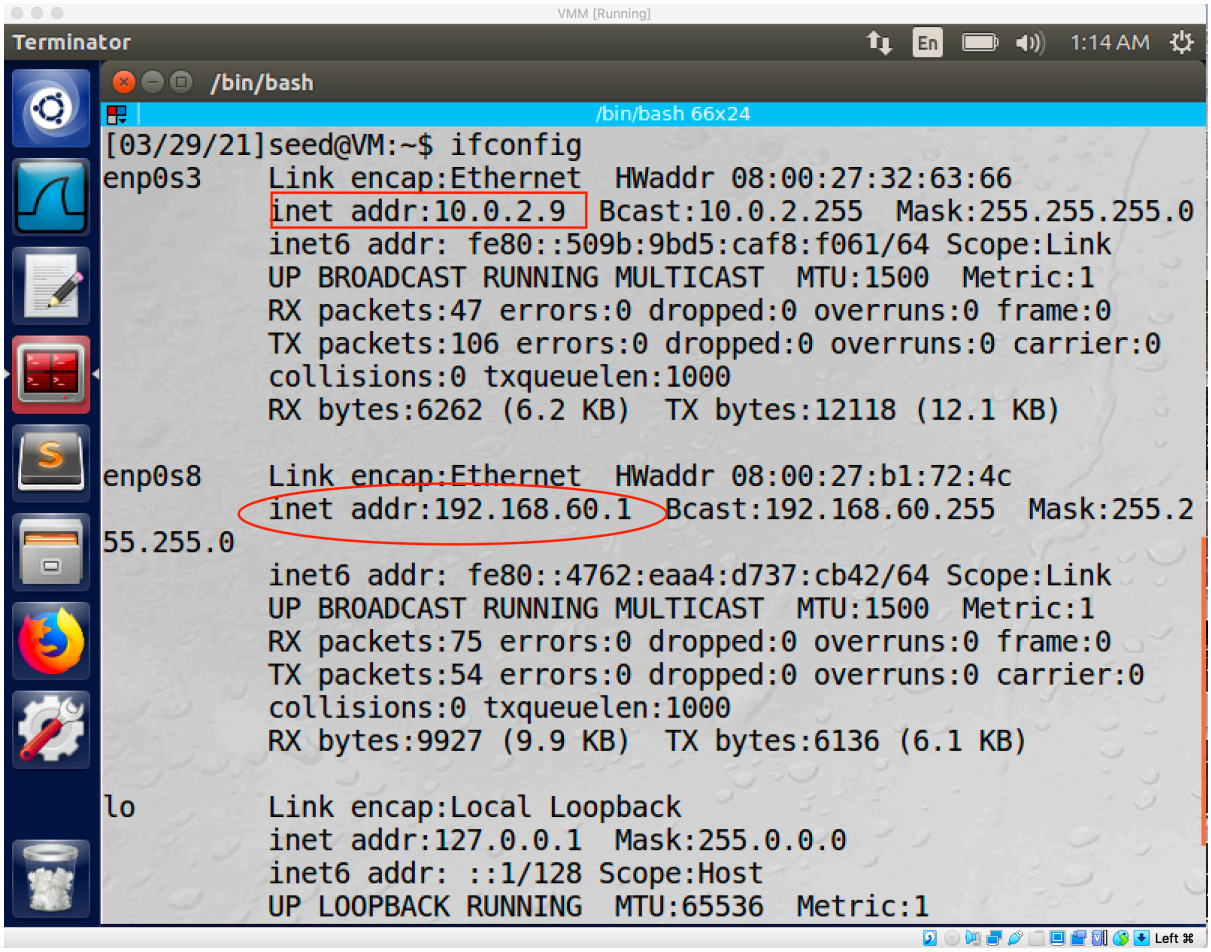
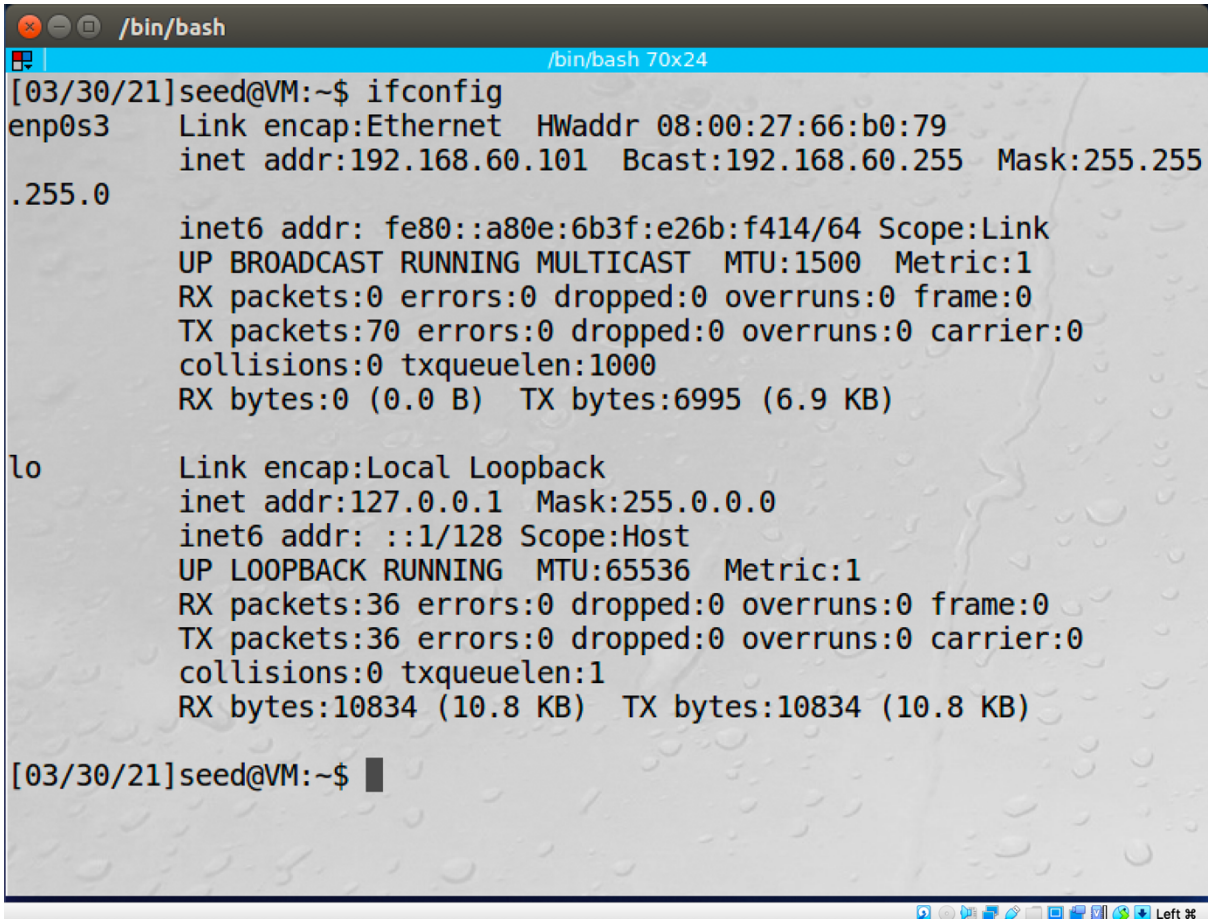
Client VM: IP: 10.0.2.7
Server VM: IP1: 10.0.2.9, IP2: 192.168.60.1
Host v: IP: 192.168.60.101

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.

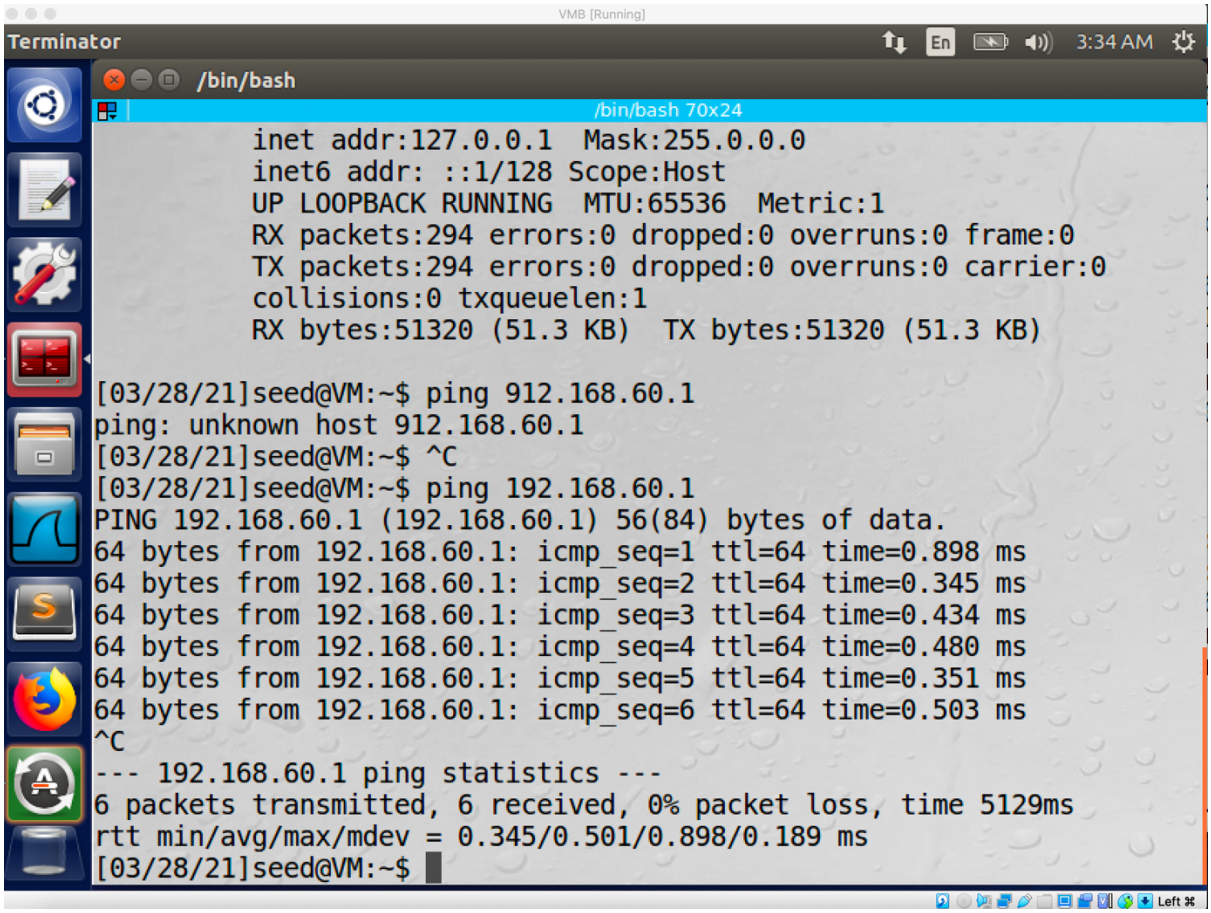


As for the host v, I create a new connection for it which set up the ipv4 address as 192.168.60.101 to make sure it within the same private network with the server, which means the client cannot access it.



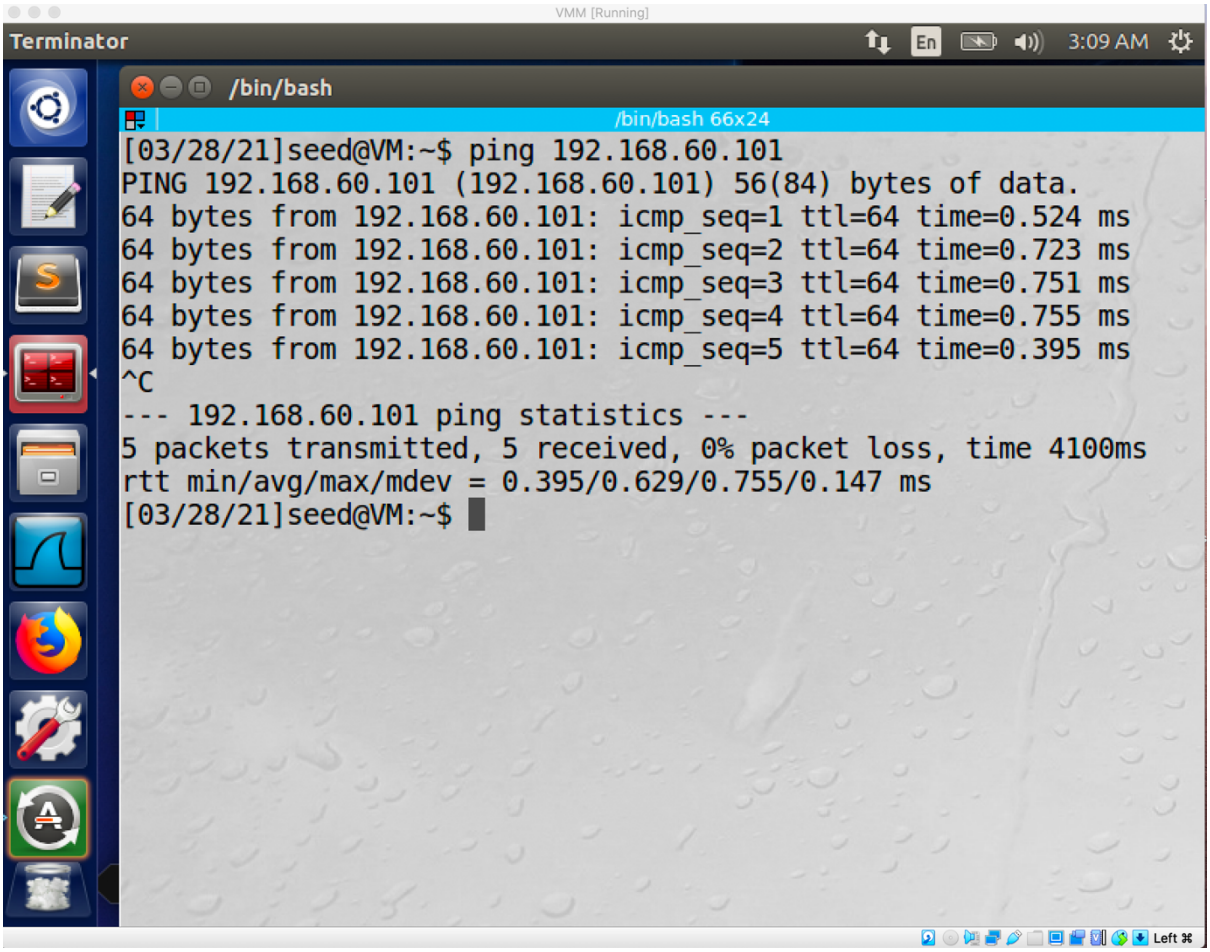
To test this setup, several tests have been done.

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



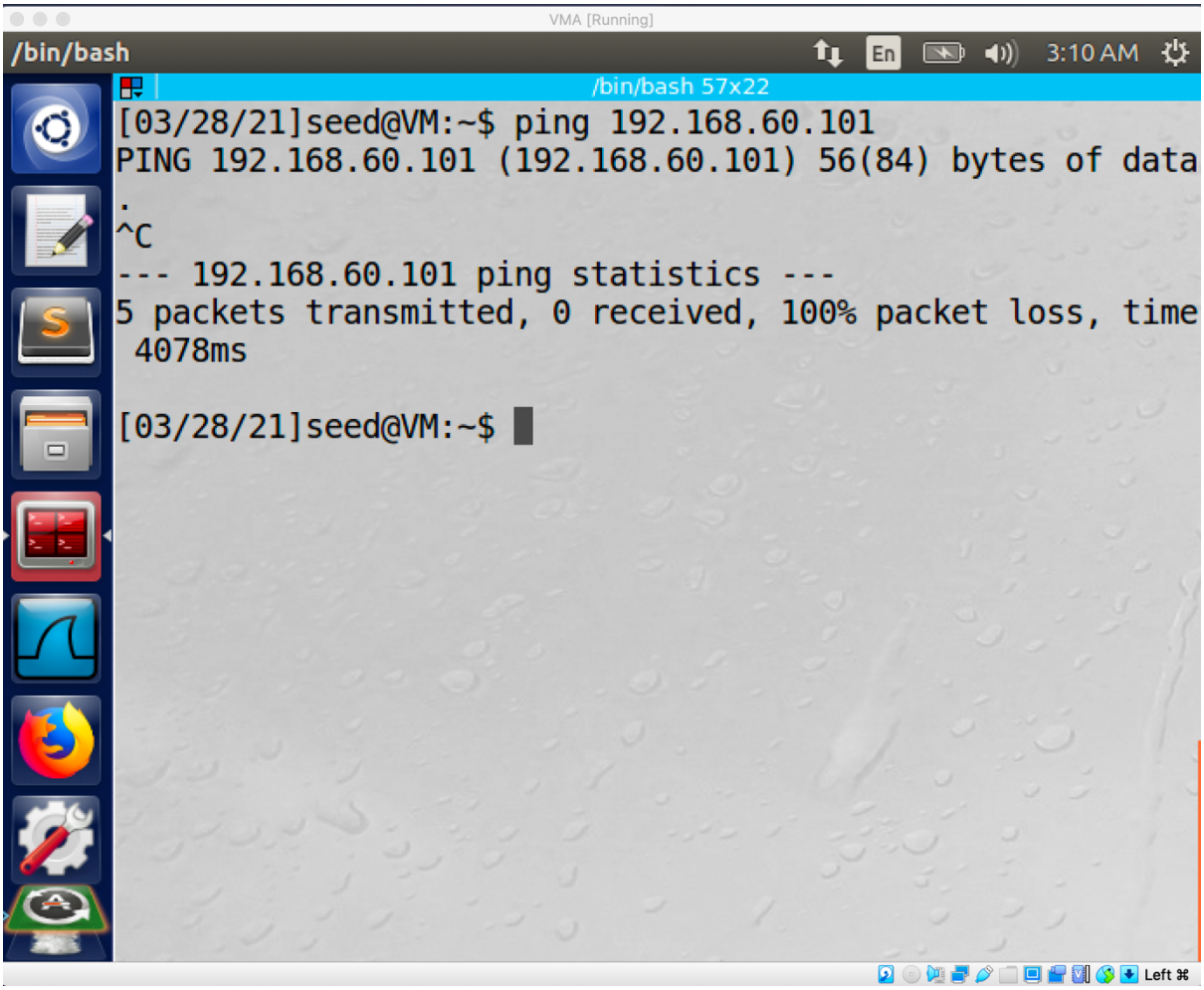
The ping operation is successful.

2. Ping host V from VPN server.



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

a. 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.

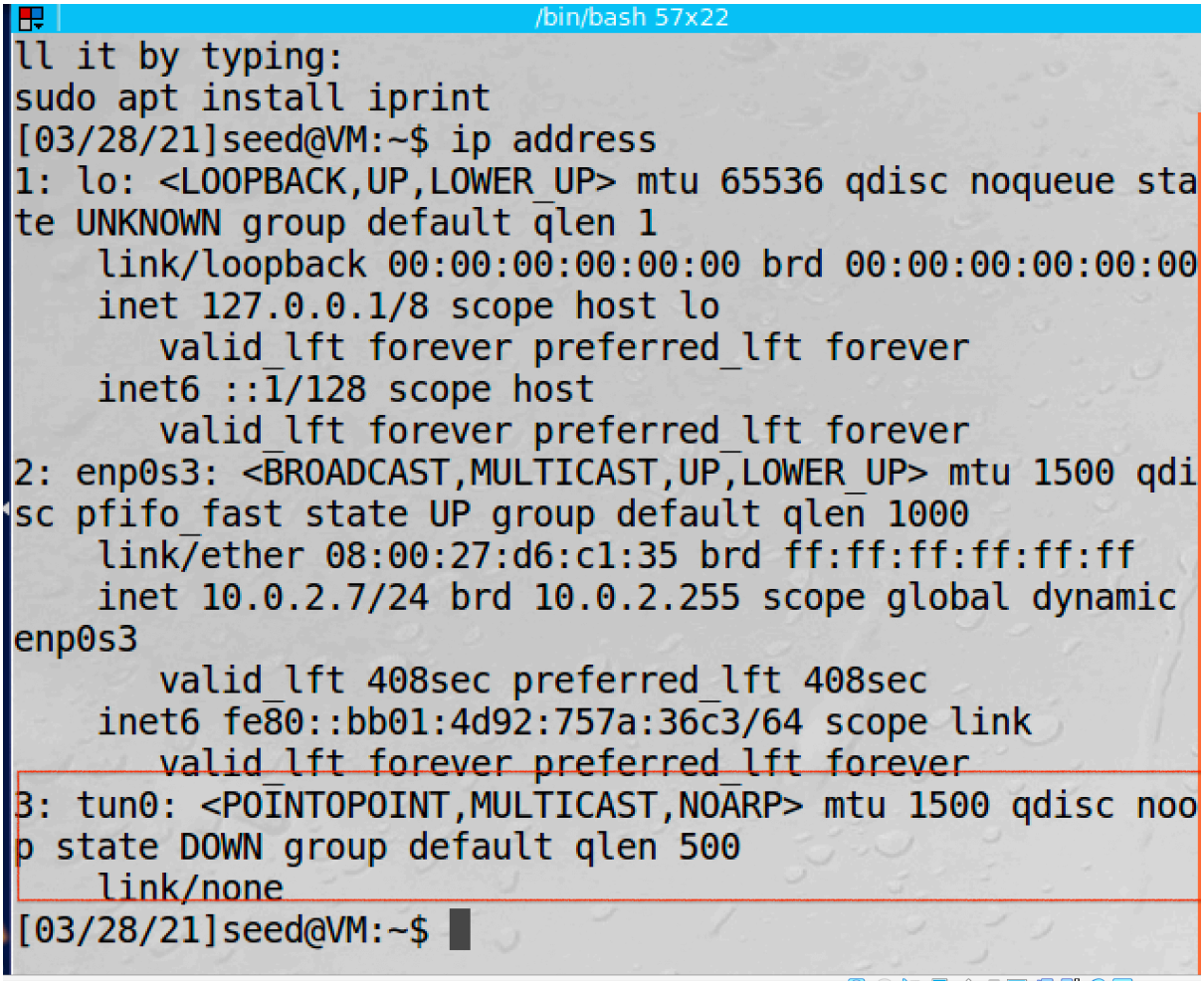
```
lab6 > tun.py > ...
 2
 3  import fcntl
 4  import struct
 5  import os
 6  import time
 7  from scapy.all import *
 8
 9  TUNSETIFF = 0x400454ca
10  IFF_TUN = 0x0001
11  IFF_TAP = 0x0002
12  IFF_NO_PI = 0x1000
13  # create TUN interface
14  tun = os.open("/dev/net/tun", os.O_RDWR)
15  # ifr = struct.pack('16sH', 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.



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 1
500 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 80
0
       valid_lft forever preferred_lft forever
```

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

c. Read from the TUN interface.

I create a tun1.py to read the traffic.

```python

import fcntl
import struct
import os
import time
from scapy.all import *

TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# create TUN interface
tun = os.open("/dev/net/tun", os.O_RDWR)
# ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
# change the name of the tun interface
ifr = struct.pack('16sH', b'zewen', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)

# get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface name : {}".format(ifname))
os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
while (True):
    packet = os.read(tun, 2048)
    if True:
        ip = IP(packet)
        ip.show()
```

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

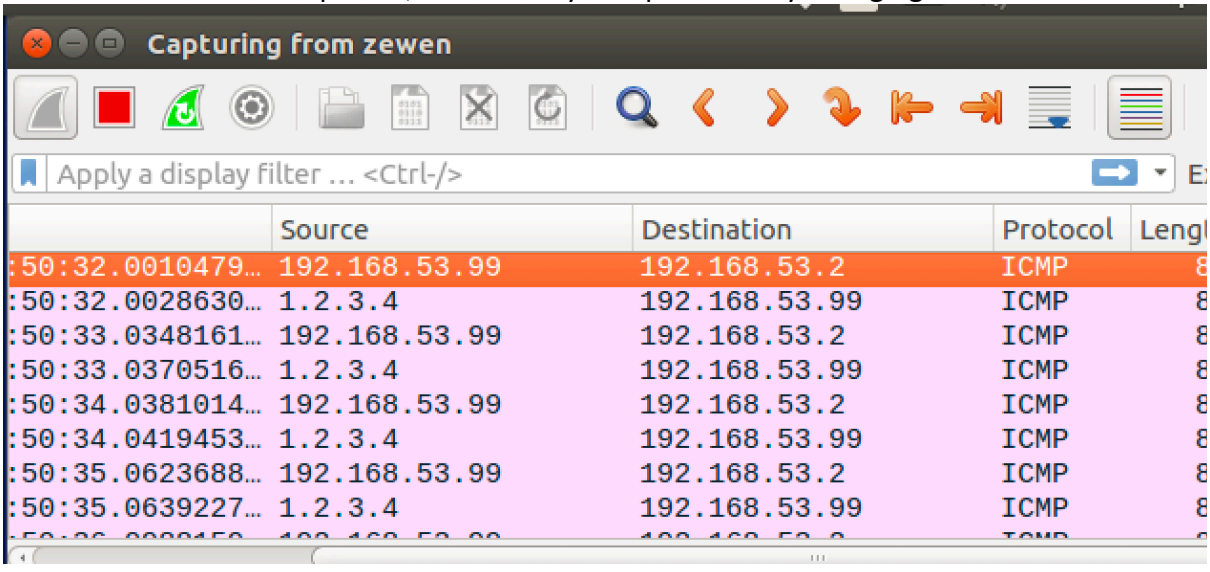
```
  len       = 84
  id        = 16197
  flags     = DF
  frag      = 0
  ttl       = 64
  proto     = icmp
  chksum    = 0xfae
  src       = 192.168.53.99
  dst       = 192.168.53.2
  \options   \
###[ ICMP ]###
     type      = echo-request
     code      = 0
     chksum    = 0xad1c
     id        = 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 > ...

```python
#!/usr/bin/python3

import fcntl
import struct
import os
import time
from scapy.all import *

TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# create TUN interface
tun = os.open("/dev/net/tun", os.O_RDWR)
# ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
# change the name of the tun interface
ifr = struct.pack('16sH', b'zewen', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)

# get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface name : {}".format(ifname))
os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
while (True):
    packet = os.read(tun, 2048)
    if True:
        ip = IP(packet)
        ip.show()
        newip = IP(src='1.2.3.4', dst=ip.src)
        newpkt = newip / ip.payload
        os.write(tun, bytes(newpkt))

```

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

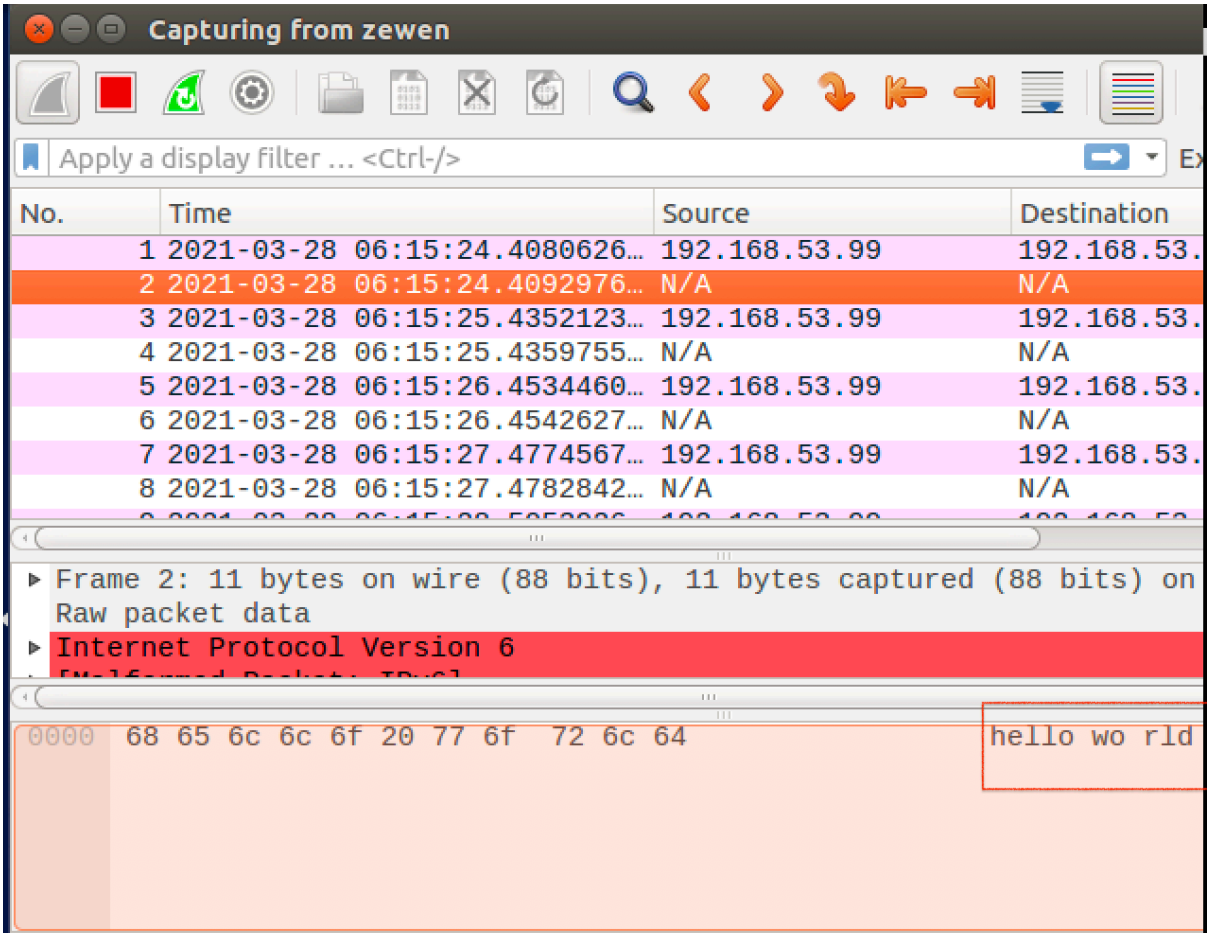


As you can see, the reply packet has the source ip address of 1.2.3.4.
Moreover, I also created a tun3.py file and add “hello world” as the data payload.

lab6 > tun3.py > ...

```python
#!/usr/bin/python3

import fcntl
import struct
import os
import time
from scapy.all import *

TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# create TUN interface
tun = os.open("/dev/net/tun", os.O_RDWR)
# ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
# change the name of the tun interface
ifr = struct.pack('16sH', b'zewen', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)

# get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface name : {}".format(ifname))
os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
while (True):
    packet = os.read(tun, 2048)
    if True:
        ip = IP(packet)
        ip.show()
        #newip = IP(src = '1.2.3.4',dst = ip.src)
        # #newpkt = newip/ip.payload
        # #os.write(tun,bytes(newpkt))
        os.write(tun, "hello world")

```

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 > ...

```python
from scapy.all import *
IP_A = "10.0.2.9"
PORT = 9090

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))

while True:
    data, (ip, port) = sock.recvfrom(2048)
    print("{}:{} --> {}:{}".format(ip, port, IP_A, PORT))
    pkt = IP(data)
    print("   Inside: {} ----> {}".format(pkt.src, pkt.dst))

```

```python
#!/usr/bin/python3

import fcntl
import struct
import os
import time
from scapy.all import *

TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# create TUN interface
tun = os.open("/dev/net/tun", os.O_RDWR)
# ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
# change the name of the tun interface
ifr = struct.pack('16sH', b'zewen', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)

# get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface name : {}".format(ifname))
os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

#  Create a UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
while (True):
    # get the packey from the tun interface
    packet = os.read(tun, 2048)
    if True:
        # Send packet through channel
        sock.sendto(packet, ("10.0.2.9", 9090))

```

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, tim
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
```

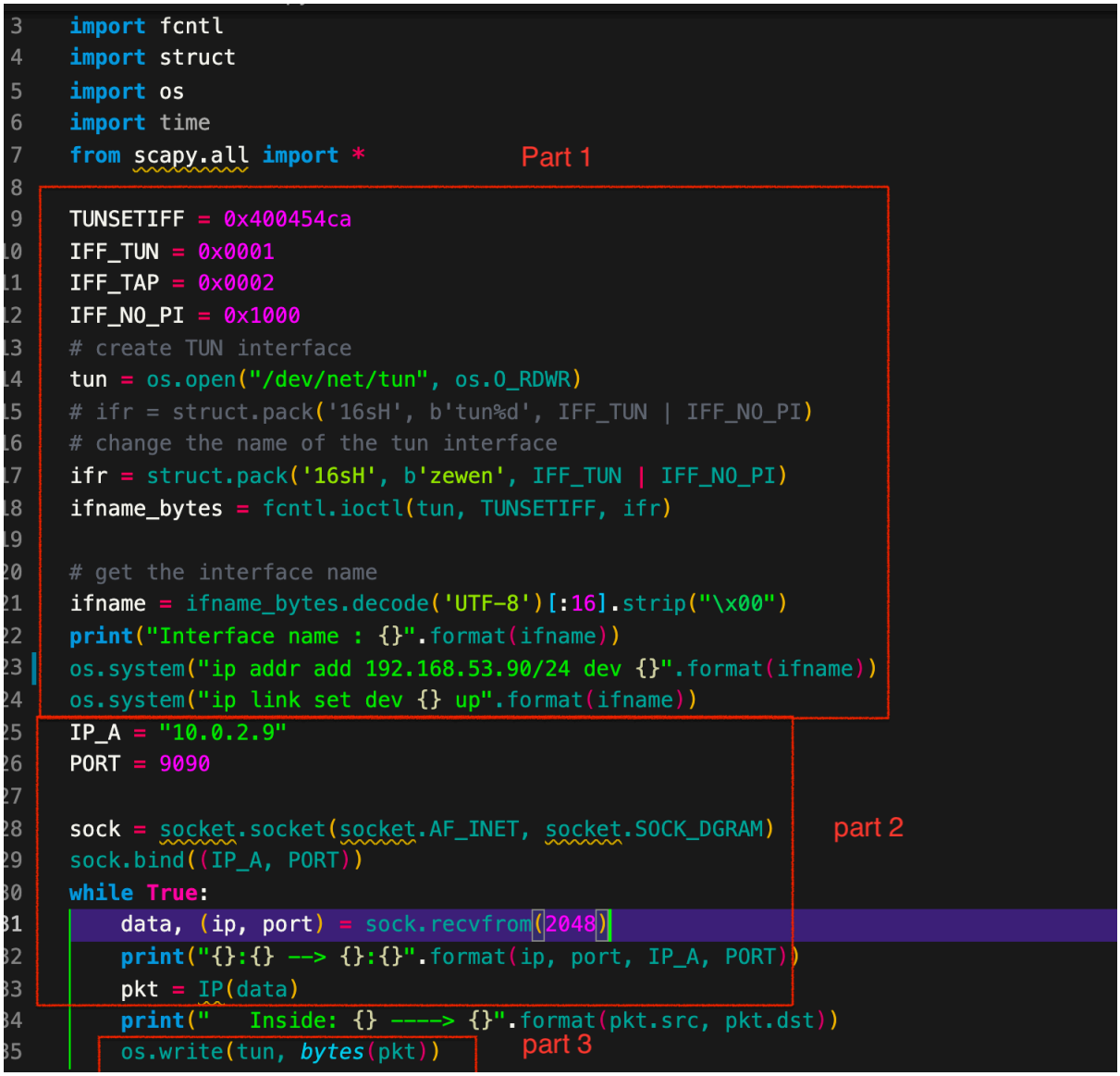
Since the 192.168.60.1 is not within the same subnet as the interface, we need to add the route to make sure all the packets going to the 192.168.60.0/24 network should be routed to the tun interface. To doing that, I use the command " sudo ip route add 192.168.60.0/24 dev zewen via 192.168.53.1" on 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
10.0.2.7:57057 --> 10.0.2.9:9090
   Inside: 192.168.53.99 ----> 192.168.60.101
10.0.2.7:57057 --> 10.0.2.9:9090
   Inside: 192.168.53.99 ----> 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.




```python
import fcntl
import struct
import os
import time
from scapy.all import *

TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# create TUN interface
tun = os.open("/dev/net/tun", os.O_RDWR)
# ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
# change the name of the tun interface
ifr = struct.pack('16sH', b'zewen', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)

# get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface name : {}".format(ifname))
os.system("ip addr add 192.168.53.90/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
IP_A = "10.0.2.9"
PORT = 9090

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))
while True:
    data, (ip, port) = sock.recvfrom(2048)
    print("{}:{} --> {}:{}".format(ip, port, IP_A, PORT))
    pkt = IP(data)
    print("    Inside: {} ----> {}".format(pkt.src, pkt.dst))
    os.write(tun, bytes(pkt))
```

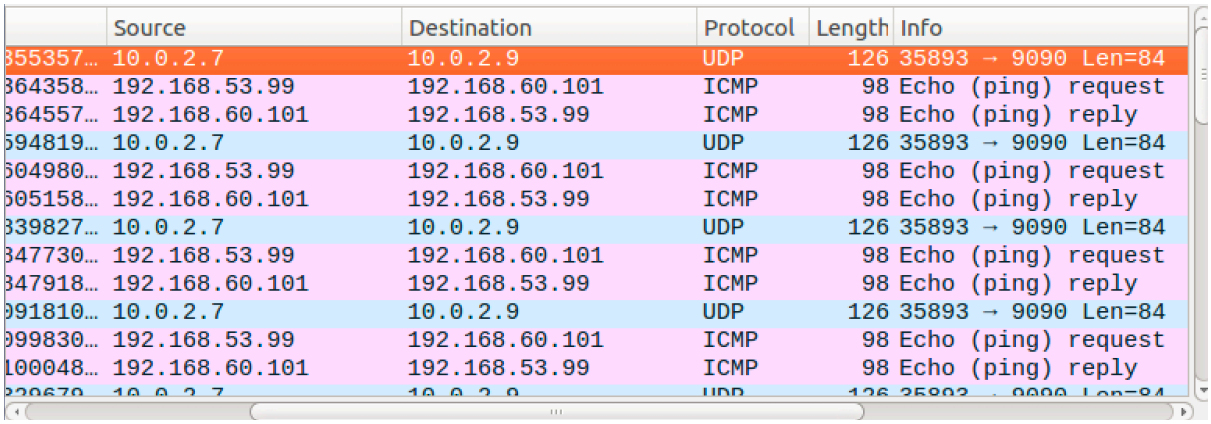
Moreover, on the client-side, I also run the tun_client_task_4.py to send all the packets which have the destination within the 192.168.60.0/24 network to the VPN server through the UDP socket.

```python

import fcntl
import struct
import os
import time
from scapy.all import *

TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# create TUN interface
tun = os.open("/dev/net/tun", os.O_RDWR)
# ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
# change the name of the tun interface
ifr = struct.pack('16sH', b'zewen', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)

# get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface name : {}".format(ifname))
os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
#IP_A = "10.0.2.7"
#PORT = 9090

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
#sock.bind((IP_A, PORT))
while True:
    packet = os.read(tun, 2048)
    if True:
        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.

```
[03/30/21]seed@VM:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Us
e Iface
0.0.0.0         192.168.60.1    0.0.0.0         UG    100    0
0 enp0s3
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0
0 enp0s3
192.168.60.0    0.0.0.0         255.255.255.0   U     100    0
0 enp0s3
[03/30/21]seed@VM:~$
```

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
604980…	192.168.53.99	192.168.60.101	ICMP	98	Echo (ping) request
605158…	192.168.60.101	192.168.53.99	ICMP	98	Echo (ping) reply
839827…	10.0.2.7	10.0.2.9	UDP	126	35893 → 9090 Len=84
847730…	192.168.53.99	192.168.60.101	ICMP	98	Echo (ping) request
847918…	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
329679…	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.

```python
import fcntl
import struct
import os
import time
from scapy.all import *

TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# create TUN interface
tun = os.open("/dev/net/tun", os.O_RDWR)
# ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
# change the name of the tun interface
ifr = struct.pack('16sH', b'zewen', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)

# get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface name : {}".format(ifname))
os.system("ip addr add 192.168.53.90/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
IP_A = "10.0.2.7"
PORT = 9090

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))
while True:
    #  this will block until at least one interface is ready
    ready, _, _ = select([sock, tun], [], [])
    for fd in ready:
        if fd is sock:
            data, (ip, port) = sock.recvfrom(2048)
            pkt = IP(data)
            print(" From socket <==: {} ----> {}".format(pkt.src, pkt.dst))
            os.write(tun, bytes(pkt))
        if fd is tun:
            packet = os.read(tun, 2048)
            pkt = IP(packet)
            print(" From tun <==: {} ----> {}".format(pkt.src, pkt.dst))
            sock.sendto(packet, ("10.0.2.9", 9090))
```

Client Ip and Port number

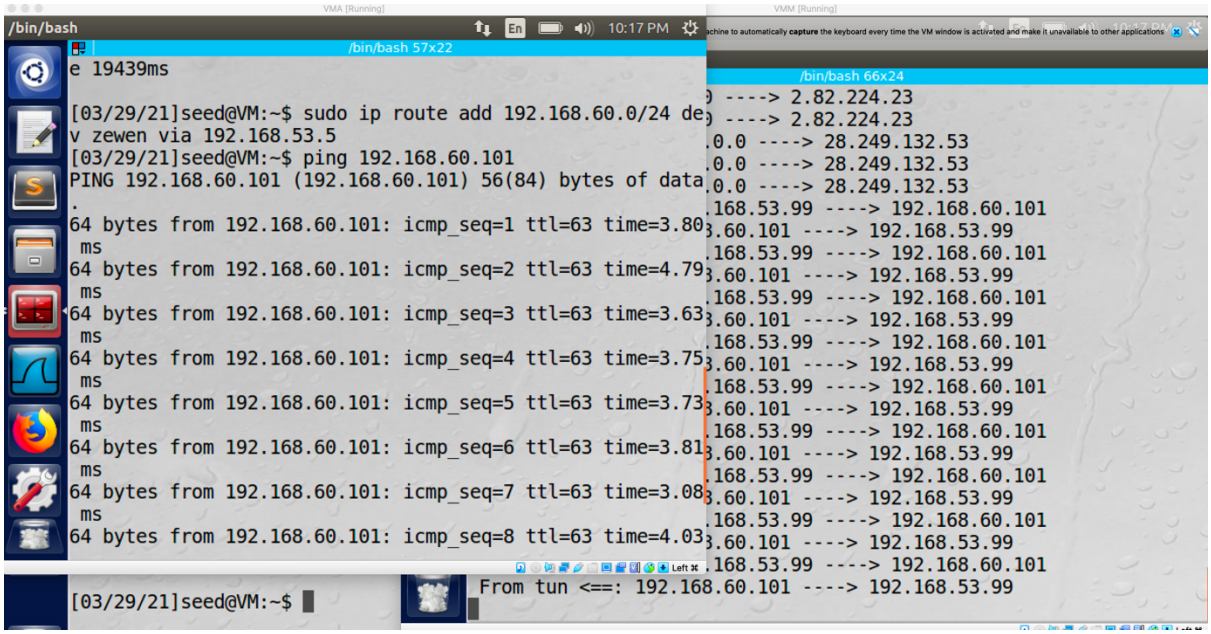
Sent to Server

```python

import fcntl
import struct
import os
import time
from scapy.all import *

TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# create TUN interface
tun = os.open("/dev/net/tun", os.O_RDWR)
# ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
# change the name of the tun interface
ifr = struct.pack('16sH', b'zewen', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)

# get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface name : {}".format(ifname))
os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
IP_A = "10.0.2.9"
PORT = 9090

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))
while True:
    #  this will block until at least one interface is ready
    ready, _, _ = select([sock, tun], [], [])
    for fd in ready:
        if fd is sock:
            data, (ip, port) = sock.recvfrom(2048)
            pkt = IP(data)
            print(" From socket <==: {} ----> {}".format(pkt.src, pkt.dst))
            os.write(tun, bytes(pkt))
        if fd is tun:
            packet = os.read(tun, 2048)
            pkt = IP(packet)
            print(" From tun <==: {} ----> {}".format(pkt.src, pkt.dst))
            sock.sendto(packet, ("10.0.2.7", 9090))
```

Server IP and port number

send to client

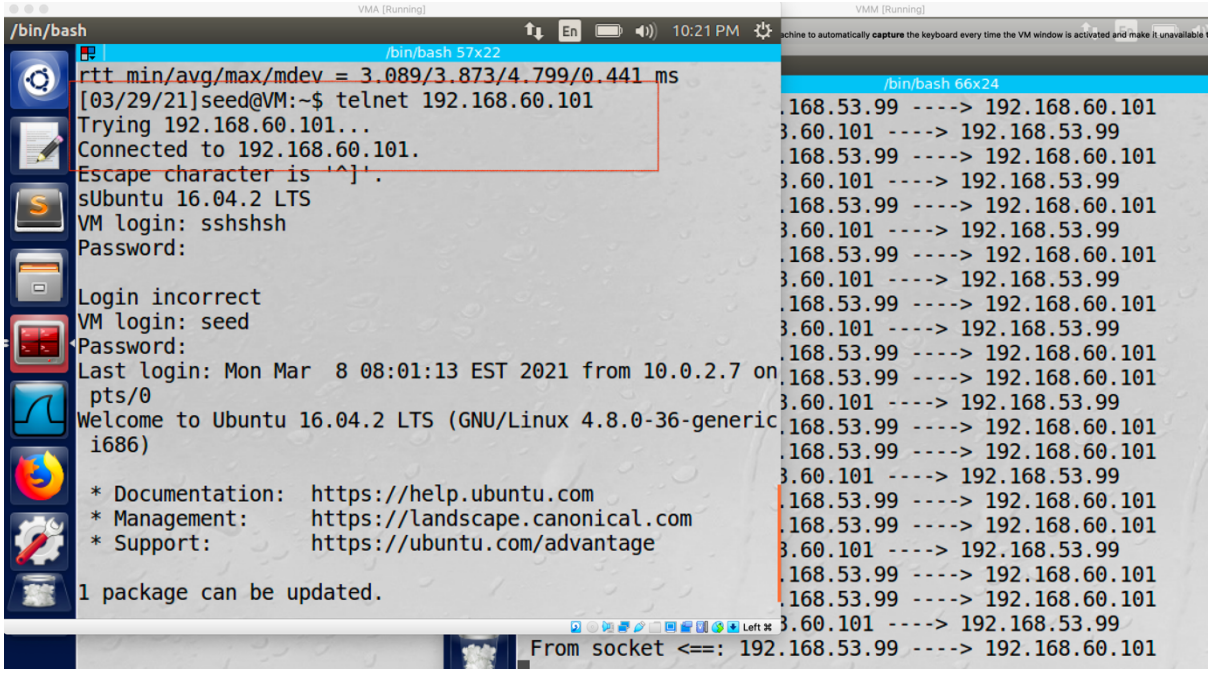
After I run these two python files at both server and client-side, I also add the same route which added at the client-side and host v side and enables the ip forwarding at the server-side. Then I ping host v at the client-side.



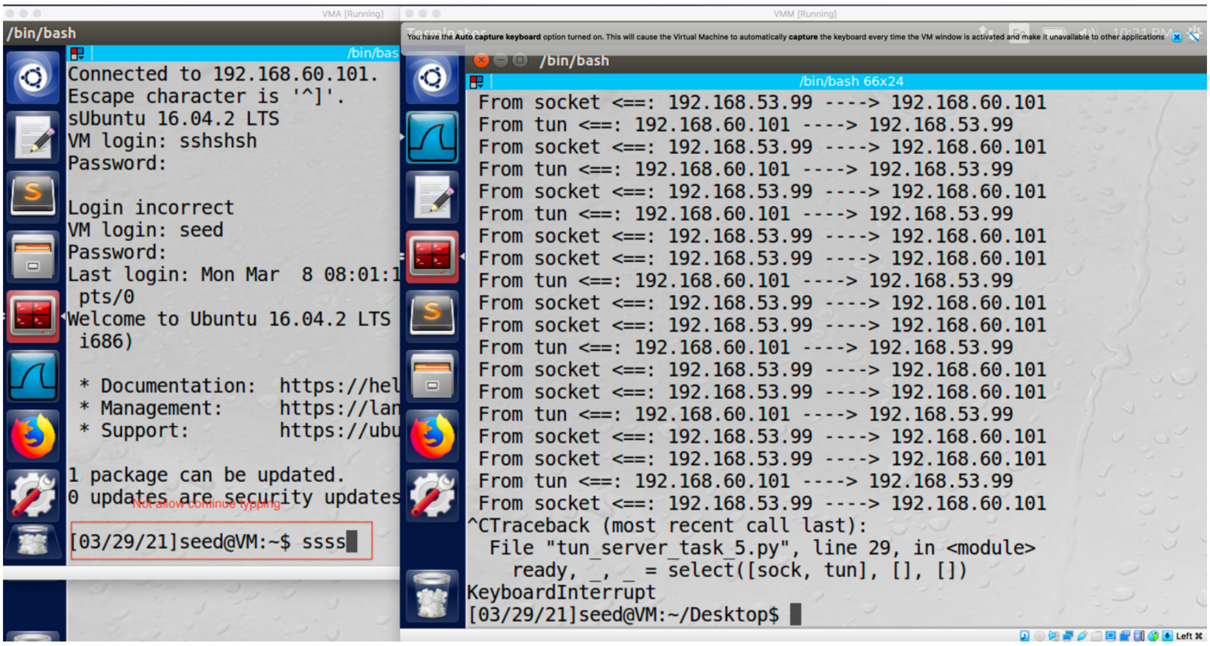
As you can see, I can ping to host v on the client-side and the VPN server is handling all the message traffic.

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.