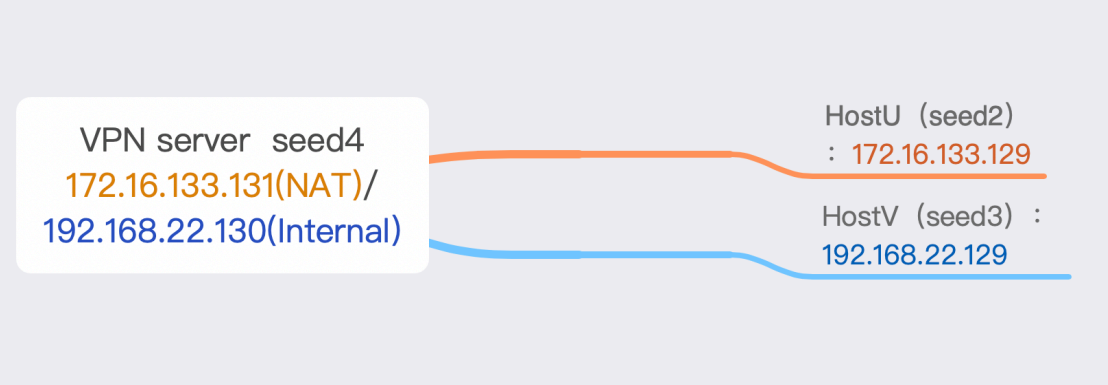
VPN Tunneling Lab

> *18307130089 吴嘉琪*

Task 1: Network Setup

网络配置:



Task 2: Create and Configure TUN Interface

Task 2.a: Name of the Interface

hostU 运行tun.py:

```
[12/10/20]seed@VM:~/.../VPN tun$ sudo ./tun.py
Interface Name: tun0
```

运行命令:

```
[12/10/20]seed@VM:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t 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: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKN
OWN group default qlen 1000
    link/ether 00:0c:29:77:a7:ee brd ff:ff:ff:ff:ff:ff
    inet 172.16.133.129/24 brd 172.16.133.255 scope global dynamic ens33
       valid_lft 1371sec preferred_lft 1371sec
    inet6 fe80::4361:7410:27e:a7e7/64 scope link
       valid_lft forever preferred_lft forever
5: tun0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group defa
ult qlen 500
    link/none
```

打印出了地址信息；

发现除了lo 和 ens33这两个原来的interface，tun.py创造出了一个新的interface：tun0

修改代码，修改tun的名字：

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

```
[12/10/20]seed@VM:~/.../VPN tun$ sudo ./tun.py
Interface Name: wu0
```

Task 2.b: Set up the TUN Interface

添加配置之后，再次运行ip address：

```
[12/10/20]seed@VM:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t 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: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKN
OWN group default qlen 1000
    link/ether 00:0c:29:77:a7:ee brd ff:ff:ff:ff:ff:ff
    inet 172.16.133.129/24 brd 172.16.133.255 scope global dynamic ens33
       valid_lft 1500sec preferred_lft 1500sec
    inet6 fe80::4361:7410:27e:a7e7/64 scope link
       valid_lft forever preferred_lft forever
7: wu0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast stat
e UNKNOWN group default qlen 500
    link/none
    inet 192.168.53.99/24 scope global wu0
       valid_lft forever preferred_lft forever
    inet6 fe80::cf1f:dfe0:7ebe:b7c4/64 scope link flags 800
       valid_lft forever preferred_lft forever
```

发现interface wu0 之后输出了相关的配置信息，说明ip被成功配置而且interface被唤醒。

Task 2.c: Read from the TUN Interface

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?

tun.py program输出信息：

```
0.0.0.0 > 255.245.220.153 128 frag:6911 / Padding
0.0.0.0 > 255.245.220.153 128 frag:6911 / Padding
IP / ICMP 192.168.53.99 > 192.168.53.2 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.2 echo-request 0 / Raw
0.0.0.0 > 255.245.220.153 128 frag:6911 / Padding
IP / ICMP 192.168.53.99 > 192.168.53.2 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.2 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.2 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.2 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.2 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.2 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.2 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.2 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.2 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.2 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.2 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.2 echo-request 0 / Raw
```

同时ping没有得到任何回应；

当结束tun.py的运行后，ping消息才得到了回应

```
64 bytes from 192.168.53.2: icmp_seq=68 ttl=128 time=5.20 ms
64 bytes from 192.168.53.2: icmp_seq=69 ttl=128 time=4.78 ms
64 bytes from 192.168.53.2: icmp_seq=70 ttl=128 time=10.3 ms
64 bytes from 192.168.53.2: icmp_seq=71 ttl=128 time=10.1 ms
64 bytes from 192.168.53.2: icmp_seq=72 ttl=128 time=10.0 ms
64 bytes from 192.168.53.2: icmp_seq=73 ttl=128 time=10.1 ms
64 bytes from 192.168.53.2: icmp_seq=74 ttl=128 time=12.1 ms
```

On Host U, ping a host in the internal network 192.168.22.0/24, Does tun.py print out anything? Why?

```
[12/10/20]seed@VM:~/.../VPN tun$ sudo ./tun.py
Interface Name: wu0
0.0.0.0 > 201.72.242.244 128 frag:6911 / Padding
0.0.0.0 > 201.72.242.244 128 frag:6911 / Padding
0.0.0.0 > 201.72.242.244 128 frag:6911 / Padding
```

tun.py没有输出任何信息；因为internal网络与NAT网络相互隔断，模拟了user无法访问某些特定网站的情况，所以无法ping通；而且router没有将目的ip为192.168.22.0/24的包转发到wu0 interface, 因此不会被tun读取并打印；

Task 2.d: Write to the TUN Interface

Please modify the tun.py code according to the following requirements:

- AftergettingapacketfromtheTUNinterface,ifthispacketisanICMPechorequestpacket,construct a corresponding echo reply packet and write it to the TUN interface. Please provide evidence to show that the code works as expected.

  修改tun.py，捕获packet的icmp部分，检测如果type=8，（为ping request）就构造type=0（reply）的spoof包写入tun

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

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

print("Interface Name: {}".format(ifname))

while True:
# Get a packet from the tun interface
  packet = os.read(tun, 2048)
  #print('here 1')
  if True:
    #print('here 2')
    ip = IP(packet)

    #print('here 3')
    print('go in:')
    print(ip.summary())

    #print(icmp)
    #print(icmp.type)

    if ICMP in ip and ip[ICMP].type is 8:
      print('this is request')

      icmp=ip[ICMP]
      print(icmp.type)
      newip = IP(src=ip.dst, dst=ip.src)
      newicmp=ICMP(type=0,id=icmp.id,seq=icmp.seq)
      data=ip[Raw].load
      newpkt = newip/newicmp/data

      print('go out')
      print(newip.summary)
      os.write(tun, bytes(newpkt))

    #Send out a spoof packet using the tun

    # newip = IP(src=ip.dst, dst=ip.src)
    # newpkt = newip/ip.payload
    # print('go out:')
    # print(newpkt.summary())
```

```
        # os.write(tun, bytes(newpkt))
```

ping 192.168.53.2之后和以前没有回应不同，确实输出了信息：

```
[12/14/20]seed@VM:~$ ping 192.168.53.2
PING 192.168.53.2 (192.168.53.2) 56(84) bytes of data.
64 bytes from 192.168.53.2: icmp_seq=1 ttl=64 time=4.63 ms
64 bytes from 192.168.53.2: icmp_seq=2 ttl=64 time=3.67 ms
64 bytes from 192.168.53.2: icmp_seq=3 ttl=64 time=4.04 ms
64 bytes from 192.168.53.2: icmp_seq=4 ttl=64 time=3.74 ms
64 bytes from 192.168.53.2: icmp_seq=5 ttl=64 time=3.58 ms
64 bytes from 192.168.53.2: icmp_seq=6 ttl=64 time=3.41 ms
64 bytes from 192.168.53.2: icmp_seq=7 ttl=64 time=3.75 ms
64 bytes from 192.168.53.2: icmp_seq=8 ttl=64 time=4.34 ms
^C
--- 192.168.53.2 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7021ms
```

```
go in:
IP / ICMP 192.168.53.99 > 192.168.53.2 echo-request 0 / Raw
this is request
8
go out
<bound method Packet.summary of <IP  src=192.168.53.2 dst=192.168.53.99 |>>
```

- Instead of writing an IP packet to the interface, write some arbitrary data to the interface, and report your observation.

如果不写入packet，随意写一些数据到interface：

```
os.write(tun, bytes([1,2,3,4]))
```

会发生报错：

```
OSError: [Errno 22] Invalid argument
```

说明tun interface有一定检查机制

Task 3: Send the IP Packet to VPN Server Through a Tunnel

Run the tun server.py program on VPN Server, and then run tun client.py on Host U. To test whether the tunnel works or not, ping any IP address belonging to the 192.168.53.0/24network. What is printed out on VPN Server? Why?

```
172.16.133.129:57661 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.2
172.16.133.129:57661 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.2
172.16.133.129:57661 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.2
172.16.133.129:57661 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.2
172.16.133.129:57661 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.2
172.16.133.129:57661 --> 0.0.0.0:9090
```

server 打印出packet的两层信息：

第一层是外面的UDP层，包括ip和端口号，第二层是内部的IP层，包括ping请求的源ip目的ip。

Let us ping Host V, and see whether the ICMP packet is sent to VPN Server through the tunnel. If not, what are the problems?

不能，因为Host V的地址没有被写入Host U的路由表来被送往wu0 interface，默认不会由server进行路由转发。

You need to solve this problem, so the ping packet can be sent through the tunnel. This is done through routing, i.e., packets going to the 192.168.60.0/24 network should be routed to the TUN interface and be given to the tun client.py program. The following command shows how to add an entry to the routing table:

写入：

```
sudo ip route add 192.168.22.0/24 dev wu0 via 192.168.53.99
```

Please provide proofs to demonstrate that when you ping an IP address in the 192.168.60.0/24 network, the ICMP packets are received by tun server.py through the tunnel.

```
 Inside: 192.168.53.99 --> 192.168.22.129
172.16.133.129:46796 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.22.129
172.16.133.129:46796 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.22.129
172.16.133.129:46796 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.22.129
172.16.133.129:46796 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.22.129
172.16.133.129:46796 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.22.129
172.16.133.129:46796 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.22.129
172.16.133.129:46796 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.22.129
```

此时继续尝试ping HostV，server上输出的信息表示packet已经被送入了tunnel。

Task 4: Set Up the VPN Server

Please modify tun server.py, so it can do the following:

> - *Create a TUN interface and configure it.*
> - *Get the data from the socket interface; treat the received data as an IP packet.*
>
> - *Write the packet to the TUN interface.*

If everything is set up properly, we can ping Host V from Host U. The ICMP echo request packets should eventually arrive at Host V through the tunnel. Please show your proof. It should be noted that although Host V will respond to the ICMP packets, the reply will not get back to Host U, because we have not set up everything yet. Therefore, for this task, it is sufficient to show (using Wireshark) that the ICMP packets have arrived at Host V.

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

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

print("Interface Name: {}".format(ifname))
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
while True:
    packet = os.read(tun, 2048)
    if True:
        ip = IP(packet)
        print(ip.summary())
        sock.sendto(packet, ('172.16.133.131', 9090))
```

在server创建一个interface：jq0

```
#!/usr/bin/python3
from scapy.all import *
import fcntl
import struct
import os
import time

#Create a TUN interface and configure it.
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'Jq%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")

#configure
os.system("ip addr add 192.168.53.11/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
print("Interface Name: {}".format(ifname))


#Get the data from the socket interface; treat the received data
as an IP packet.
```

```python
#Write the packet to the TUN 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))

  os.write(tun, bytes(pkt))
  print('sent')
```

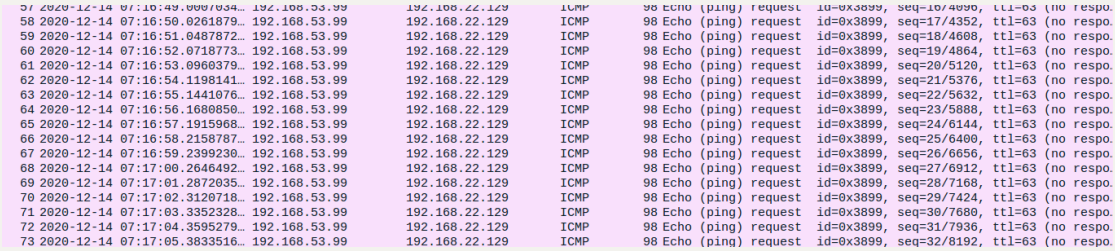
在从socket interface 监听到HostU的ping后，将packet作为IP包，送进jq0；

```
172.16.133.129:46796 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.22.129
sent
172.16.133.129:46796 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.22.129
sent
172.16.133.129:46796 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.22.129
sent
172.16.133.129:46796 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.22.129
sent
172.16.133.129:46796 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.22.129
sent
172.16.133.129:46796 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.22.129
sent
```

server端的wireshark对**interface jq0**抓包，表明ping包被成功发送往Host V（192.168.22.129）

3	2020-12-14 07:16:33.6468021…	192.168.53.99	192.168.22.129	ICMP	84	Echo (ping) request id=0x3899,
4	2020-12-14 07:16:34.6654943…	192.168.53.99	192.168.22.129	ICMP	84	Echo (ping) request id=0x3899,
5	2020-12-14 07:16:35.6893692…	192.168.53.99	192.168.22.129	ICMP	84	Echo (ping) request id=0x3899,
6	2020-12-14 07:16:36.7148194…	192.168.53.99	192.168.22.129	ICMP	84	Echo (ping) request id=0x3899,
7	2020-12-14 07:16:37.7383069…	192.168.53.99	192.168.22.129	ICMP	84	Echo (ping) request id=0x3899,
8	2020-12-14 07:16:38.7614778…	192.168.53.99	192.168.22.129	ICMP	84	Echo (ping) request id=0x3899,
9	2020-12-14 07:16:39.7855211…	192.168.53.99	192.168.22.129	ICMP	84	Echo (ping) request id=0x3899,
10	2020-12-14 07:16:40.8087898…	192.168.53.99	192.168.22.129	ICMP	84	Echo (ping) request id=0x3899,
11	2020-12-14 07:16:41.8342062…	192.168.53.99	192.168.22.129	ICMP	84	Echo (ping) request id=0x3899,
12	2020-12-14 07:16:42.8583653…	192.168.53.99	192.168.22.129	ICMP	84	Echo (ping) request id=0x3899,
13	2020-12-14 07:16:43.8811066…	192.168.53.99	192.168.22.129	ICMP	84	Echo (ping) request id=0x3899,
14	2020-12-14 07:16:44.9058293…	192.168.53.99	192.168.22.129	ICMP	84	Echo (ping) request id=0x3899,

Host V的 wireshark也表明自己收到了ping 包：



而在这之前，两者的wireshark都无法抓到ping包；表明了VPN server端的配置有效。

Task 5: Handling Traffic in Both Directions

> *To achieve that, our TUN client and server programs need to read data from two interfaces, the TUN interface and the socket interface.*

Testing. Once this is done, we should be able to communicate with Machine V from Machine U, and the VPN tunnel (un-encrypted) is now complete. Please show your wireshark proof using about ping and telnet commands. In your proof, you need to point out how your packets flow.

Server

```
#!/usr/bin/python3
from scapy.all import *
import fcntl
import struct
import os
import time
from select import *

#Create a TUN interface and configure it.
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'Jq%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")

#configure
os.system("ip addr add 192.168.53.11/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
```

```
print("Interface Name: {}".format(ifname))


#Get the data from the socket interface; treat the received data
as an IP packet.
#Write the packet to the TUN 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:

  # 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))
      #... (code needs to be added by students) ...
      if pkt.dst == '192.168.22.129':
        os.write(tun, bytes(pkt))
        print('request go ')


    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) ...
      if pkt.src == '192.168.22.129':
      # Send the packet via the tunnel
        sock.sendto(packet, ('172.16.133.129', 9090))
        print('reply go')

'''
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))
  print('sent')

'''
```

client

```
#!/usr/bin/python3
import fcntl
import struct
import os
import time
from scapy.all import *
from select 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'wu%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")

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

print("Interface Name: {}".format(ifname))

# Create UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
IP_A = "0.0.0.0"
PORT = 9090

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))
        #... (code needs to be added by students) ...
        if pkt.dst == '192.168.53.99':
          #注意这里必须是tunnel的ip地址不是Host U自己原本的，不然无法成功；
          print('reply back')
          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) ...
        if pkt.dst == '192.168.22.129':
        # Send the packet via the tunnel
          sock.sendto(packet, ('172.16.133.131', 9090))
          print('request go')
```

按照这种方法尝试，可发现虽然request被成功传递到Host V，但是reply却没有被发送；

经过漫长的尝试发现可能是因为上个lab中提到的tunnel的虚拟ip地址**192.168.53.xxx**无法被查找的问题；

于是在server尝试按上个lab的方法配制了新的NAT，让server发给Host V的包以internal网络的IP地址进行ip伪装：

```
tunnel$ sudo iptables -t nat -A POSTROUTING -j MASQUERADE -o ens38
```

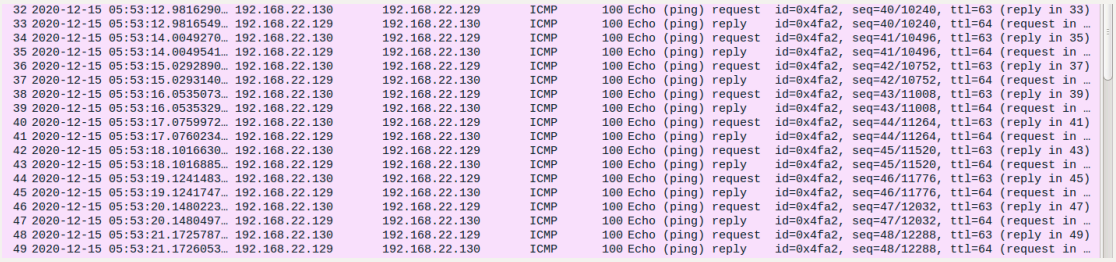
在尝试中还在server端路由配置了如下命令：

```
sudo ip route add 192.168.53.0/24 dev Jq0 via 192.168.53.11
```

```
sudo ip route add 172.16.133.0/24 dev Jq0 via 192.168
```

现在reply也能被成功发放：

Host V端的Wireshark:



```
32 2020-12-15 05:53:12.9816290… 192.168.22.130 192.168.22.129 ICMP 100 Echo (ping) request id=0x4fa2, seq=40/10240, ttl=63 (reply in 33)
33 2020-12-15 05:53:12.9816549… 192.168.22.129 192.168.22.130 ICMP 100 Echo (ping) reply   id=0x4fa2, seq=40/10240, ttl=64 (request in …
34 2020-12-15 05:53:14.0049270… 192.168.22.130 192.168.22.129 ICMP 100 Echo (ping) request id=0x4fa2, seq=41/10496, ttl=63 (reply in 35)
35 2020-12-15 05:53:14.0049541… 192.168.22.129 192.168.22.130 ICMP 100 Echo (ping) reply   id=0x4fa2, seq=41/10496, ttl=64 (request in …
36 2020-12-15 05:53:15.0292890… 192.168.22.130 192.168.22.129 ICMP 100 Echo (ping) request id=0x4fa2, seq=42/10752, ttl=63 (reply in 37)
37 2020-12-15 05:53:15.0293140… 192.168.22.129 192.168.22.130 ICMP 100 Echo (ping) reply   id=0x4fa2, seq=42/10752, ttl=64 (request in …
38 2020-12-15 05:53:16.0535073… 192.168.22.130 192.168.22.129 ICMP 100 Echo (ping) request id=0x4fa2, seq=43/11008, ttl=63 (reply in 39)
39 2020-12-15 05:53:16.0535329… 192.168.22.129 192.168.22.130 ICMP 100 Echo (ping) reply   id=0x4fa2, seq=43/11008, ttl=64 (request in …
40 2020-12-15 05:53:17.0759972… 192.168.22.130 192.168.22.129 ICMP 100 Echo (ping) request id=0x4fa2, seq=44/11264, ttl=63 (reply in 41)
41 2020-12-15 05:53:17.0760234… 192.168.22.129 192.168.22.130 ICMP 100 Echo (ping) reply   id=0x4fa2, seq=44/11264, ttl=64 (request in …
42 2020-12-15 05:53:18.1016630… 192.168.22.130 192.168.22.129 ICMP 100 Echo (ping) request id=0x4fa2, seq=45/11520, ttl=63 (reply in 43)
43 2020-12-15 05:53:18.1016885… 192.168.22.129 192.168.22.130 ICMP 100 Echo (ping) reply   id=0x4fa2, seq=45/11520, ttl=64 (request in …
44 2020-12-15 05:53:19.1241483… 192.168.22.130 192.168.22.129 ICMP 100 Echo (ping) request id=0x4fa2, seq=46/11776, ttl=63 (reply in 45)
45 2020-12-15 05:53:19.1241747… 192.168.22.129 192.168.22.130 ICMP 100 Echo (ping) reply   id=0x4fa2, seq=46/11776, ttl=64 (request in …
46 2020-12-15 05:53:20.1480223… 192.168.22.130 192.168.22.129 ICMP 100 Echo (ping) request id=0x4fa2, seq=47/12032, ttl=63 (reply in 47)
47 2020-12-15 05:53:20.1480497… 192.168.22.129 192.168.22.130 ICMP 100 Echo (ping) reply   id=0x4fa2, seq=47/12032, ttl=64 (request in …
48 2020-12-15 05:53:21.1725787… 192.168.22.130 192.168.22.129 ICMP 100 Echo (ping) request id=0x4fa2, seq=48/12288, ttl=63 (reply in 49)
49 2020-12-15 05:53:21.1726053… 192.168.22.129 192.168.22.130 ICMP 100 Echo (ping) reply   id=0x4fa2, seq=48/12288, ttl=64 (request in …
```

server端输出信息：

```
From socket <==: 192.168.53.99 --> 192.168.22.129
request go
From tun ==>: 192.168.22.129 --> 192.168.53.99
reply go
From socket <==: 192.168.53.99 --> 192.168.22.129
request go
From tun ==>: 192.168.22.129 --> 192.168.53.99
reply go
```

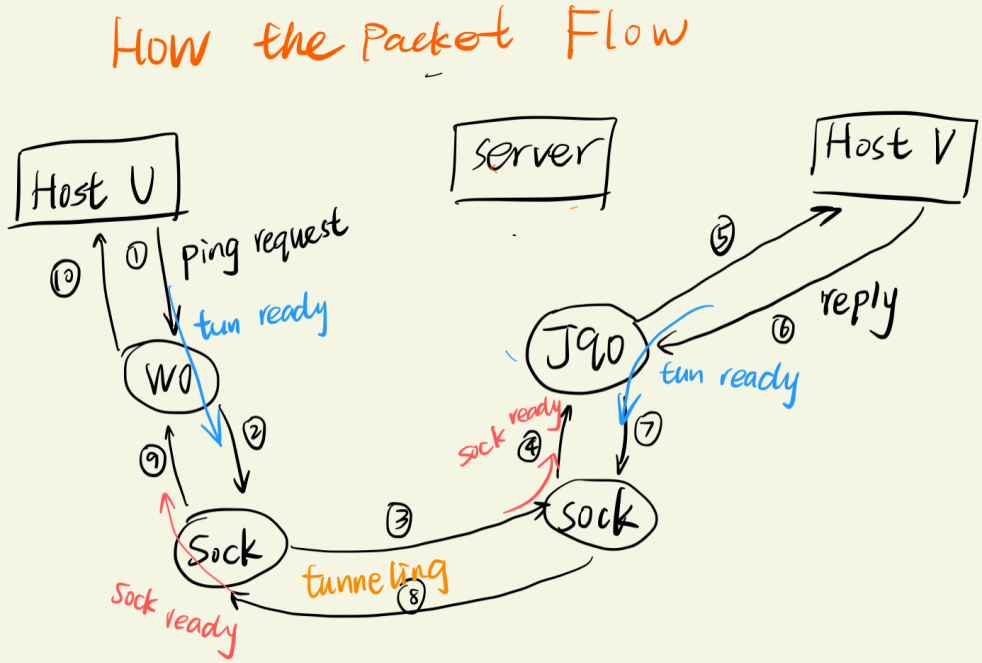
Host U输出信息：

```
From socket <==: 192.168.22.129 --> 192.168.53.99
reply back
From tun ==>: 192.168.53.99 --> 192.168.22.129
request go
From socket <==: 192.168.22.129 --> 192.168.53.99
reply back
From tun ==>: 192.168.53.99 --> 192.168.22.129
request go
From socket <==: 192.168.22.129 --> 192.168.53.99
reply back
From tun ==>: 192.168.53.99 --> 192.168.22.129
request go
From socket <==: 192.168.22.129 --> 192.168.53.99
reply back
From tun ==>: 192.168.53.99 --> 192.168.22.129
request go
From socket <==: 192.168.22.129 --> 192.168.53.99
```

ping成功收到reply：

```
[12/15/20]seed@VM:~$ ping 192.168.22.129
PING 192.168.22.129 (192.168.22.129) 56(84) bytes of data.
64 bytes from 192.168.22.129: icmp_seq=1 ttl=63 time=3.99 ms
64 bytes from 192.168.22.129: icmp_seq=2 ttl=63 time=3.63 ms
64 bytes from 192.168.22.129: icmp_seq=3 ttl=63 time=3.77 ms
64 bytes from 192.168.22.129: icmp_seq=4 ttl=63 time=3.75 ms
64 bytes from 192.168.22.129: icmp_seq=5 ttl=63 time=4.03 ms
64 bytes from 192.168.22.129: icmp_seq=6 ttl=63 time=3.55 ms
64 bytes from 192.168.22.129: icmp_seq=7 ttl=63 time=3.87 ms
64 bytes from 192.168.22.129: icmp_seq=8 ttl=63 time=3.67 ms
```

总结：packet的流向图：



进行telnet命令，同样成功

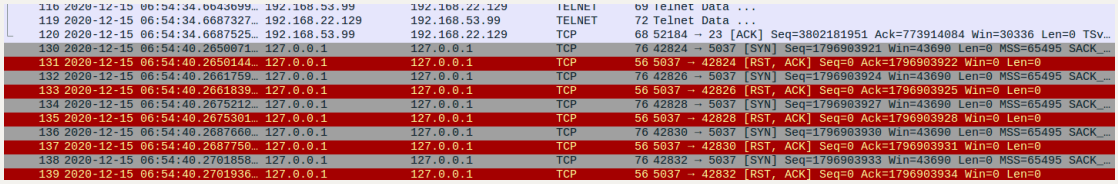
```
[12/15/20]seed@VM:~$ telnet 192.168.22.129
Trying 192.168.22.129...
Connected to 192.168.22.129.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Tue Dec  1 07:38:13 EST 2020 from 172.16.133.129 on pts/4
/usr/lib/update-notifier/update-motd-fsck-at-reboot:[:59: integer expression e
ected:            0
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

1 package can be updated.
0 updates are security updates.
```

Task 6: Tunnel-Breaking Experiment

Q：we break the VPN tunnel by stopping the tun client.py or tun server.py program. We then type something in the telnet window. Do you see what you type? What happens to the TCP connection? Is the connection broken?

断开链接后，打出的东西无法显示在telnet窗口上；

telnet链接表面上还是链接状态，但是实际上Host U 与 Host V通过tun的链接已经断开；所以无法再传递数据

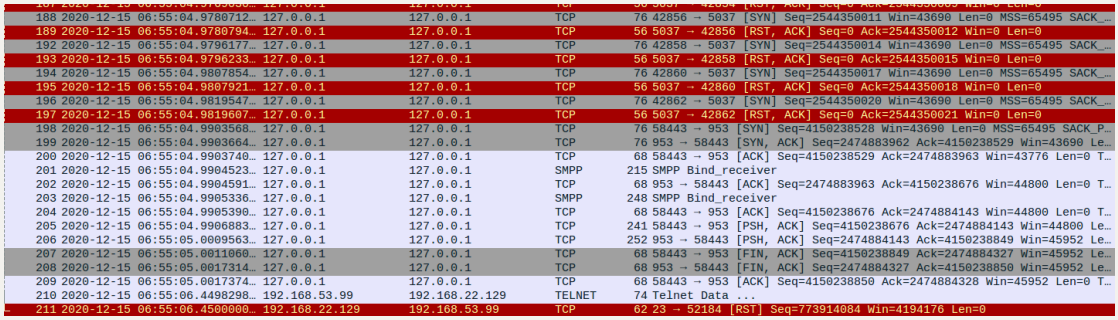


Q：Once the tunnel is re-established, what is going to happen to the telnet connection? Please describe and explain your observations.

```
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

1 package can be updated.
0 updates are security updates.

[12/15/20]seed@VM:~$ Connection closed by foreign host.
```

tunnel重新链接后，telnet connection被彻底断开，显示：connection closed by foreign host.

因为此时当链接重新被建立时，Host U收到了Host V发来的RST报文，TCP链接被重置



188	2020-12-15 06:55:04.9780712…	127.0.0.1	127.0.0.1	TCP	76 42856 → 5037 [SYN] Seq=2544350011 Win=43690 Len=0 MSS=65495 SACK_…
189	2020-12-15 06:55:04.9780794…	127.0.0.1	127.0.0.1	TCP	56 5037 → 42856 [RST, ACK] Seq=0 Ack=2544350012 Win=0 Len=0
192	2020-12-15 06:55:04.9796177…	127.0.0.1	127.0.0.1	TCP	76 42858 → 5037 [SYN] Seq=2544350014 Win=43690 Len=0 MSS=65495 SACK_…
193	2020-12-15 06:55:04.9796233…	127.0.0.1	127.0.0.1	TCP	56 5037 → 42858 [RST, ACK] Seq=0 Ack=2544350015 Win=0 Len=0
194	2020-12-15 06:55:04.9807854…	127.0.0.1	127.0.0.1	TCP	76 42860 → 5037 [SYN] Seq=2544350017 Win=43690 Len=0 MSS=65495 SACK_…
195	2020-12-15 06:55:04.9807921…	127.0.0.1	127.0.0.1	TCP	56 5037 → 42860 [RST, ACK] Seq=0 Ack=2544350018 Win=0 Len=0
196	2020-12-15 06:55:04.9819547…	127.0.0.1	127.0.0.1	TCP	76 42862 → 5037 [SYN] Seq=2544350020 Win=43690 Len=0 MSS=65495 SACK_…
197	2020-12-15 06:55:04.9819607…	127.0.0.1	127.0.0.1	TCP	56 5037 → 42862 [RST, ACK] Seq=0 Ack=2544350021 Win=0 Len=0
198	2020-12-15 06:55:04.9903568…	127.0.0.1	127.0.0.1	TCP	76 58443 → 953 [SYN] Seq=4150238528 Win=43690 Len=0 MSS=65495 SACK_P…
199	2020-12-15 06:55:04.9903664…	127.0.0.1	127.0.0.1	TCP	76 953 → 58443 [SYN, ACK] Seq=2474883962 Ack=4150238529 Win=43690 Le…
200	2020-12-15 06:55:04.9903740…	127.0.0.1	127.0.0.1	TCP	68 58443 → 953 [ACK] Seq=4150238529 Ack=2474883963 Win=43776 Len=0 T…
201	2020-12-15 06:55:04.9904523…	127.0.0.1	127.0.0.1	SMPP	215 SMPP Bind_receiver
202	2020-12-15 06:55:04.9904591…	127.0.0.1	127.0.0.1	TCP	68 953 → 58443 [ACK] Seq=2474883963 Ack=4150238676 Win=44800 Len=0 T…
203	2020-12-15 06:55:04.9905336…	127.0.0.1	127.0.0.1	SMPP	248 SMPP Bind_receiver
204	2020-12-15 06:55:04.9905390…	127.0.0.1	127.0.0.1	TCP	68 58443 → 953 [ACK] Seq=4150238676 Ack=2474884143 Win=44800 Len=0 T…
205	2020-12-15 06:55:04.9906883…	127.0.0.1	127.0.0.1	TCP	241 58443 → 953 [PSH, ACK] Seq=4150238676 Ack=2474884143 Win=44800 Le…
206	2020-12-15 06:55:05.0009563…	127.0.0.1	127.0.0.1	TCP	252 953 → 58443 [PSH, ACK] Seq=2474884143 Ack=4150238849 Win=45952 Le…
207	2020-12-15 06:55:05.0011060…	127.0.0.1	127.0.0.1	TCP	68 58443 → 953 [FIN, ACK] Seq=4150238849 Ack=2474884327 Win=45952 Le…
208	2020-12-15 06:55:05.0017314…	127.0.0.1	127.0.0.1	TCP	68 953 → 58443 [FIN, ACK] Seq=2474884327 Ack=4150238850 Win=45952 Le…
209	2020-12-15 06:55:05.0017374…	127.0.0.1	127.0.0.1	TCP	68 58443 → 953 [ACK] Seq=4150238850 Ack=2474884328 Win=45952 Len=0 T…
210	2020-12-15 06:55:06.4498298…	192.168.53.99	192.168.22.129	TELNET	74 Telnet Data ...
211	2020-12-15 06:55:06.4500000…	192.168.22.129	192.168.53.99	TCP	62 23 → 52184 [RST] Seq=773914084 Win=4194176 Len=0

Task 7: Routing Experiment on Host V

> *n the real world, Host V may be a few hops away from the VPN server, and the default routing entry may not guarantee to route the return packet back to the VPN server. Routing tables inside a private network have to be set up properly to ensure that packets going to the other end of the tunnel will be routed to the VPN server.*

we will remove the default entry from Host V, and add a more specific entry to the routing table, so the return packets can be routed back to the VPN server. The network prefix for this entry should have no less than 24 bits. Students can use the following commands to remove the default entry and add a new entry:

```
sudo ip route del 0.0.0.0/0
```

```
sudo ip route add <network> dev <interface> via <router ip>
```

这一步在Task5中我已经添加过：

```
sudo ip route add 192.168.53.0/24 dev Jq0 via 192.168.53.11
```

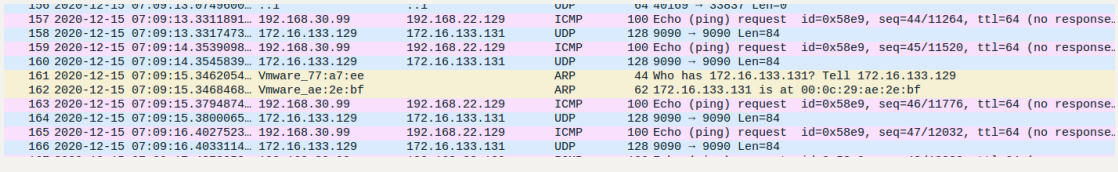
```
sudo ip route add 172.16.133.0/24 dev Jq0 via 192.168.53.11
```

Task 8: Experiment with the TUN IP Address

如果tun_server 和tun_client的interface不再使用相同网络下的ip地址，数据包会被丢弃。

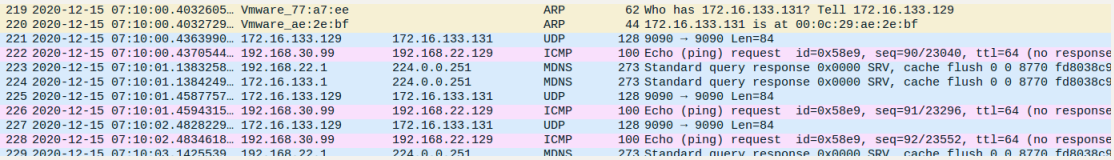
• Where are the packets dropped? Please provide evidence using Wireshark traces.

Host U上：



157	2020-12-15 07:09:13.3311891…	192.168.30.99	192.168.22.129	ICMP	100 Echo (ping) request id=0x58e9, seq=44/11264, ttl=64 (no response…
158	2020-12-15 07:09:13.3317473…	172.16.133.129	172.16.133.131	UDP	128 9090 → 9090 Len=84
159	2020-12-15 07:09:14.3539098…	192.168.30.99	192.168.22.129	ICMP	100 Echo (ping) request id=0x58e9, seq=45/11520, ttl=64 (no response…
160	2020-12-15 07:09:14.3545839…	172.16.133.129	172.16.133.131	UDP	128 9090 → 9090 Len=84
161	2020-12-15 07:09:15.3462054…	Vmware_77:a7:ee		ARP	44 Who has 172.16.133.131? Tell 172.16.133.129
162	2020-12-15 07:09:15.3468468…	Vmware_ae:2e:bf		ARP	62 172.16.133.131 is at 00:0c:29:ae:2e:bf
163	2020-12-15 07:09:15.3794874…	192.168.30.99	192.168.22.129	ICMP	100 Echo (ping) request id=0x58e9, seq=46/11776, ttl=64 (no response…
164	2020-12-15 07:09:15.3800065…	172.16.133.129	172.16.133.131	UDP	128 9090 → 9090 Len=84
165	2020-12-15 07:09:16.4027523…	192.168.30.99	192.168.22.129	ICMP	100 Echo (ping) request id=0x58e9, seq=47/12032, ttl=64 (no response…
166	2020-12-15 07:09:16.4033114…	172.16.133.129	172.16.133.131	UDP	128 9090 → 9090 Len=84

Host V上：



可以得知packet是在server的interface Jq0上被丢弃的

• Why are the packets dropped? See the hint below.

> *Linux kernel implements a filtering rule called reverse path filtering, which ensures the symmetric routing rule. When a packet with the source IP address X comes from an interface (say I), the OS will check whether the return packet will return from the same interface, i.e., whether the routing for packets going to X is symmetric.*
> *If this interface is not I, i.e., different from where the original packet comes from, the routing*
> *path is asymmetric. In this case, the kernel will drop the packet.*

分析：此时，两个**interface**不再属于同一网络；当**packet**被传到**server**时，**os**会根据**reverse path filtering mechanism**来进行检测：发现来自**Host U**的**ping request**包，是通过**wu0 interface**（**ip=192.168.30.0/24**）被发送的；而返回的**ping request**，却要通过另外一个网络中的**Jq0 interface**（**ip=192.168.53.0/24**）送回**Host U**；这不符合路由路径对称性，于是将**packet**丢弃。

• How to solve this problem? You are not allowed to change the IP address on any of the TUN interfaces? Please demonstrate that your solution works.

解决方案：可以关闭**linux**的据**reverse path filtering mechanism**

```
echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter
```

收到了reply

```
[12/15/20]seed@VM:~$ ping 192.168.22.129
PING 192.168.22.129 (192.168.22.129) 56(84) bytes of data.
64 bytes from 192.168.22.129: icmp_seq=1 ttl=63 time=3.99 ms
64 bytes from 192.168.22.129: icmp_seq=2 ttl=63 time=3.63 ms
64 bytes from 192.168.22.129: icmp_seq=3 ttl=63 time=3.77 ms
64 bytes from 192.168.22.129: icmp_seq=4 ttl=63 time=3.75 ms
64 bytes from 192.168.22.129: icmp_seq=5 ttl=63 time=4.03 ms
64 bytes from 192.168.22.129: icmp_seq=6 ttl=63 time=3.55 ms
64 bytes from 192.168.22.129: icmp_seq=7 ttl=63 time=3.87 ms
64 bytes from 192.168.22.129: icmp_seq=8 ttl=63 time=3.67 ms
```

Task 9: Experiment with the TAP Interface

> *The way how the TAP interface works is quite similar to the TUN interface. The main difference is that the kernel end of the TUN interface is hooked to the IP layer, while the kernel end of the TAP interface is hooked to the MAC layer. Therefore, the packet going through the TAP interface includes the MAC header, while the packet going through the TUN interface only includes the IP header. Other than getting the frames containing IP packets, using the TAP interface, applications can also get other types of frames, such as ARP frames.*

Try to ping an IP address in the 192.168.53.0/24 network; report and explain your observations.

```
#!/usr/bin/python3
import fcntl
import struct
import os
import time
from scapy.all import *
from select import *

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

# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")

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

print("Interface Name: {}".format(ifname))



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

```
    print(ether.summary())
```

```
Ether / ARP who has 192.168.53.2 says 192.168.53.99
Ether / ARP who has 192.168.53.2 says 192.168.53.99
Ether / ARP who has 192.168.53.2 says 192.168.53.99
Ether / IP / UDP / DNS Qry "b'_ipps._tcp.local.'"
Ether / ARP who has 192.168.53.2 says 192.168.53.99
Ether / ARP who has 192.168.53.2 says 192.168.53.99
Ether / IPv6 / UDP / DNS Qry "b'_ipps._tcp.local.'"
Ether / ARP who has 192.168.53.2 says 192.168.53.99
Ether / ARP who has 192.168.53.2 says 192.168.53.99
Ether / ARP who has 192.168.53.2 says 192.168.53.99
Ether / ARP who has 192.168.53.2 says 192.168.53.99
Ether / ARP who has 192.168.53.2 says 192.168.53.99
```

ping192.168.53.2之后tap捕获并且打印出了ARP查询报文；因为Host U 的tap interface并不知道192.168.53.2的mac地址，只能发送ARP包查询；而因为这个ip地址并不存在，并没有回复，于是ARP包被不断发送。

To make this more interesting, once you get an ethernet frame from the TAP interface, you can check whether it is an ARP request; if it is, generate a corresponding ARP reply and write it to the TAP interface.

To test your TAP program, you can run the arping command on any IP address. This command sends out an ARP request for the specified IP address via the specified interface. If your spoof-arp-reply TAP program works, you should be able to get a response. See the following examples.

```
#!/usr/bin/python3
import fcntl
import struct
import os
import time
from scapy.all import *
from select import *

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

# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")

#configure
os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
```

```
os.system("ip link set dev {} up".format(ifname))

print("Interface Name: {}".format(ifname))



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



  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='00:0c:29:70:30:23',#伪造Mac为 Host V 的
MAC地址
                dst='FF:FF:FF:FF:FF:FF'#广播发送
              )

      newarp = ARP(op=2,hwsrc='00:0c:29:70:30:23',#发送端以太网地址
                psrc=arp.pdst,#发送端ip
                hwdst=arp.hwsrc,#目的以太网地址
                pdst=arp.psrc#目的ip地址
              )

      newpkt = newether/newarp
      print("***** Fake response: {}".format(newpkt.summary()))
      os.write(tap, bytes(newpkt))
```

```
arping -I wut0 192.168.53.33
```

```
[12/16/20]seed@VM:~$ arping -I wut0 192.168.53.33
ARPING 192.168.53.33 from 192.168.53.99 wut0
Broadcast reply from 192.168.53.33 [00:0C:29:70:30:23]  2.377ms
Broadcast reply from 192.168.53.33 [00:0C:29:70:30:23]  2.475ms
Broadcast reply from 192.168.53.33 [00:0C:29:70:30:23]  2.435ms
Broadcast reply from 192.168.53.33 [00:0C:29:70:30:23]  2.348ms
Broadcast reply from 192.168.53.33 [00:0C:29:70:30:23]  2.478ms
Broadcast reply from 192.168.53.33 [00:0C:29:70:30:23]  2.799ms
Broadcast reply from 192.168.53.33 [00:0C:29:70:30:23]  3.316ms
```

```
--------------------------------
Ether / ARP who has 192.168.53.33 says 192.168.53.99
***** Fake response: Ether / ARP is at 00:0c:29:70:30:23 says 192.168.53.33
--------------------------------
Ether / ARP who has 192.168.53.33 says 192.168.53.99
***** Fake response: Ether / ARP is at 00:0c:29:70:30:23 says 192.168.53.33
--------------------------------
Ether / ARP who has 192.168.53.33 says 192.168.53.99
***** Fake response: Ether / ARP is at 00:0c:29:70:30:23 says 192.168.53.33
--------------------------------
Ether / ARP who has 192.168.53.33 says 192.168.53.99
***** Fake response: Ether / ARP is at 00:0c:29:70:30:23 says 192.168.53.33
--------------------------------
Ether / ARP who has 192.168.53.33 says 192.168.53.99
***** Fake response: Ether / ARP is at 00:0c:29:70:30:23 says 192.168.53.33
--------------------------------
Ether / ARP who has 192.168.53.33 says 192.168.53.99
***** Fake response: Ether / ARP is at 00:0c:29:70:30:23 says 192.168.53.33
```

`arping -I wut0 1.2.3.4`

```
ARPING 1.2.3.4 from 192.168.53.99 wut0
Broadcast reply from 1.2.3.4 [00:0C:29:70:30:23]  3.442ms
Broadcast reply from 1.2.3.4 [00:0C:29:70:30:23]  3.903ms
Broadcast reply from 1.2.3.4 [00:0C:29:70:30:23]  2.547ms
Broadcast reply from 1.2.3.4 [00:0C:29:70:30:23]  2.553ms
Broadcast reply from 1.2.3.4 [00:0C:29:70:30:23]  2.740ms
Broadcast reply from 1.2.3.4 [00:0C:29:70:30:23]  2.265ms
Broadcast reply from 1.2.3.4 [00:0C:29:70:30:23]  2.582ms
Broadcast reply from 1.2.3.4 [00:0C:29:70:30:23]  2.888ms
```

```
--------------------------------
Ether / ARP who has 1.2.3.4 says 192.168.53.99
***** Fake response: Ether / ARP is at 00:0c:29:70:30:23 says 1.2.3.4
--------------------------------
Ether / ARP who has 1.2.3.4 says 192.168.53.99
***** Fake response: Ether / ARP is at 00:0c:29:70:30:23 says 1.2.3.4
--------------------------------
Ether / ARP who has 1.2.3.4 says 192.168.53.99
***** Fake response: Ether / ARP is at 00:0c:29:70:30:23 says 1.2.3.4
--------------------------------
Ether / ARP who has 1.2.3.4 says 192.168.53.99
***** Fake response: Ether / ARP is at 00:0c:29:70:30:23 says 1.2.3.4
--------------------------------
Ether / ARP who has 1.2.3.4 says 192.168.53.99
***** Fake response: Ether / ARP is at 00:0c:29:70:30:23 says 1.2.3.4
--------------------------------
```

TAP interface成功在捕获到arp包之后，构造虚假arp response，回答中的mac地址都设为了Host V的地址（00:0c:29:70:30:23）