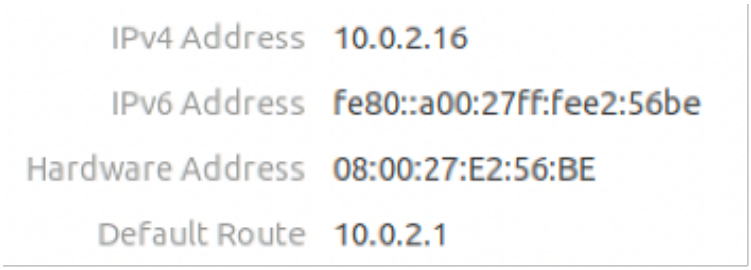
Lab3

CS690 Lijun Yu

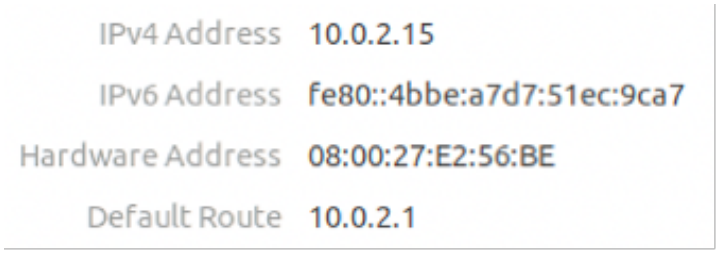
Task 1 Network Setup

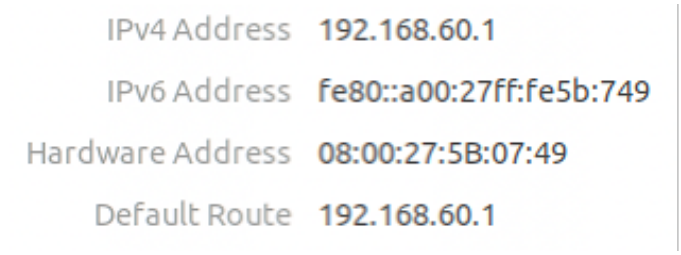
1. Prepare 3 Ubuntu20.04 VMs in virtual box

- hostu: 10.0.2.16

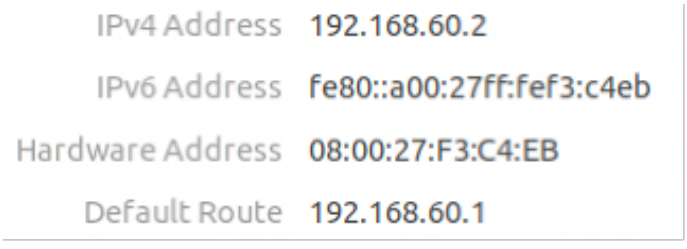


- vpn-server: 10.0.2.15, 192.168.60.1



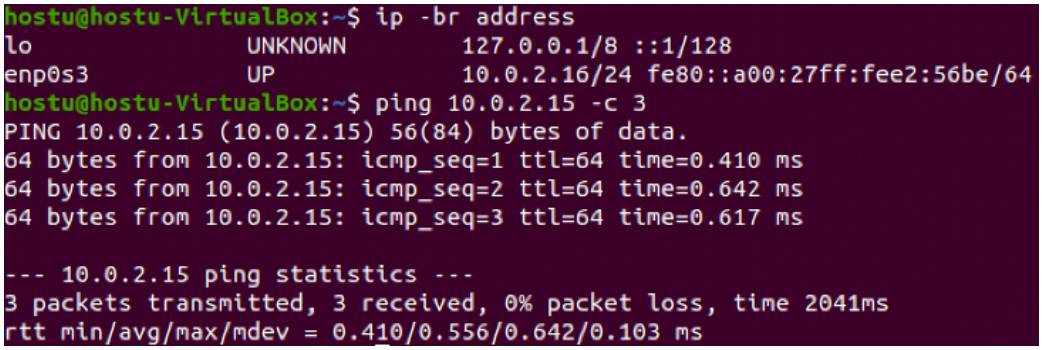


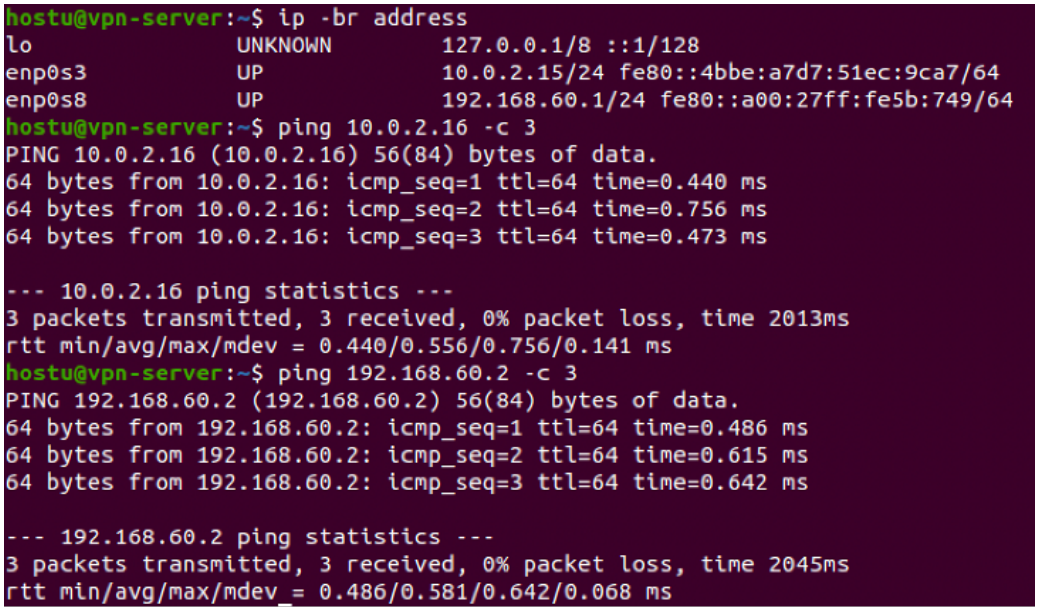
- hostv: 192.168.60.2

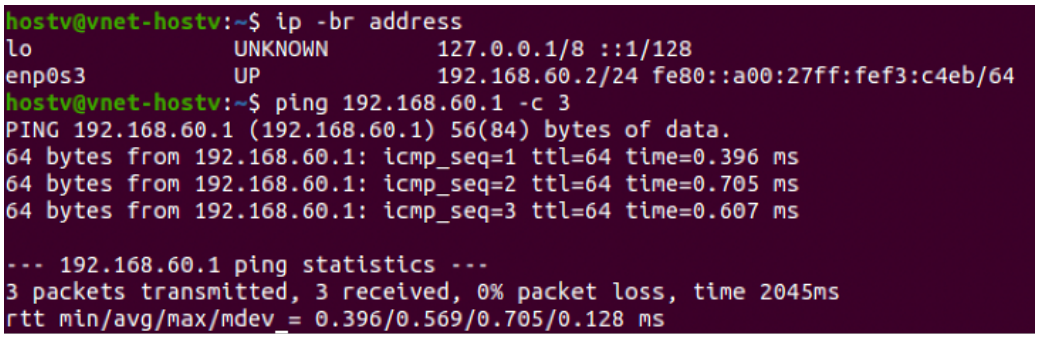


2. Verify the connectivity

Check the VMs' ip address through `ip -br address` and verify their connectivity between each other with by `ping`.







Task2 Create and Configure TUN Interface

Create a TUN interface by using the Python code.

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

2.a Change Name of the Interface

To change the prefix of the interface name, revise this line

```
ifr = struct.pack('16sH', b'YU%d', IFF_TUN | IFF_NO_PI)
```

Check the result and it showed the success.

```
hostu@hostu-VirtualBox:~/dev$ sudo ./tun.py
[sudo] password for hostu:
Interface Name: YU0
```

```
hostu@hostu-VirtualBox:~/dev$ ip -br address
lo               UNKNOWN        127.0.0.1/8 ::1/128
enp0s3           UP             10.0.2.16/24 fe80::a00:27ff:fee2:56be/64
YU0              DOWN
```

2.b Set up the Tun Interface

Adding the following code to the tun.py to assign an IP address to the interface and make the configuration be automatically performed by the program.

```
os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
```

Now run the `ip-br  address` again and the result is as followed.

```
hostu@hostu-VirtualBox:~/dev$ ip -br address
lo               UNKNOWN        127.0.0.1/8 ::1/128
enp0s3           UP             10.0.2.16/24 fe80::a00:27ff:fee2:56be/64
YU0              UNKNOWN        192.168.53.99/24 fe80::9da7:34df:a789:c9e5/64
```

The TUN interface is now assigned to 192.168.53.99/24.

2.c Read from thr TUN Interface

To cast the data received from the interface into a Scapy IP object, replace the codes in the `while` loop.

```
while True:
    # Get a packet from the tun interface
    packet = os.read(tun, 2048)
    if True:
        ip = IP(packet)
        print(ip.summary())
```

Run the revised tun.py and get the result.

```
ping 192.168.53.66
```

```
hostu@hostu-VirtualBox:~/dev$ sudo ./tun.py
Interface Name: YU0
0.0.0.0 > 211.11.205.204 128 frag:6911 / Padding
0.0.0.0 > 211.11.205.204 128 frag:6911 / Padding
0.0.0.0 > 211.11.205.204 128 frag:6911 / Padding
IP / ICMP 192.168.53.99 > 192.168.53.66 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.66 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.66 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.66 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.66 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.66 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.66 echo-request 0 / Raw
```

• On Host U, ping a host in the 192.168.53.0/24 network. What are printed out by the tun.py program? What has happened? Why?

> The printed out is shown as above:

IP / ICMP 192.168.53.99 > 192.168.53.66 echo-request 0 / Raw

The ping request was received by the tun interface and then piped into the program. Since there is nothing handling as reply, nothing happens.

```
hostu@hostu-VirtualBox:~/dev$ ping 192.168.53.66
PING 192.168.53.66 (192.168.53.66) 56(84) bytes of data.
From 10.18.208.1 icmp_seq=107 Packet filtered
^C
--- 192.168.53.66 ping statistics ---
118 packets transmitted, 0 received, +1 errors, 100% packet loss, time 119764ms
```

• On Host U, ping a host in the internal network 192.168.60.0/24, Does tun.py print out anything? Why?

```
ping 192.168.60.33
```

> As the figure shown, tun.py didn't print anything. This is because the CIDRs are different.

```
hostu@hostu-VirtualBox:~/dev$ sudo ./tun.py
Interface Name: YU0
0.0.0.0 > 157.191.83.186 128 frag:6911 / Padding
0.0.0.0 > 157.191.83.186 128 frag:6911 / Padding
0.0.0.0 > 157.191.83.186 128 frag:6911 / Padding
0.0.0.0 > 157.191.83.186 128 frag:6911 / Padding
0.0.0.0 > 157.191.83.186 128 frag:6911 / Padding
0.0.0.0 > 157.191.83.186 128 frag:6911 / Padding
```

```
hostu@hostu-VirtualBox:~/dev$ ping 192.168.60.33
PING 192.168.60.33 (192.168.60.33) 56(84) bytes of data.
From 10.18.208.1 icmp_seq=35 Packet filtered
From 10.18.208.1 icmp_seq=36 Packet filtered
^C
--- 192.168.60.33 ping statistics ---
45 packets transmitted, 0 received, +2 errors, 100% packet loss, time 44999ms
```

2.d Write to the TUN Interface

Construct a new packet based on the received packet and write it to the TUN interface.

First, verify if the data is ICMP every time we receive a new packet:

```
if ip.proto == 1 and ip[ICMP].type == 8:
    request_handle(ip)
```

Once verified, we'll handle the data. Define a funcition named `request_handle`. Since ping is also based on ICMP, we can customize the function to reply the ping request.

```
def request_handle:
    print("src IP:" + str(ip.src) + "; dst IP:" + str(ip.dst))

    newip = IP(src = ip.dst, dst = ip.src)
    newicmp = ICMP(type = "echo-reply", id = ip[ICMP].id, seq = ip[ICMP].seq)
    newpkt = newip/newicmp/ip[Raw].load
    os.write(tun,bytes(newpkt))
```

We can grab the source and destination IPs from the receiving packet, as well as the id of ICMP and the sequence number. Based on that, we can create the new packet to send back to the requester.

The result is shown below.

```
hostu@hostu-VirtualBox:~/dev$ sudo ./tun.py
Interface Name: YU0
src IP:192.168.53.99; dst IP:192.168.53.66
src IP:192.168.53.99; dst IP:192.168.53.66
src IP:192.168.53.99; dst IP:192.168.53.66
```

```
hostu@hostu-VirtualBox:~/dev$ ping 192.168.53.66 -c 3
PING 192.168.53.66 (192.168.53.66) 56(84) bytes of data.
64 bytes from 192.168.53.66: icmp_seq=1 ttl=64 time=1.77 ms
64 bytes from 192.168.53.66: icmp_seq=2 ttl=64 time=2.03 ms
64 bytes from 192.168.53.66: icmp_seq=3 ttl=64 time=1.98 ms

--- 192.168.53.66 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.766/1.925/2.027/0.114 ms
```

To write 'arbitrary data' instead of the IP packet, replace with the code:

```
os.write(tun, b'abcde')
```

This would lead to failure as it's no longer a standard IP package.

```
hostu@hostu-VirtualBox:~/dev$ ping 192.168.53.66 -c 3
PING 192.168.53.66 (192.168.53.66) 56(84) bytes of data.

--- 192.168.53.66 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2055ms
```

Task3 Send the IP Packet to VPN Server Through a Tunnel

Change the name of tun.py to tun_client.py and add somes codes to send data to the vpn server's port 9090.

```
# Create UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
SERVER_IP = "10.0.2.15"
SERVER_PORT = 9090

while True:
    # Get a packet from the tun interface
    packet = os.read(tun, 2048)
    # Send the packet via the tunnel
    sock.sendto(packet, (SERVER_IP, SERVER_PORT))
```

Then add a program on the vpn server to listen to port 9090 and print out whatever is received based on the code provided by the tutorial.

Try `ping 192.168.53.1`, and the vpn server would print:

```
hostu@vpn-server:~/dev$ sudo ./tun_server.py
[sudo] password for hostu:
10.0.2.16:33265 --> 0.0.0.0:9090
 Inside:192.168.53.99 --> 192.168.53.1
10.0.2.16:33265 --> 0.0.0.0:9090
 Inside:192.168.53.99 --> 192.168.53.1
10.0.2.16:33265 --> 0.0.0.0:9090
 Inside:192.168.53.99 --> 192.168.53.1
10.0.2.16:33265 --> 0.0.0.0:9090
 Inside:192.168.53.99 --> 192.168.53.1
10.0.2.16:33265 --> 0.0.0.0:9090
 Inside:192.168.53.99 --> 192.168.53.1
10.0.2.16:33265 --> 0.0.0.0:9090
 Inside:192.168.53.99 --> 192.168.53.1
```

This is because hostu sent UDP packet to vpn server's port 9090, and actually the tun(192.168.53.99) sent packet to the target address 192.168.53.1 as they are the same CIDR.

To let hosts inside `192.168.60.0/24` be accessed, add one line in the tun_client.py

```
os.system("sudo ip route add 192.168.60.0/24 dev{} via 192.168.53.99",format(ifname))
```

Test by `ping 192.168.60.66`:

```
hostu@vpn-server:~/dev$ sudo ./tun_server.py
[sudo] password for hostu:
10.0.2.16:36093 --> 0.0.0.0:9090
 Inside:192.168.53.99 --> 192.168.60.66
10.0.2.16:36093 --> 0.0.0.0:9090
 Inside:192.168.53.99 --> 192.168.60.66
10.0.2.16:36093 --> 0.0.0.0:9090
 Inside:192.168.53.99 --> 192.168.60.66
10.0.2.16:36093 --> 0.0.0.0:9090
 Inside:192.168.53.99 --> 192.168.60.66
10.0.2.16:36093 --> 0.0.0.0:9090
```

The ICMP packets are successfully received by the server through the tunnel. This is because the above code set the rule that all request toward 192.168.60.0/24 is handled by the tun interface(192.168.53.99).

Task4 Set up the VPN Server

Modify thet tun_server.py to create a TUN interface, get the data from the socket and write the packet to the TUN interface.

```
#!/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 the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'YU%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 192.168.53.66/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

# Get the data from the socket interface
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))
    # Write the pecket to the tun interface
    os.write(tun,bytes(pkt))
```

As the server tun's IP is `192.168.53.66`, we may need to enable the IP forwarding by

```
sudo sysctl net.ipv4.ip_forward=1
```

The packet transmission is reflected by wireshark as below, which implies ICMP packets have been successfully transmitted from Host U to HOST V (although the response sent by Host v is dropped).

```
192.168.60.2      192.168.53.99     ICMP     98 Echo (ping) reply    id=0x0020, seq=88/22528, ttl=64 (request in 58)
192.168.53.99     192.168.60.2      ICMP     98 Echo (ping) request  id=0x0020, seq=89/22784, ttl=63 (reply in 61)
192.168.60.2      192.168.53.99     ICMP     98 Echo (ping) reply    id=0x0020, seq=89/22784, ttl=64 (request in 60)
192.168.53.99     192.168.60.2      ICMP     98 Echo (ping) request  id=0x0020, seq=90/23040, ttl=63 (reply in 63)
192.168.60.2      192.168.53.99     ICMP     98 Echo (ping) reply    id=0x0020, seq=90/23040, ttl=64 (request in 62)
```

Task5 Handling Traffic in Both Directions

To make the tunnel two directional, for the client, create a UDP socket as the one in server. To prevent some distubance, set a `Black_List` to block unrelated packets.

```
# Avoid sending unexpected packets
BLACK_LIST = [ "34.122.121.32", "35.232.111.17" ]
```

And then modify the `while` loop.

```
# TUN client program
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)
            if str(pkt.dst) in BLACK_LIST or str(pkt_src) == "0.0.0.0":
                continue
            print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))
            sock.sendto(packet,(SERVER_IP, SERVER_PORT))
```

The above is code for the TUN client program. For the server program, just sendto (CLIENT_IP,CLIENT_PORT)).

Now both the server and client can receive the reply package.

```
From tun ==>: 192.168.60.2 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.2
From tun ==>: 192.168.60.2 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.2
From tun ==>: 192.168.60.2 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.2
From tun ==>: 192.168.60.2 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.2
From tun ==>: 192.168.60.2 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.2
From tun ==>: 192.168.60.2 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.2
From tun ==>: 192.168.60.2 --> 192.168.53.99
```

(vpn server)

```
From tun ==>: 192.168.53.99 --> 192.168.60.2
From socket <==: 192.168.60.2 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.2
From socket <==: 192.168.60.2 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.2
From socket <==: 192.168.60.2 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.2
From socket <==: 192.168.60.2 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.2
From socket <==: 192.168.60.2 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.2
From socket <==: 192.168.60.2 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.2
From socket <==: 192.168.60.2 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.2
```

(vpn client)

```
64 bytes from 192.168.60.2: icmp_seq=1313 ttl=63 time=6.47 ms
64 bytes from 192.168.60.2: icmp_seq=1314 ttl=63 time=4.11 ms
64 bytes from 192.168.60.2: icmp_seq=1315 ttl=63 time=3.72 ms
64 bytes from 192.168.60.2: icmp_seq=1316 ttl=63 time=4.68 ms
64 bytes from 192.168.60.2: icmp_seq=1317 ttl=63 time=33.8 ms
64 bytes from 192.168.60.2: icmp_seq=1318 ttl=63 time=3.97 ms
64 bytes from 192.168.60.2: icmp_seq=1319 ttl=63 time=2.80 ms
64 bytes from 192.168.60.2: icmp_seq=1320 ttl=63 time=3.80 ms
64 bytes from 192.168.60.2: icmp_seq=1321 ttl=63 time=6.86 ms
64 bytes from 192.168.60.2: icmp_seq=1322 ttl=63 time=3.89 ms
64 bytes from 192.168.60.2: icmp_seq=1323 ttl=63 time=3.93 ms
64 bytes from 192.168.60.2: icmp_seq=1324 ttl=63 time=3.90 ms
64 bytes from 192.168.60.2: icmp_seq=1325 ttl=63 time=29.8 ms
64 bytes from 192.168.60.2: icmp_seq=1326 ttl=63 time=5.84 ms
64 bytes from 192.168.60.2: icmp_seq=1327 ttl=63 time=26.7 ms
64 bytes from 192.168.60.2: icmp_seq=1328 ttl=63 time=27.2 ms
64 bytes from 192.168.60.2: icmp_seq=1329 ttl=63 time=23.7 ms
64 bytes from 192.168.60.2: icmp_seq=1330 ttl=63 time=23.6 ms
64 bytes from 192.168.60.2: icmp_seq=1331 ttl=63 time=23.0 ms
64 bytes from 192.168.60.2: icmp_seq=1332 ttl=63 time=23.5 ms
64 bytes from 192.168.60.2: icmp_seq=1333 ttl=63 time=19.8 ms
64 bytes from 192.168.60.2: icmp_seq=1334 ttl=63 time=17.5 ms
64 bytes from 192.168.60.2: icmp_seq=1335 ttl=63 time=3.30 ms
64 bytes from 192.168.60.2: icmp_seq=1336 ttl=63 time=18.2 ms
64 bytes from 192.168.60.2: icmp_seq=1337 ttl=63 time=17.0 ms
64 bytes from 192.168.60.2: icmp_seq=1338 ttl=63 time=3.04 ms
```

(ping result)

Task6 Tunnel-Breaking Experiment

First set up the Telnet connection between Host U and Host V.

```
hostu@hostu-VirtualBox:~/dev$ telnet 192.168.60.2
Trying 192.168.60.2...
Connected to 192.168.60.2.
Escape character is '^]'.

Ubuntu 20.04.3 LTS

vnet-hostv login: hostv
Password:
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.13.0-30-generic x86_64)

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

187 updates can be applied immediately.
100 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Wed Feb 23 19:38:10 EST 2022 on pts/1
hostv@vnet-hostv:~$ ls
```

Make a ping request to hold the connection.

```
hostv@vnet-hostv:~$ 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=46.9 ms
64 bytes from 192.168.60.1: icmp_seq=2 ttl=64 time=44.3 ms
64 bytes from 192.168.60.1: icmp_seq=3 ttl=64 time=2.47 ms
64 bytes from 192.168.60.1: icmp_seq=4 ttl=64 time=0.628 ms
```

Now shut down vpn server to break the tunnel. The Telnet connection is stuck. No new output of the ping request, and the ^c command is sent but not triggered.

```
64 bytes from 192.168.60.1: icmp_seq=23 ttl=64 time=1.95 ms
^C
```

Then reconnect the tunnel by restarting the vpn server. The Telnet connection work again. The ^c command is triggered.

```
^C
--- 192.168.60.1 ping statistics ---
44 packets transmitted, 44 received, 0% packet loss, time 44015ms
rtt min/avg/max/mdev = 0.301/5.152/102.786/17.591 ms
```

This happens similarly when setting a TCP connection over SSH.

The reason is that after the break of the connection, the Telnet client didn't receive the replay for the first input, so Telnet would hold the inputs and continuously retransmit the packet for the first input. After the vpn is reconnected, Telnet client receive the reply for the first input, so it would continue to transmit the packets for the held input. That is the similar mechanism to SSH.

Task7 Reset Default Route for Host V

First do the ops:

```
sudo ip route del 0.0.0.0/0
sudo ip route add default dev enp0s3 via 192.168.60.1
```

Then verify the status:

```
hostu@vnet-hostv:~$ ip route list
default via 192.168.60.1 dev enp0s3 proto static metric 20104
169.254.0.0/16 dev enp0s3 scope link metric 1000
192.168.60.0/24 dev enp0s3 proto kernel scope link src 192.168.60.2 metric 104
```

Task8 Experiment with the TUN IP address

Change the client TUN's interface to `192.168.30.99`, the request packets get to the virtual network interface at the server side.

```
     9 7.167966161   192.168.30.99   192.168.60.2   ICMP   84 Echo (ping) request  id=0x002c, seq=7430/1565, tt
    10 8.191612636   192.168.30.99   192.168.60.2   ICMP   84 Echo (ping) request  id=0x002c, seq=7431/1821, tt
    11 9.215605814   192.168.30.99   192.168.60.2   ICMP   84 Echo (ping) request  id=0x002c, seq=7432/2077, tt
    12 10.239509405  192.168.30.99   192.168.60.2   ICMP   84 Echo (ping) request  id=0x002c, seq=7433/2333, tt
    13 11.263766473  192.168.30.99   192.168.60.2   ICMP   84 Echo (ping) request  id=0x002c, seq=7434/2589, tt
    14 12.287648078  192.168.30.99   192.168.60.2   ICMP   84 Echo (ping) request  id=0x002c, seq=7435/2845, tt
    15 13.311682590  192.168.30.99   192.168.60.2   ICMP   84 Echo (ping) request  id=0x002c, seq=7436/3101, tt
```

The vpn server got the ping request from Host U, but it didn't sent it to host V, instead the packet was dropped at the server TUN interface.

The reason is that the server TUN interface would check the packet source, and found `192.168.30.99` is unreachable. This break the rule of rp_filter.

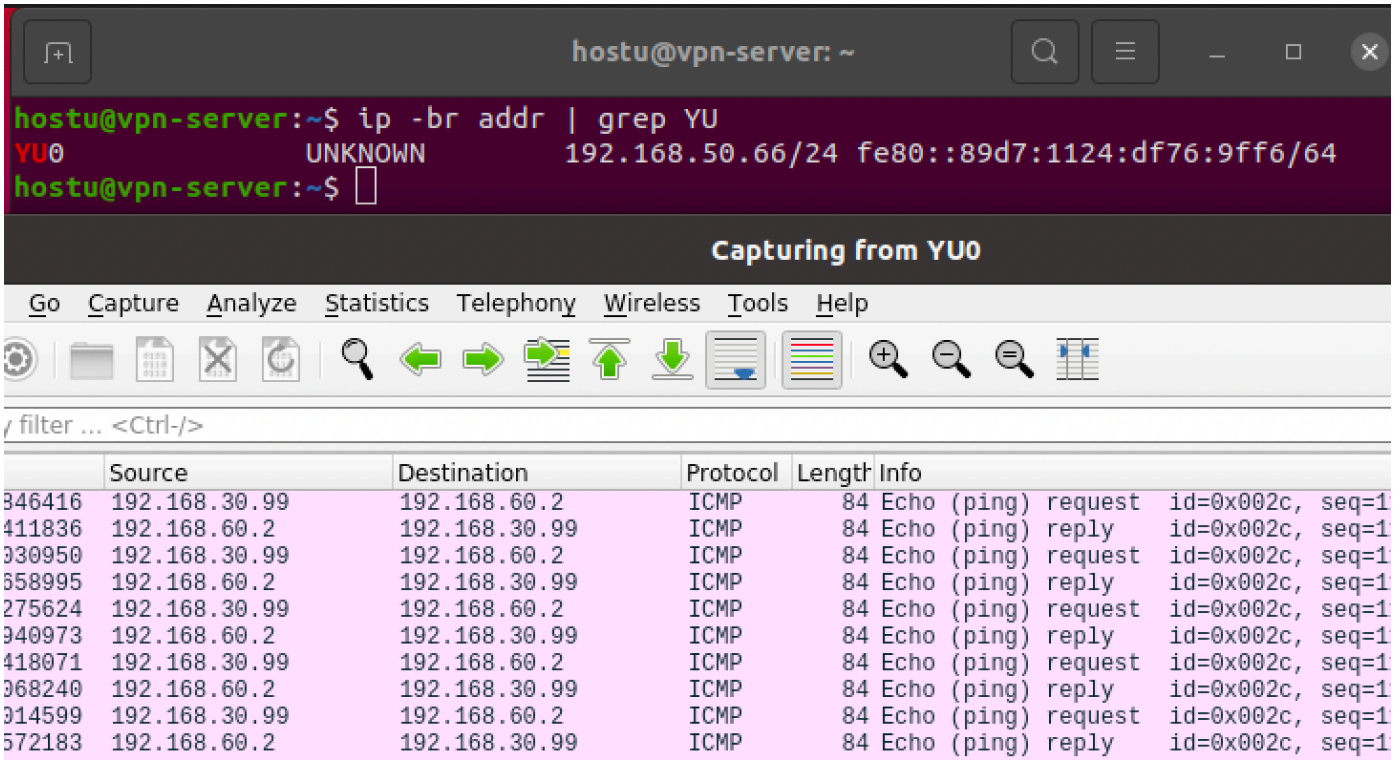
To solve the problem, enable the rp_filter on the server TUN interface:

```
os.system("sudo sysctl -w net.ipv4.conf.{}.rp_filter=0".format(ifname))
```

Then in the tun_server.py, set up the route:

```
os.system("sudo ip route add 192.168.30.0/24 dev{}".format(ifname))
```

Now the request packet can be allowed to be forward to the destination.



Task9 Experiment with the TAP Interface

Modify the client to change it to a TAP interface.

```
while True:
  packet = os.read(tap, 2048)
  if True:
    print("--------------------------")
    ether = Ether(packet)
    print(ether.summary())

    # Send a spoofed ARP response
    if ARP in ether and ether[ARP].op == 1:
      arp = ether[ARP]
      newether = Ether(src = ether.dst, dst = ether.src)
      newarp = ARP(op = 2, hwsrc = arp.hwdst, hwdst = arp.hwsrc, psrc = arp.pdst, pdst
= arp.psrc)
      newpkt = newether/newarp
      print("****** Fake response: {}".format(newpkt.symmary()))
      os.write(tap, bytes(newpkt))
```

Try the ping request,

```
Ether / IP / UDP / DNS Qry "b'_ipps._tcp.local.'"
Ether / IPv6 / UDP / DNS Qry "b'_ipps._tcp.local.'"
Ether / IPv6 / ICMPv6ND_RS / ICMPv6 Neighbor Discovery Option - Source Link-Lay
er Address 7a:20:dc:d7:39:11
```

The ping request won't get a response.

Then change the `while` loop to send a spoof ARP response and test with

```
arping -I YU0 192.168.53.66
```

```
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  2.480ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  1.744ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  2.151ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  2.464ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  2.315ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  4.310ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  2.521ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  2.470ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  2.450ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  2.426ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  2.668ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  2.055ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  2.073ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  2.932ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  2.571ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  2.410ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  2.121ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  2.340ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  2.442ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  2.612ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  2.233ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  2.403ms
Unicast reply from 192.168.53.66 [FF:FF:FF:FF:FF:FF]  3.162ms
```

Then, test another fake ip:

```
arping -I YU0 1.2.3.4
```

```
hostu@hostu-VirtualBox:~/dev$ arping -I YU0 1.2.3.4
ARPING 1.2.3.4 from 192.168.30.99 YU0
Unicast reply from 1.2.3.4 [FF:FF:FF:FF:FF:FF]  1.959ms
Unicast reply from 1.2.3.4 [FF:FF:FF:FF:FF:FF]  2.409ms
Unicast reply from 1.2.3.4 [FF:FF:FF:FF:FF:FF]  2.189ms
Unicast reply from 1.2.3.4 [FF:FF:FF:FF:FF:FF]  2.251ms
```

Because we made the fake response:

```
----------------------------
Ether / ARP who has 1.2.3.4 says 192.168.30.99
***** Fake response: Ether / ARP is at ff:ff:ff:ff:ff:ff says 1.2.3.4
----------------------------
```