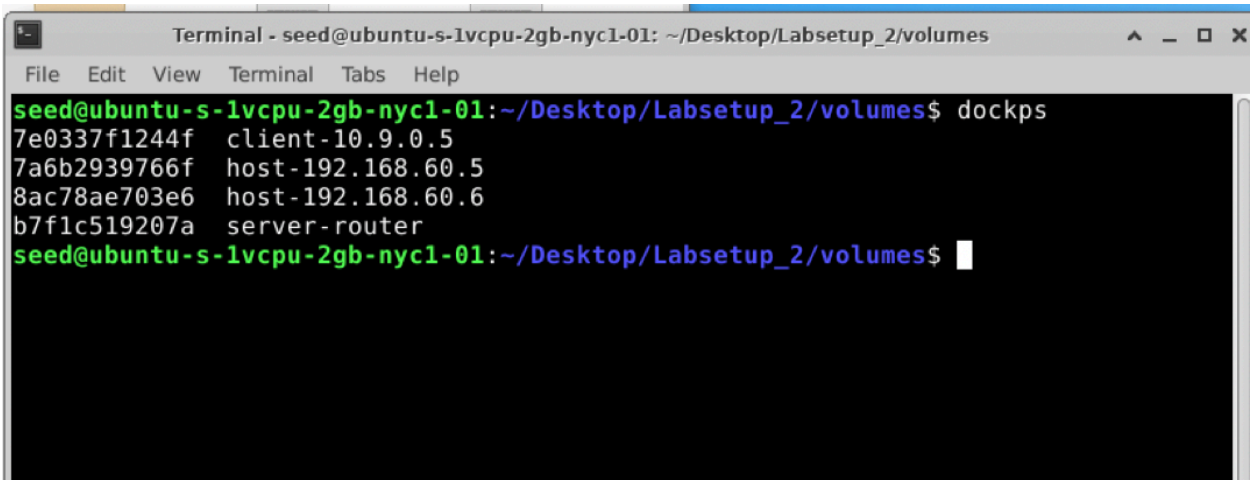
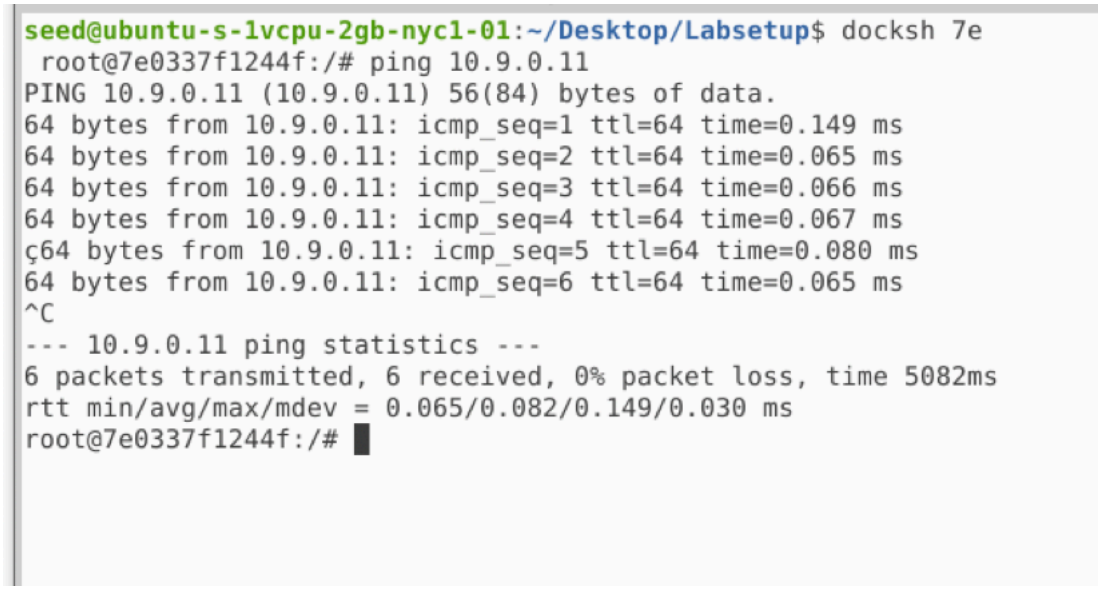
Lab2.0 VPN-TUNNELING



Task1
Host-U: host-192.168.60.5
Host-V: host-192.168.60.6

Testing:
Host U can communicate with VPN Server.



VPN Server can communicate with Host V

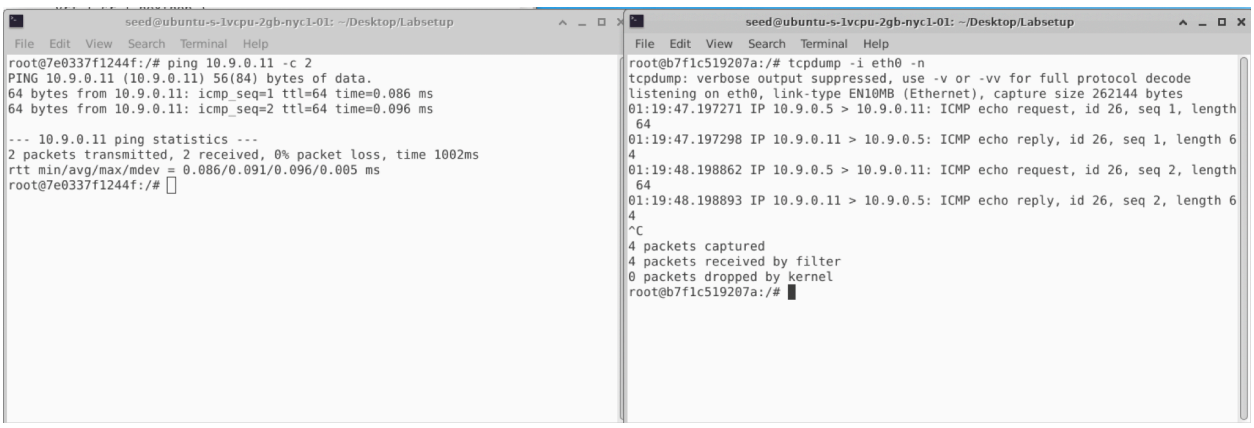
```
root@b7f1c519207a:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=64 time=0.160 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=64 time=0.077 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=64 time=0.072 ms
^C
--- 192.168.60.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 0.072/0.103/0.160/0.040 ms
```

Host U should not be able to communicate with Host V

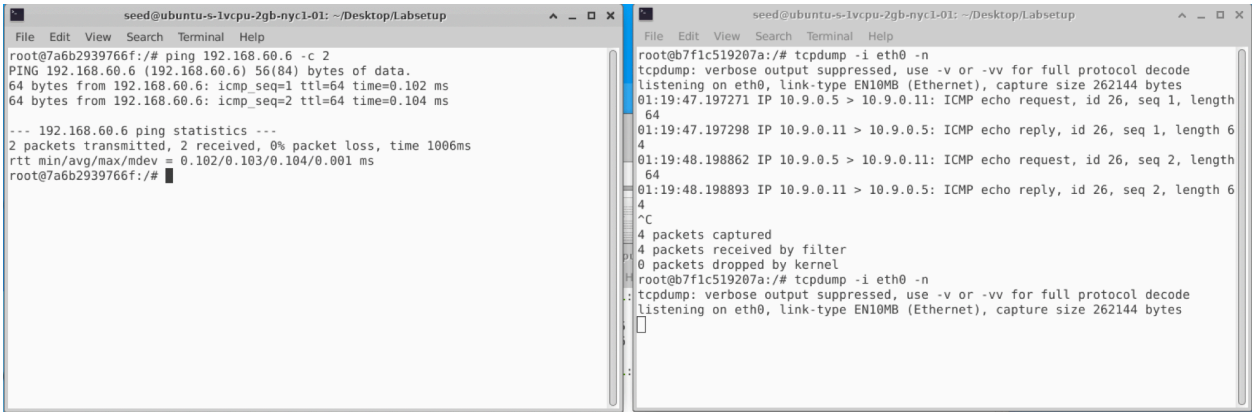
```
root@7e0337f1244f:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1004ms

root@7e0337f1244f:/#
```

Run tcpdump on the router, and sniff the traffic on each of the network. Show that you can capture packets

First ping the router from client(left), we can see the router can sniff the four packets(right)



Try to ping from one host to another host in the private net, we are not going to see any sniffed packets on the router(right). That's because the packets between the hosts do not go in or out of the router.



Task2

2.a

On Client machine, run tun.py by default first, and then run ip addr to find the interface called tun0.

```
3: tun0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group defa
ult qlen 500
    link/none
```

Modify tun.py to change the name of interface:(qiu)

```
#!/usr/bin/env 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 the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'qiu%d', 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))

while True:
    time.sleep(10)

~
```

Save and run. The name of the interface is changed as expected.

```
bash: vim: command not found
root@7e0337f1244f:/volumes# ./tun.py &
[1] 40
root@7e0337f1244f:/volumes# Interface Name: qiu0
```

2.b

First, modify tun.py as below:

```
#!/usr/bin/env 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 the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'qiu%d', 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:
   time.sleep(10)

~
~
~
~
~
```

Save and run tun.py again. We can see the IP address 192.168.53.99/24 is successfully assigned to it. If we did not add the two lines in the code. We can see from the screenshot from task 2.a there are no ip addresses attached to it.

```
root@7e0337f1244f:/volumes# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
    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
7: qiu0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state
 UNKNOWN group default qlen 500
    link/none
    inet 192.168.53.99/24 scope global qiu0
       valid_lft forever preferred_lft forever
31: eth0@if32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
 group default
    link/ether 02:42:0a:09:00:05 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.9.0.5/24 brd 10.9.0.255 scope global eth0
       valid_lft forever preferred_lft forever
root@7e0337f1244f:/volumes#
```

2.c
Modify tun,py as below:

```
#!/usr/bin/env 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 the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'qiu%d', 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:
#    time.sleep(10)
while True:
  # Get a packet from the tun interface
  packet = os.read(tun, 2048)
  if packet:
    ip = IP(packet)
    print("{}:".format(ifname),ip.summary())
~
~
~
~
```

Run tun.py again, and try to ping a host in 192.168.53.0/24. The IP packet is printed. Since we have attached the network 192.168.53.0/24 to the server-router, host U can ping the hosts in this network through the server. No reply is received because there is no such host exists.

```
root@7e0337f1244f:/volumes# ping 192.168.53.5 -c 2
PING 192.168.53.5 (192.168.53.5) 56(84) bytes of data.
qiu0: IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
qiu0: IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw

--- 192.168.53.5 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1020ms
```

Ping 192.168.60.5(host V):

```
root@7e0337f1244f:/volumes# ping 192.168.60.5 -c 2
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.

--- 192.168.60.5 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1023ms

root@7e0337f1244f:/volumes#
```

Nothing is printed. The packets from Host U try to sent to the internal network of the router, but it fails and the packets do not go through the router. Nothing is printed unless we specifiy the routing rules for the router.

2.d
Requirement 1
First, in tun.py, add the following code block.

```
    if ICMP in pkt and pkt[ICMP].type == 8:
        print("Original Packet.........")
        print("Source IP : ", pkt[IP].src)
        print("Destination IP :", pkt[IP].dst)

        # spoof an icmp echo reply packet
        # swap srcip and dstip
        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
        icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
        data = pkt[Raw].load
        newpkt = ip/icmp/data

        print("Spoofed Packet.........")
        print("Source IP : ", newpkt[IP].src)
        print("Destination IP :", newpkt[IP].dst)

    #    send(newpkt, verbose=0)

    #newip = IP(src='1.2.3.4', dst=ip.src)
    #newpkt = newip/ip.payload
        os.write(tun, bytes(newpkt))
~
```

Save and run tun.py, and then ping 192.168.53.5:

```
root@7e0337f1244f:/volumes# ping 192.168.53.5 -c 2
PING 192.168.53.5 (192.168.53.5) 56(84) bytes of data.
qiu0: IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
Original Packet.........
Source IP :  192.168.53.99
Destination IP : 192.168.53.5
Spoofed Packet.........
Source IP :  192.168.53.5
Destination IP : 192.168.53.99
64 bytes from 192.168.53.5: icmp_seq=1 ttl=64 time=11.0 ms
qiu0: IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
Original Packet.........
Source IP :  192.168.53.99
Destination IP : 192.168.53.5
Spoofed Packet.........
Source IP :  192.168.53.5
Destination IP : 192.168.53.99
64 bytes from 192.168.53.5: icmp_seq=2 ttl=64 time=2.23 ms

--- 192.168.53.5 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 2.227/6.635/11.043/4.408 ms
```

From the printed information above we can see that the in the ICMP reply Spoofed Packet has destination 192.168.53.5 and is sent back to its original Source IP address. This is done by ping 192.168.53.5 from host U. Since 192.168.53.0/24 is attached to the server-router, the server-router replies the request as expected.

Requirement 2
In tun.py, modify the os.write code as below

```
    #newip = IP(src='1.2.3.4', dst=ip.src)
    #newpkt = newip/ip.payload
        rbdata=b'Instead of writing an IP packet to the interface, write some ar
bitrary data to the interface, and report your observation.'
        os.write(tun, rbdata)
"tun.py" 58L, 1650C                                           57,122        Bot
```

Run tun.py and sniff the packet using tcpdump on interface qiu0:

```
root@7e0337f1244f:/volumes# ./tun.py &
[1] 175
root@7e0337f1244f:/volumes# Interface Name: qiu0
^C
root@7e0337f1244f:/volumes# tcpdump -i qiu0 -n 2>/dev/null &
[2] 183
root@7e0337f1244f:/volumes# ping 192.168.53.5 -c 1
PING 192.168.53.5 (192.168.53.5) 56(84) bytes of data.
03:42:58.693795 IP 192.168.53.99 > 192.168.53.5: ICMP echo request, id 184, seq
1, length 64
qiu0: IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
Original Packet.........
Source IP :  192.168.53.99
Destination IP : 192.168.53.5
Spoofed Packet.........
Source IP :  192.168.53.5
Destination IP : 192.168.53.99
03:42:58.699126 IP truncated-ip - 29434 bytes missing! 114.105.116.105 > 110.103
.32.97: ip-proto-102

--- 192.168.53.5 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

As suggested by the above message, the os.write function interprets the random data string as ip addresses. So it reports the random data as missing ip addresses.

Task3

In volumes folder, create tun_server.py as below:

```
#!/usr/bin/env python3

from scapy.all import *

IP_A = '0.0.0.0'
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))
~
```

Create tun_client.py as below:

```
#!/usr/bin/env python3
import fcntl
import struct
import os
import time
from scapy.all import *
# Create UDP socket
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
SERVER_IP, SERVER_PORT="10.9.0.11", 9090
TUNSETIFF = 0x400454ca
IFF_TUN   = 0x0001
IFF_TAP   = 0x0002
IFF_NO_PI = 0x1000

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'qiu%d', 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:
  # Get a packet from the tun interface
  packet = os.read(tun, 2048)
  if packet:
    # Send the packet via the tunnel
    sock.sendto(packet, (SERVER_IP, SERVER_PORT))
```

Run ./tun_client.py on Host U and ./tun_server.py on server_router, and on host U ping 192.168.53.0/24:

```
root@7e0337f1244f:/volumes# ./tun_client.py &
[1] 228
root@7e0337f1244f:/volumes# Interface Name: qiu0
^C
root@7e0337f1244f:/volumes# ping 192.168.53.5
PING 192.168.53.5 (192.168.53.5) 56(84) bytes of data.
^C
--- 192.168.53.5 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3054ms
```

On server_router we can see the printout:

```
root@b7f1c519207a:/volumes# ./tun_server.py &
[1] 60
root@b7f1c519207a:/volumes# 10.9.0.5:55624 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:55624 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:55624 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:55624 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.53.5
```

We can see these printout because we run tun_client.py tun_server.py. On tun_client.py we send UDP packet to the server(which hardcoded IP address as 10.9.0.11 and PORT as 9090). On server the tun_server.py prints out the information of the received packets.

Try to ping Host V:

```
root@7e0337f1244f:/volumes# ping 192.168.60.5 -c 2
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.

--- 192.168.60.5 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1004ms

root@7e0337f1244f:/volumes#
```

We can see no reply from Host V, and no output on the server-router. The connection to private network of VPN fails, because we have not yet set routing rules for the server-router.

On Host U, add the following routing rules and ping 192.168.60.0/24:

```
root@7e0337f1244f:/volumes# ip route add 192.168.60.0/24 dev qiu0
root@7e0337f1244f:/volumes# 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 ---
3 packets transmitted, 0 received, 100% packet loss, time 2031ms
```

On the server, we can see the following output. We can see we have sent ICMP packets to the private network. The output of 192.168.60.5 shows the ICMP packets are received by tun_server.py, which prints out the packet destination(192.168.60.5 in this case).

```
root@b7f1c519207a:/volumes# 10.9.0.5:55624 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:55624 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:55624 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.60.5
```

Task4

Modify tun_server.py as below:

1. Create a TUN interface and configure it

```python
#!/usr/bin/env 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 the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'qiu%d', 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))

```

Get the data from the socket interface; treat the received data as an IP packet(line 38) & Write the packet to the TUN interface(line 40):

```
34
35 while True:
36   data, (ip,port) = sock.recvfrom(2048)
37   print("{}:{} --> {}:{}".format(ip,port, IP_A, PORT))
38   pkt = IP(data)
39   print("  Inside: {} --> {}".format(pkt.src, pkt.dst))
40   os.write(tun,bytes(pkt))
-- INSERT --                                                    40,21
```

Add Routing on tun_client.py:

```
#routing
os.system('ip route add 192.168.60.0/24 dev {}'.format(ifname))
```

On Host U, run tun_client.py and on server_router, run tun_server.py. On Host U, ping 2 packets to Host V, we received no reply as expected.

```
root@7e0337f1244f:/volumes# ping 192.168.60.5 -c 2
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.

--- 192.168.60.5 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1023ms
```

Then on server, we see the following output:

```
root@b7f1c519207a:/volumes# ./tun_server.py &
[1] 52
root@b7f1c519207a:/volumes# Interface Name: qiu0
10.9.0.5:49180 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:49180 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.60.5
```

Inside the router, we can see the kernel sends the packets from 192.168.53.99(tun interface) to 192.168.60.5(Host V).
On Host V, we get the following output on tcpdump:

```
^C
root@7a6b2939766f:/# tcpdump -i eth0 -n 2>/dev/null
06:09:14.460839 IP 192.168.60.1.5353 > 224.0.0.251.5353: 0 [2q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
06:12:19.774514 ARP, Request who-has 192.168.60.5 tell 192.168.60.11, length 28
06:12:19.774541 ARP, Reply 192.168.60.5 is-at 02:42:c0:a8:3c:05, length 28
06:12:19.774559 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 118, seq 1, length 64
06:12:19.774575 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 118, seq 1, length 64
06:12:20.797541 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 118, seq 2, length 64
06:12:20.797586 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 118, seq 2, length 64
06:12:24.892377 ARP, Request who-has 192.168.60.11 tell 192.168.60.5, length 28
06:12:24.892647 ARP, Reply 192.168.60.11 is-at 02:42:c0:a8:3c:0b, length 28
```

We can see two ICMP requests and replies on host V. The packets are successfully sent to Host V and Host V tries to reply, but the reply is dropped somewhere and Host U cannot receive.

Task5
Modify the while loop of tun_server.py:

```python
ip,port='10.9.0.5',12345
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.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))
      # ... (code needs to be added by students) ...
      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))
      # ... (code needs to be added by students) ...
      sock.sendto(packet, (ip,port))
```

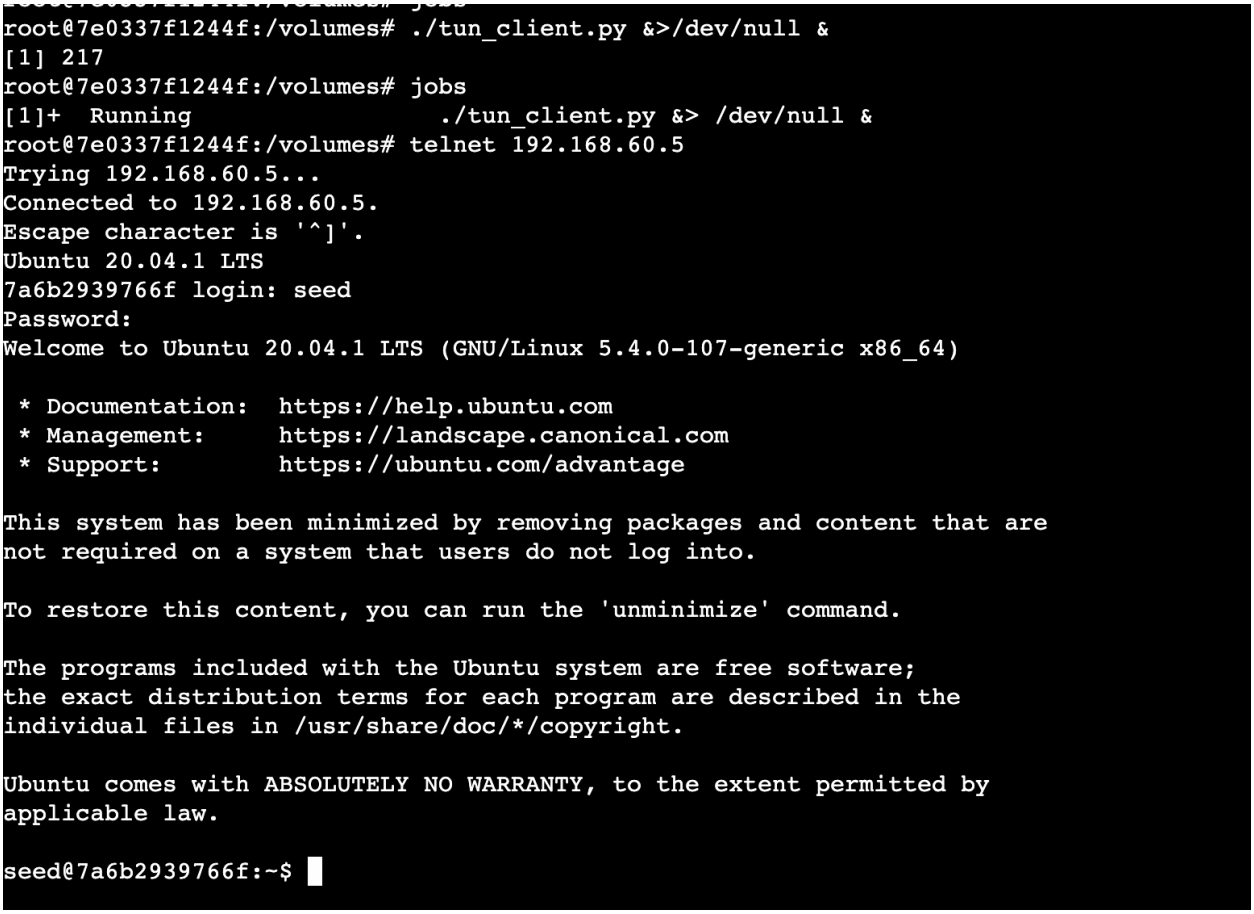
Modify while loop of tun_client.py as below:

```
while True:
  # this will block until at least one interface is ready
  ready,_,_ = select.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))
      # ... (code needs to be added by students) ...
      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))
      # ... (code needs to be added by students) ...
#       sock.sendto(packet, (ip,port))

    # Send the packet via the tunnel
      sock.sendto(packet, (SERVER_IP, SERVER_PORT))
```

Run tun_server.py on server_router and tun_client.py on Host U. Ping Host V(192.168.60.5) from Host U, we can see the following output:
We have received packets from Host V.

```
root@7e0337f1244f:/volumes# jobs
root@7e0337f1244f:/volumes# ./tun_client.py &
[1] 170
root@7e0337f1244f:/volumes# Interface Name: qiu0
^C
root@7e0337f1244f:/volumes# ping 192.168.60.5 -c 2
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket ==>: 192.168.60.5 --> 192.168.53.99
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=16.7 ms
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket ==>: 192.168.60.5 --> 192.168.53.99
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=4.31 ms

--- 192.168.60.5 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
```

Telnet to Host V from Host U: we can see the telnet is successful.



To see the packets flow, add the printing command in tun_client.py(line 40):

```
30        # print("hhhhh")
31 while True:
32   # this will block until at least one interface is ready
33   ready,_,_ = select.select([sock,tun],[],[])
34
35   for fd in ready:
36     if fd is sock:
37       data, (ip,port) = sock.recvfrom(2048)
38       print("From UDP {}:{}-->{}".format(ip,port,"10.9.0.5"))
39       pkt = IP(data)
40       print("From socket(IP) ==>: {} --> {}".format(pkt.src, pkt.dst))
41       # ... (code needs to be added by students) ...
42       os.write(tun,bytes(pkt) )
43     if fd is tun:
44       packet = os.read(tun,2048)
45       pkt = IP(packet)
46       print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))
47       # ... (code needs to be added by students) ...
48 #       sock.sendto(packet, (ip,port))
49
50     # Send the packet via the tunnel
51       sock.sendto(packet, (SERVER_IP, SERVER_PORT))
```

Add the printing command in tun_server.py(line 57)

```
36
37 #UDP server
38 IP_A = '0.0.0.0'
39 PORT = 9090
40
41
42 ip,port='10.9.0.5',12345
43 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
44 sock.bind((IP_A, PORT))
45
46 while True:
47   # this will block until at least one interface is ready
48   ready,_,_ = select.select([sock,tun],[],[])
49
50   for fd in ready:
51     if fd is sock:
52       #print("From {}:{}-->{}:{}".format(ip,port,IP_A,PORT))
53
54       data, (ip,port) = sock.recvfrom(2048)
55       print("From UDP {}:{}-->{}:{}".format(ip,port,IP_A,PORT))
56       pkt = IP(data)
57       print("From socket(IP) ==>: {} --> {}".format(pkt.src, pkt.dst))
58       # ... (code needs to be added by students) ...
59       os.write(tun,bytes(pkt) )
60     if fd is tun:
61       packet = os.read(tun,2048)
62       pkt = IP(packet)
:set number
```

Save and run them again, and ping Host V from Host U again.

```
root@7e0337f1244f:/volumes# jobs
root@7e0337f1244f:/volumes# ./tun_client.py &
[1] 184
root@7e0337f1244f:/volumes# Interface Name: qiu0
^C
root@7e0337f1244f:/volumes# ping 192.168.60.5 -c 2
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
From tun ==>: 192.168.53.99 --> 192.168.60.5
From UDP 10.9.0.11:9090-->10.9.0.5
From socket(IP) ==>: 192.168.60.5 --> 192.168.53.99
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=12.3 ms
From tun ==>: 192.168.53.99 --> 192.168.60.5
From UDP 10.9.0.11:9090-->10.9.0.5
From socket(IP) ==>: 192.168.60.5 --> 192.168.53.99
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=6.82 ms
```

On server the printing information is below:

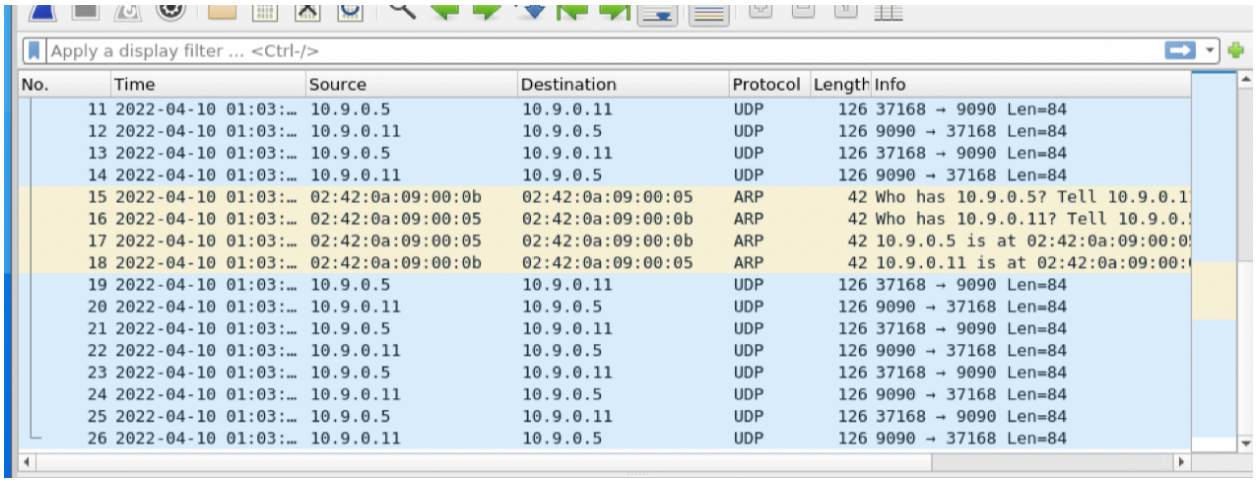
```
root@b7f1c519207a:/volumes# jobs
root@b7f1c519207a:/volumes# ./tun_server.py &
[1] 105
root@b7f1c519207a:/volumes# Interface Name: qiu0
From UDP 10.9.0.5:35765-->0.0.0.0:9090
From socket(IP) ==>: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
From UDP 10.9.0.5:35765-->0.0.0.0:9090
From socket(IP) ==>: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
```

Packet Flow Overview:

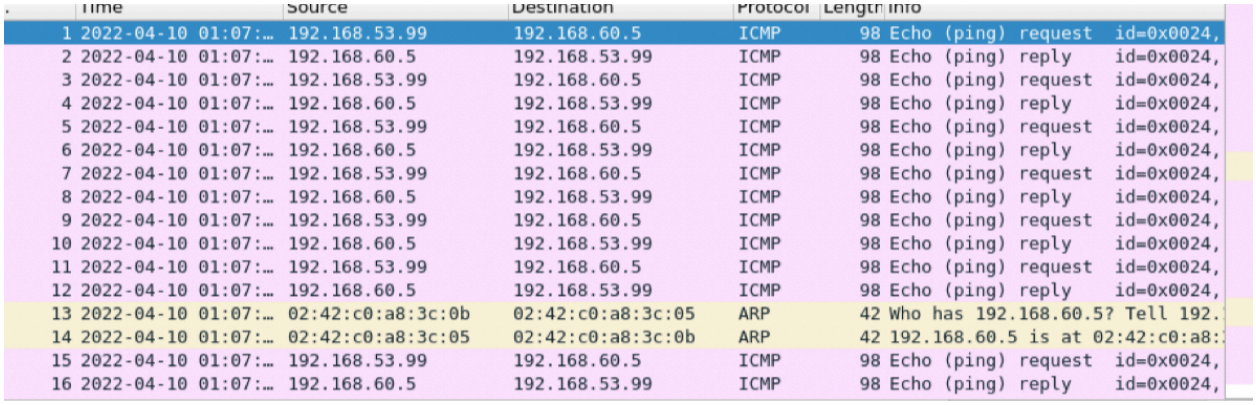
On client side, a tunnel is created and received packet first. It extracts the IP segment of the packet, prints it and sent the packet to server_ip and server port. Then the sock interface detects data input(reply from host V), prints the IP segment of the packet and writes it into its kernel.

On server side, the sock interface finds the data sent from client side. It extracts the IP segment and writes it into its kernel. Doing this successfully send the packet from client to its designed destination. Then the tun interface reads data sent from Host V. It prints out its information and sent it back to the hardcoded client ip and port.

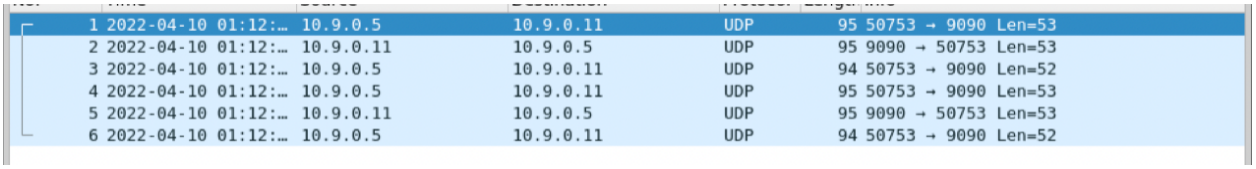
Wireshark:
Ping Host V from Host U: UDP packet sent from Host U to router_server.



ICMP request/reply between Host V and Host U.

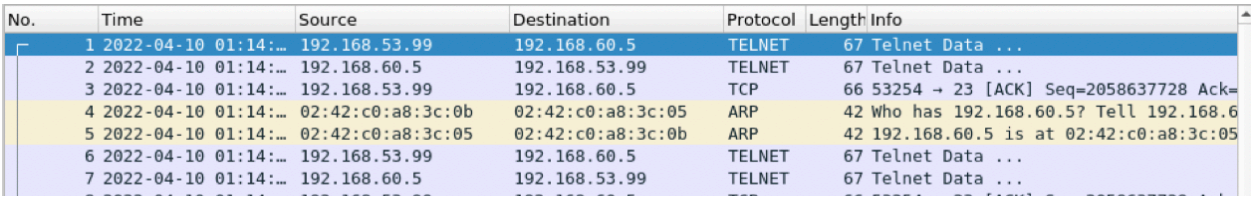


Telnet Host V from Host U. UDP packets sent by tun_client.py



1	2022-04-10 01:12:…	10.9.0.5	10.9.0.11	UDP	95	50753 → 9090 Len=53
2	2022-04-10 01:12:…	10.9.0.11	10.9.0.5	UDP	95	9090 → 50753 Len=53
3	2022-04-10 01:12:…	10.9.0.5	10.9.0.11	UDP	94	50753 → 9090 Len=52
4	2022-04-10 01:12:…	10.9.0.5	10.9.0.11	UDP	95	50753 → 9090 Len=53
5	2022-04-10 01:12:…	10.9.0.11	10.9.0.5	UDP	95	9090 → 50753 Len=53
6	2022-04-10 01:12:…	10.9.0.5	10.9.0.11	UDP	94	50753 → 9090 Len=52

After typing some characters on client, we can see the telnet is established between Host U and Host V.



No.	Time	Source	Destination	Protocol	Length	Info
1	2022-04-10 01:14:…	192.168.53.99	192.168.60.5	TELNET	67	Telnet Data ...
2	2022-04-10 01:14:…	192.168.60.5	192.168.53.99	TELNET	67	Telnet Data ...
3	2022-04-10 01:14:…	192.168.53.99	192.168.60.5	TCP	66	53254 → 23 [ACK] Seq=2058637728 Ack=
4	2022-04-10 01:14:…	02:42:c0:a8:3c:0b	02:42:c0:a8:3c:05	ARP	42	Who has 192.168.60.5? Tell 192.168.6
5	2022-04-10 01:14:…	02:42:c0:a8:3c:05	02:42:c0:a8:3c:0b	ARP	42	192.168.60.5 is at 02:42:c0:a8:3c:05
6	2022-04-10 01:14:…	192.168.53.99	192.168.60.5	TELNET	67	Telnet Data ...
7	2022-04-10 01:14:…	192.168.60.5	192.168.53.99	TELNET	67	Telnet Data ...

Task6

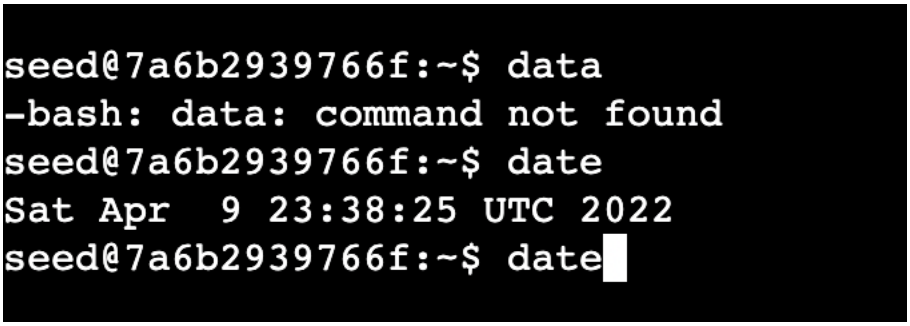
First, telnet to Host V from Host U. Try some commands:

```
root@7e0337f1244f:/volumes# ./tun_client.py &>/dev/null &
[1] 217
root@7e0337f1244f:/volumes# jobs
[1]+  Running                 ./tun_client.py &> /dev/null &
root@7e0337f1244f:/volumes# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
7a6b2939766f login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-107-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

seed@7a6b2939766f:~$ data
-bash: data: command not found
seed@7a6b2939766f:~$ date
Sat Apr  9 23:38:25 UTC 2022
seed@7a6b2939766f:~$
```

Stop tun_server.py:

```
From socket(IP) ==>: 192.168.53.99 --> 192.168.60.5
From UDP 10.9.0.5:38115-->0.0.0.0:9090
From socket(IP) ==>: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
From UDP 10.9.0.5:38115-->0.0.0.0:9090
From socket(IP) ==>: 192.168.53.99 --> 192.168.60.5
From UDP 10.9.0.5:38115-->0.0.0.0:9090
From socket(IP) ==>: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
From UDP 10.9.0.5:38115-->0.0.0.0:9090
From socket(IP) ==>: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
From UDP 10.9.0.5:38115-->0.0.0.0:9090
From socket(IP) ==>: 192.168.53.99 --> 192.168.60.5
^CTraceback (most recent call last):
  File "./tun_server.py", line 48, in <module>
    ready,_,_ = select.select([sock,tun],[],[])
KeyboardInterrupt

root@b7f1c519207a:/volumes#
```

On Host U we see nothing when we type. Running tun_server.py again, we can see the output we previously type again.



The TCP connection does not break. It keeps resending the packets, but since the server stopped, VPN client only gets error messages. The commands we typed are not lost. They are buffered, waiting to be sent to telnet server. When tun_server.py restarts again, the commands we typed before will reach telnet server due to TCP re-transmission. Telnet server will echo the commands back to the client side. That's why they will appear on the terminal again.