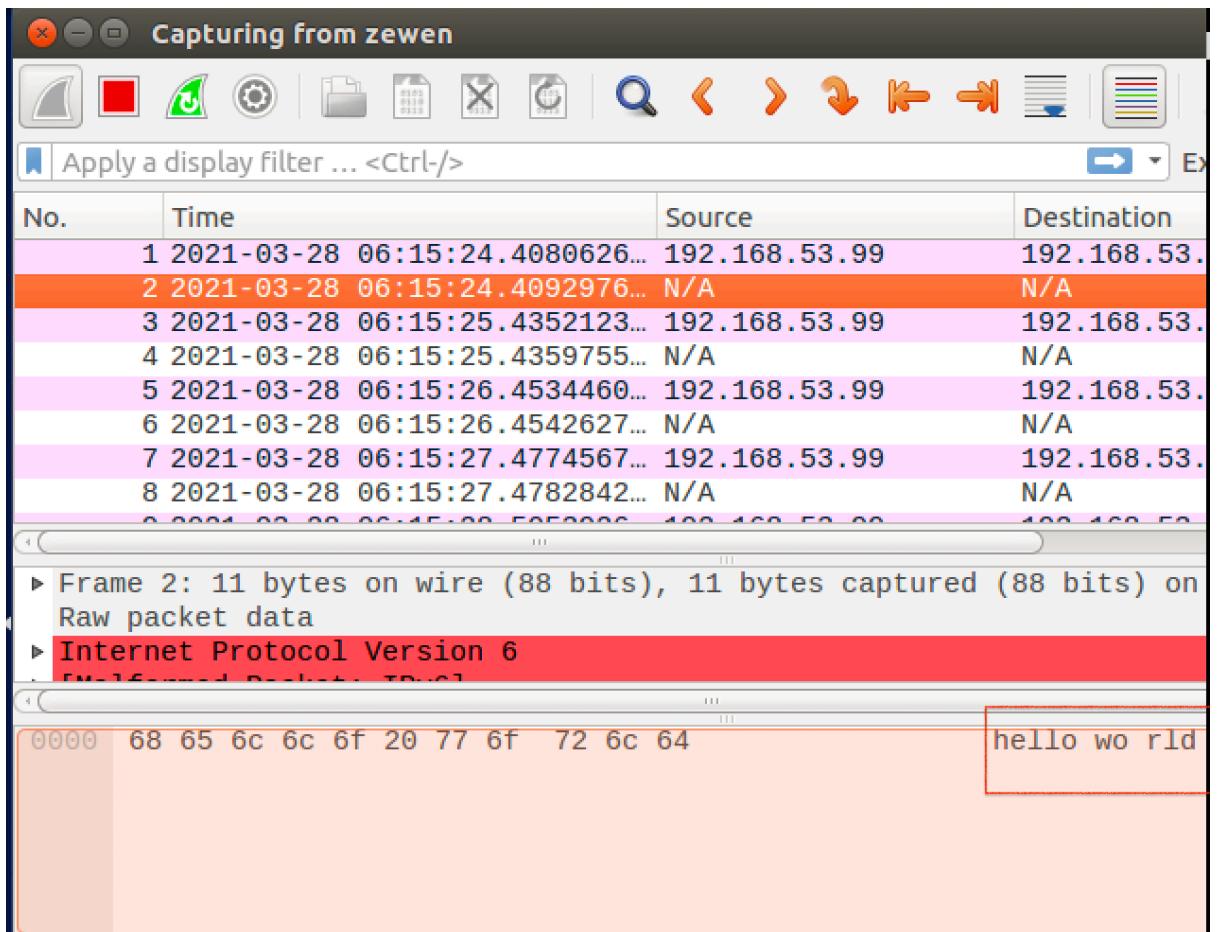
	Source	Destination	Protocol	Length
:50:32.0010479...	192.168.53.99	192.168.53.2	ICMP	8
:50:32.0028630...	1.2.3.4	192.168.53.99	ICMP	8
:50:33.0348161...	192.168.53.99	192.168.53.2	ICMP	8
:50:33.0370516...	1.2.3.4	192.168.53.99	ICMP	8
:50:34.0381014...	192.168.53.99	192.168.53.2	ICMP	8
:50:34.0419453...	1.2.3.4	192.168.53.99	ICMP	8
:50:35.0623688...	192.168.53.99	192.168.53.2	ICMP	8
:50:35.0639227...	1.2.3.4	192.168.53.99	ICMP	8
:50:36.0000150...	192.168.53.99	192.168.53.2	ICMP	8

As you can see, the reply packet has the source ip address of 1.2.3.4.

Moreover, I also created a tun3.py file and add “hello world” as the data payload.

```
labb6 > tun3.py > ...
1  #!/usr/bin/python3
2
3  import fcntl
4  import struct
5  import os
6  import time
7  from scapy.all import *
8
9  TUNSETIFF = 0x400454ca
10 IFF_TUN = 0x0001
11 IFF_TAP = 0x0002
12 IFF_NO_PI = 0x1000
13 # create TUN interface
14 tun = os.open("/dev/net/tun", os.O_RDWR)
15 # ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
16 # change the name of the tun interface
17 ifr = struct.pack('16sH', b'zewen', IFF_TUN | IFF_NO_PI)
18 ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
19
20 # get the interface name
21 ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
22 print("Interface name : {}".format(ifname))
23 os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
24 os.system("ip link set dev {} up".format(ifname))
25 while (True):
26     packet = os.read(tun, 2048)
27     if True:
28         ip = IP(packet)
29         ip.show()
30         #newip = IP(src = '1.2.3.4',dst = ip.src)
31         # #newpkt = newip/ip.payload
32         # #os.write(tun,bytes(newpkt))
33         os.write(tun, "hello world")
34
```

Then, the data payload of the packets we get from the tun interface will become “hello world”.



Task3: Send the IP packet to VPN Server Through the tunnel

I create the tun_server.py and the tun_client.py file.

```
lab6 > tun_server.py > ...
1  from scapy.all import *
2  IP_A = "10.0.2.9"
3  PORT = 9090
4
5  sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
6  sock.bind((IP_A, PORT))
7
8  while True:
9      data, (ip, port) = sock.recvfrom(2048)
10     print("{}:{} --> {}:{}".format(ip, port, IP_A, PORT))
11     pkt = IP(data)
12     print("    Inside: {} ----> {}".format(pkt.src, pkt.dst))
13
```

```

1  #!/usr/bin/python3
2
3  import fcntl
4  import struct
5  import os
6  import time
7  from scapy.all import *
8
9  TUNSETIFF = 0x400454ca
10 IFF_TUN = 0x0001
11 IFF_TAP = 0x0002
12 IFF_NO_PI = 0x1000
13 # create TUN interface
14 tun = os.open("/dev/net/tun", os.O_RDWR)
15 # ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
16 # change the name of the tun interface
17 ifr = struct.pack('16sH', b'zewen', IFF_TUN | IFF_NO_PI)
18 ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
19
20 # get the interface name
21 ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
22 print("Interface name : {}".format(ifname))
23 os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
24 os.system("ip link set dev {} up".format(ifname))
25
26 # Create a UDP socket
27 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
28 while (True):
29     # get the packet from the tun interface
30     packet = os.read(tun, 2048)
31     if True:
32         # Send packet through channel
33         sock.sendto(packet, ("10.0.2.9", 9090))
34

```

After I run the tun.server.py at the VPN server and the tun.client.py at the client part. I ping 192.168.53.5 then the server-side will print out the actual packet flow is from 192.168.53.99 which is the ip address of the tun interface to the 192.168.53.5. However, the outside packet flow is from 10.0.2.7 to 10.0.2.9. It is because, the 192.168.53.5 is within the same subnet with the interface, so it will directly go through the interface.

```
[03/28/21]seed@VM:~$ ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
18 packets transmitted, 0 received, 100% packet loss, time
e 17423ms
```

```
[03/28/21]seed@VM:~$ █
```

```
10.0.2.7:40577 --> 10.0.2.9:9090
  Inside: 192.168.53.99 ----> 192.168.60.5
10.0.2.7:40577 --> 10.0.2.9:9090
  Inside: 192.168.53.99 ----> 192.168.60.5
```

Since the 192.168.60.1 is not within the same subnet as the interface, we need to add the route to make sure all the packets going to the 192.168.60.0/24 network should be routed to the tun interface. To do that, I use the command “ sudo ip route add 192.168.60.0/24 dev zewen via 192.168.53.1” on the client-side. After applying this command, I ping 192.168.60.101 from the client-side, the server-side will print out the inside packet is coming from 192.168.53.99 and send it to 192.168.60.101.

```
10.0.2.7:57057 --> 10.0.2.9:9090
  Inside: 192.168.53.99 ----> 192.168.60.101
```

Task4: Set Up the VPN Server

By referring to tun_server.py, I construct the tun_server_task_4.py file by combine three parts.

- 1) Create a TUN interface and configure it.
- 2) Get the data from the socket interface; treat the received data as an IP packet.
- 3) Write the packet to the TUN interface.

```

3  import fcntl
4  import struct
5  import os
6  import time
7  from scapy.all import *          Part 1
8
9  TUNSETIFF = 0x400454ca
10 IFF_TUN = 0x0001
11 IFF_TAP = 0x0002
12 IFF_NO_PI = 0x1000
13 # create TUN interface
14 tun = os.open("/dev/net/tun", os.O_RDWR)
15 # ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
16 # change the name of the tun interface
17 ifr = struct.pack('16sH', b'zewen', IFF_TUN | IFF_NO_PI)
18 ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
19
20 # get the interface name
21 ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
22 print("Interface name : {}".format(ifname))
23 os.system("ip addr add 192.168.53.90/24 dev {}".format(ifname))
24 os.system("ip link set dev {} up".format(ifname))
25 IP_A = "10.0.2.9"
26 PORT = 9090
27
28 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)      part 2
29 sock.bind((IP_A, PORT))
30 while True:
31     data, (ip, port) = sock.recvfrom(2048)
32     print("{}:{} --> {}:{}".format(ip, port, IP_A, PORT))
33     pkt = IP(data)
34     print(" Inside: {} -----> {}".format(pkt.src, pkt.dst))    part 3
35     os.write(tun, bytes(pkt))

```

Moreover, on the client-side, I also run the tun_client_task_4.py to send all the packets which have the destination within the 192.168.60.0/24 network to the VPN server through the UDP socket.

```

2
3     import fcntl
4     import struct
5     import os
6     import time
7     from scapy.all import *
8
9     TUNSETIFF = 0x400454ca
10    IFF_TUN = 0x0001
11    IFF_TAP = 0x0002
12    IFF_NO_PI = 0x1000
13
14    # create TUN interface
15    tun = os.open("/dev/net/tun", os.O_RDWR)
16    # ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
17    # change the name of the tun interface
18    ifr = struct.pack('16sH', b'zewen', IFF_TUN | IFF_NO_PI)
19    ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
20
21    # get the interface name
22    ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
23    print("Interface name : {}".format(ifname))
24    os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
25    os.system("ip link set dev {} up".format(ifname))
26    #IP_A = "10.0.2.7"
27    #PORT = 9090
28
29    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
30    #sock.bind((IP_A, PORT))
31    while True:
32        packet = os.read(tun, 2048)
33        if True:
34            sock.sendto(packet, ("10.0.2.9", 9090))

```

After I run the python file at the client-side, I also add the route which will guide all the packets send to the 192.168.60.0/24 network go through the ‘zewen’ interface by using the command “ sudo ip route add 192.168.60.0/24 dev zewen via 192.168.53.1”. Moreover, I also set up the VPN server as a gateway by enabling the IP forwarding using the command “ sudo sysctl net.ipv4.ip_forward=1” at the server-side. Moreover, a need route also needs to add to the host v to make sure all the traffic will go to the gateway which is the VPN server.

Destination	Gateway	Genmask	Flags	Metric	Ref	Us
0.0.0.0	192.168.60.1	0.0.0.0	UG	100	0	
169.254.0.0	0.0.0.0	255.255.0.0	U	1000	0	
192.168.60.0	0.0.0.0	255.255.255.0	U	100	0	

After setting up everything I ping 192.168.60.101 at the client-side.

	Source	Destination	Protocol	Length	Info
355357...	10.0.2.7	10.0.2.9	UDP	126	35893 → 9090 Len=84
364358...	192.168.53.99	192.168.60.101	ICMP	98	Echo (ping) request
364557...	192.168.60.101	192.168.53.99	ICMP	98	Echo (ping) reply
594819...	10.0.2.7	10.0.2.9	UDP	126	35893 → 9090 Len=84
504980...	192.168.53.99	192.168.60.101	ICMP	98	Echo (ping) request
505158...	192.168.60.101	192.168.53.99	ICMP	98	Echo (ping) reply
339827...	10.0.2.7	10.0.2.9	UDP	126	35893 → 9090 Len=84
347730...	192.168.53.99	192.168.60.101	ICMP	98	Echo (ping) request
347918...	192.168.60.101	192.168.53.99	ICMP	98	Echo (ping) reply
091810...	10.0.2.7	10.0.2.9	UDP	126	35893 → 9090 Len=84
099830...	192.168.53.99	192.168.60.101	ICMP	98	Echo (ping) request
100048...	192.168.60.101	192.168.53.99	ICMP	98	Echo (ping) reply
220670...	10.0.2.7	10.0.2.9	UDP	126	35893 → 9090 Len=84

As you can see, the host v can receive the ICMP request packets and generate reply packets which will be discarded due to non-complete VPN implementation.

Task5: Handling Traffic in Both Directions

To handle traffic in both directions, I implement the select() method to handle both the socket and tun interface packets on both the client and server-side. Please refer to tun_client_task_5.py and tun_server_task_5.py.

```

3   import fcntl
4   import struct
5   import os
6   import time
7   from scapy.all import *
8
9   TUNSETIFF = 0x400454ca
10  IFF_TUN = 0x0001
11  IFF_TAP = 0x0002
12  IFF_NO_PI = 0x1000
13  # create TUN interface
14  tun = os.open("/dev/net/tun", os.O_RDWR)
15  # ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
16  # change the name of the tun interface
17  ifr = struct.pack('16sH', b'zewen', IFF_TUN | IFF_NO_PI)
18  ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
19
20  # get the interface name
21  ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
22  print("Interface name : {}".format(ifname))
23  os.system("ip addr add 192.168.53.90/24 dev {}".format(ifname))
24  os.system("ip link set dev {} up".format(ifname))
25  IP_A = "10.0.0.2.7"
26  PORT = 9090
27
28  sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
29  sock.bind((IP_A, PORT))
30  while True:
31      # this will block until at least one interface is ready
32      ready, _ = select([sock, tun], [], [])
33      for fd in ready:
34          if fd is sock:
35              data, (ip, port) = sock.recvfrom(2048)
36              pkt = IP(data)
37              print(" From socket <==: {} ----> {}".format(pkt.src, pkt.dst))
38              os.write(tun, bytes(pkt))
39          if fd is tun:
40              packet = os.read(tun, 2048)
41              pkt = IP(packet)
42              print(" From tun <==: {} ----> {}".format(pkt.src, pkt.dst))
43              sock.sendto(packet, ("10.0.2.9", 9090))
44

```

Client Ip and Port number

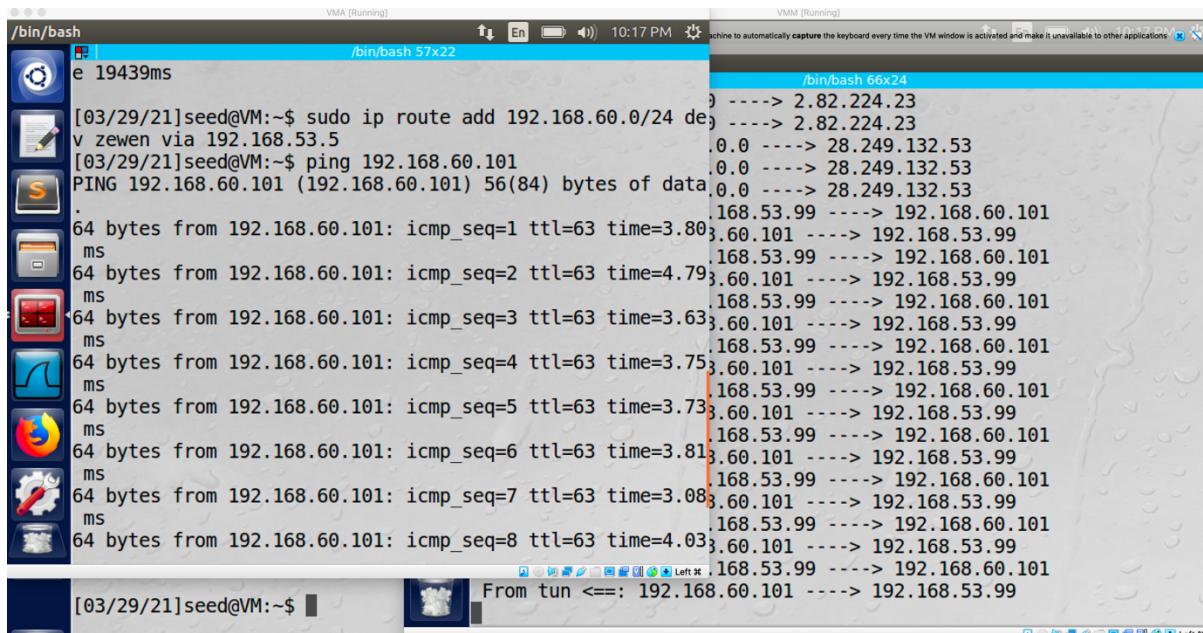
Sent to Server

```

2      import fcntl
3      import struct
4      import os
5      import time
6      from scapy.all import *
7
8
9      TUNSETIFF = 0x400454ca
10     IFF_TUN = 0x0001
11     IFF_TAP = 0x0002
12     IFF_NO_PI = 0x1000
13
14     # create TUN interface
15     tun = os.open("/dev/net/tun", os.O_RDWR)
16     # ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
17     # change the name of the tun interface
18     ifr = struct.pack('16sH', b'zewen', IFF_TUN | IFF_NO_PI)
19     ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
20
21     # get the interface name
22     ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
23     print("Interface name : {}".format(ifname))
24     os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
25     os.system("ip link set dev {} up".format(ifname))
26
27     IP_A = "10.0.2.9"           Server IP and port number
28     PORT = 9090
29
30     sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
31     sock.bind((IP_A, PORT))
32
33     while True:
34         # this will block until at least one interface is ready
35         ready, _, _ = select([sock, tun], [], [])
36         for fd in ready:
37             if fd is sock:
38                 data, (ip, port) = sock.recvfrom(2048)
39                 pkt = IP(data)
40                 print(" From socket <==: {} ----> {}".format(pkt.src, pkt.dst))
41                 os.write(tun, bytes(pkt))
42             if fd is tun:
43                 packet = os.read(tun, 2048)
44                 pkt = IP(packet)
45                 print(" From tun <==: {} ----> {}".format(pkt.src, pkt.dst))
46                 sock.sendto(packet, ("10.0.2.7", 9090))    send to client

```

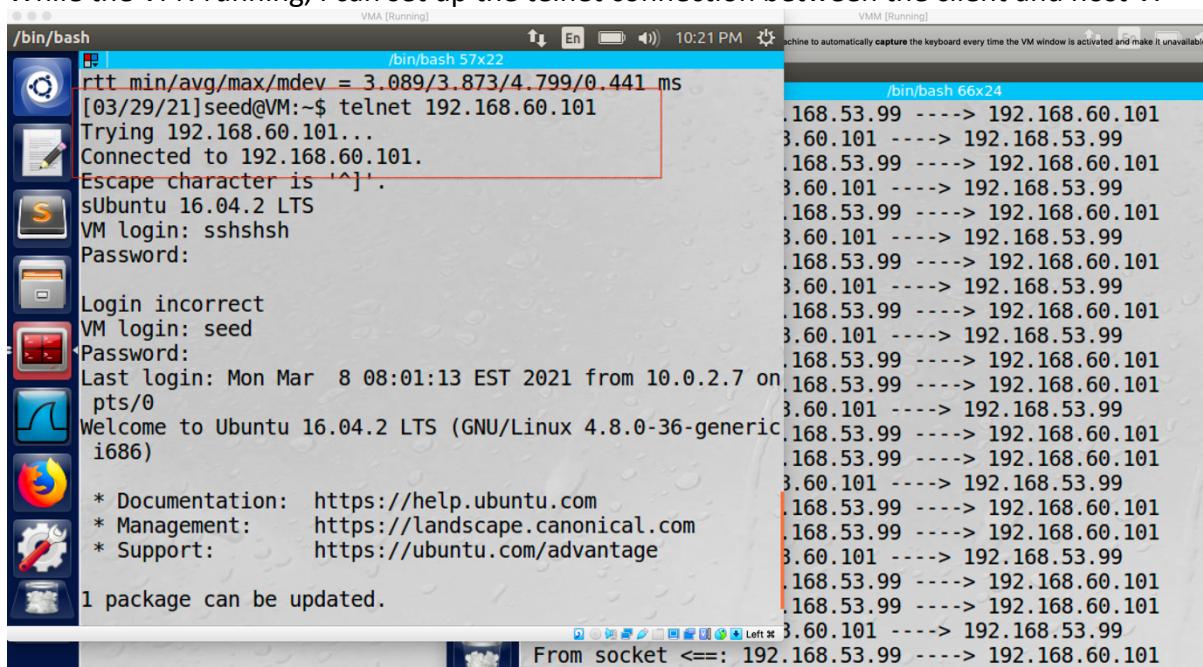
After I run these two python files at both server and client-side, I also add the same route which added at the client-side and host v side and enables the ip forwarding at the server-side. Then I ping host v at the client-side.



As you can see, I can ping to host v on the client-side and the VPN server is handling all the message traffic.

Task6: Tunnel-Breaking Experiment

While the VPN running, I can set up the telnet connection between the client and host V.



And I can type during the telnet connection. However, if I break the VPN connection by stopping running the tun interface at the server part. I am not able to type anymore.

However, the TCP connection will not break, when I re-run the interface file at the server-side, all the character I typed before will appears again since after the VPN connection is broken, whatever I typed will be stored into the queue when the connection is established again, the tun interface will resend the packets which stored in the queue.