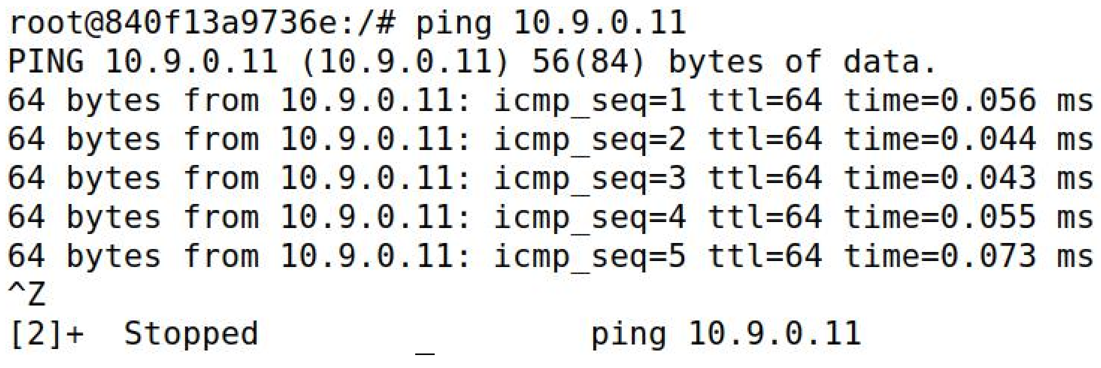
VPN Lab: The Container Version

57118229 袁超然

2 Task 1: Network Setup

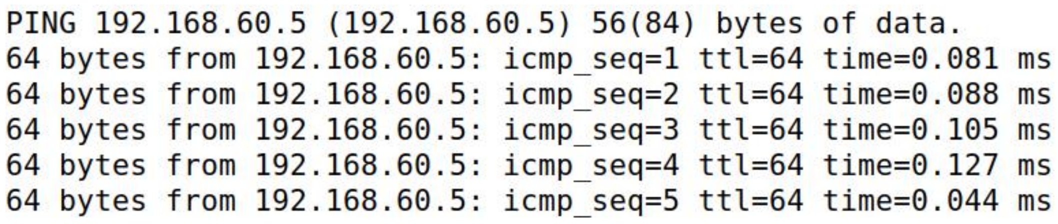
设置实验环境后，进行一下测试：

• Host U can communicate with VPN Server:



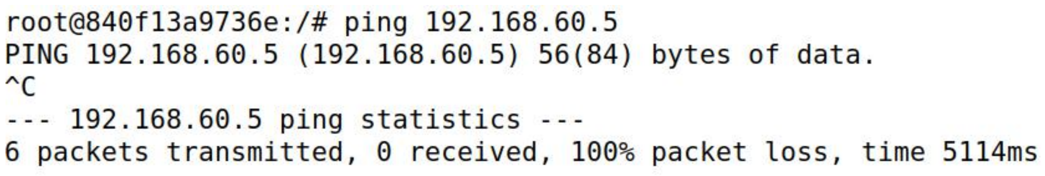
```
root@840f13a9736e:/# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.056 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.044 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.043 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.055 ms
64 bytes from 10.9.0.11: icmp_seq=5 ttl=64 time=0.073 ms
^Z
[2]+  Stopped                 ping 10.9.0.11
```

• VPN Server can communicate with Host V:



```
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=0.081 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=64 time=0.088 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=64 time=0.105 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=64 time=0.127 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=64 time=0.044 ms
```

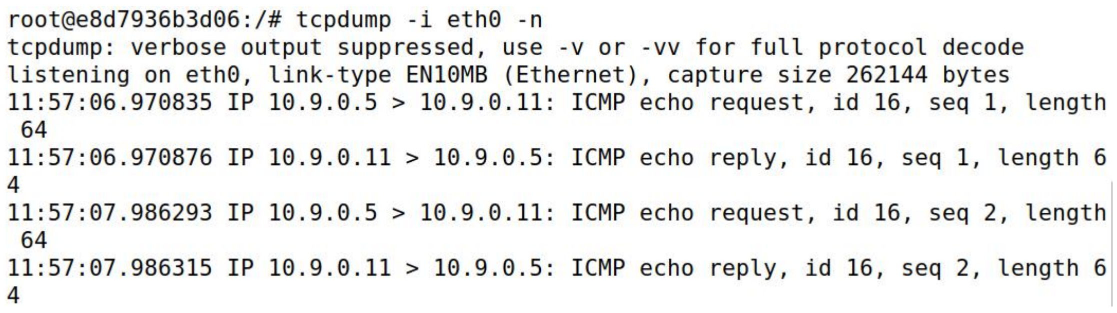
• Host U should not be able to communicate with Host V:



```
root@840f13a9736e:/# 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 ---
6 packets transmitted, 0 received, 100% packet loss, time 5114ms
```

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

10.9.0.0 子网：



```
root@e8d7936b3d06:/# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:57:06.970835 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 16, seq 1, length
 64
11:57:06.970876 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 16, seq 1, length 6
4
11:57:07.986293 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 16, seq 2, length
 64
11:57:07.986315 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 16, seq 2, length 6
4
```

192.168.60.0 子网：

```
root@e8d7936b3d06:/# tcpdump -i eth1 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
11:59:36.432773 IP 192.168.60.5 > 192.168.60.11: ICMP echo request, id 29, seq 1
, length 64
11:59:36.432799 IP 192.168.60.11 > 192.168.60.5: ICMP echo reply, id 29, seq 1,
length 64
11:59:37.458644 IP 192.168.60.5 > 192.168.60.11: ICMP echo request, id 29, seq 2
, length 64
11:59:37.458707 IP 192.168.60.11 > 192.168.60.5: ICMP echo reply, id 29, seq 2,
length 64
```

3 Task 2: Create and Configure TUN Interface

3.1 Task 2.a: Name of the Interface

在 Host U 上运行修改后的 tun.py，并查看网络接口，可以看到一个叫 ycr 的接口，且处于 DOWN 状态:

```
root@4ee09fe79651:/volumes# ip address
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
2: ycr: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group defau
lt qlen 500
    link/none
118: eth0@if119: <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
```

3.2 Task 2.b: Set up the TUN Interface

添加如下两行代码，给端口自动分配地址：

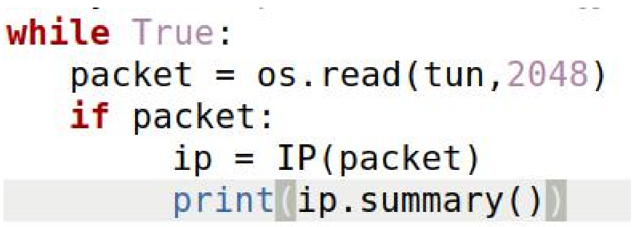
```
os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
```

查看新建的端口，不处于 DOWN 状态：

```
root@4ee09fe79651:/volumes# ip address
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
2: ycr: <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 ycr
       valid_lft forever preferred_lft forever
132: eth0@if133: <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
```

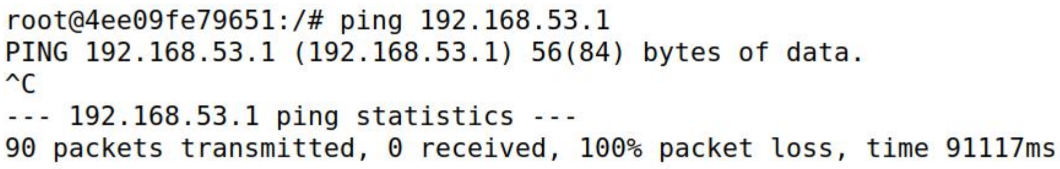
3.3 Task 2.c: Read from the TUN Interface

用以下代码替换 while 循环部分：



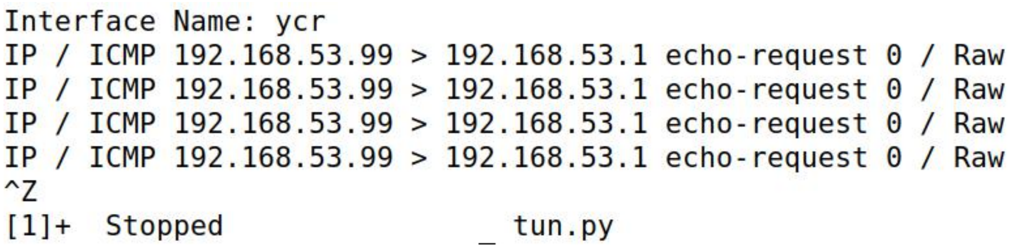
```
while True:
    packet = os.read(tun,2048)
    if packet:
        ip = IP(packet)
        print(ip.summary())
```

在主机 U 端 ping 192.168.53.1，无法连接：



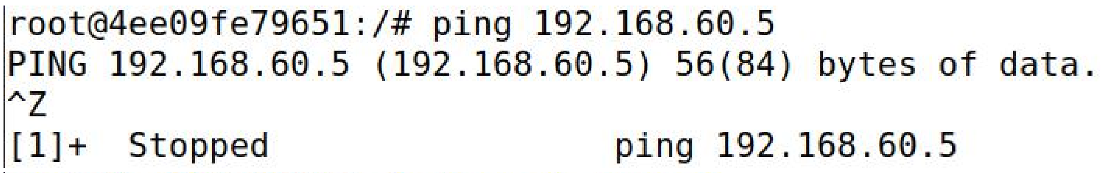
```
root@4ee09fe79651:/# ping 192.168.53.1
PING 192.168.53.1 (192.168.53.1) 56(84) bytes of data.
^C
--- 192.168.53.1 ping statistics ---
90 packets transmitted, 0 received, 100% packet loss, time 91117ms
```

同时运行上述程序，可知 icmp 请求报文已发送，但目标主机不存在，无法建立连接：

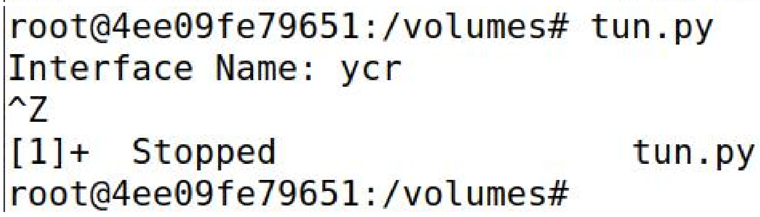


```
Interface Name: ycr
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
^Z
[1]+  Stopped                    _ tun.py
```

在主机 U 端 ping 主机 V，无法连接：

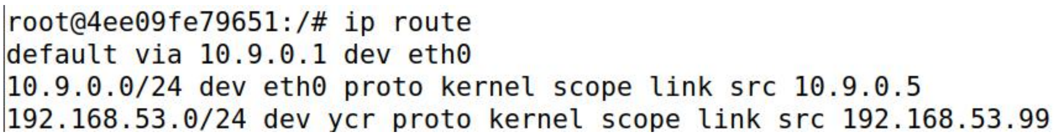


```
root@4ee09fe79651:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^Z
[1]+  Stopped                 ping 192.168.60.5
```



```
root@4ee09fe79651:/volumes# tun.py
Interface Name: ycr
^Z
[1]+  Stopped                 tun.py
root@4ee09fe79651:/volumes#
```

且查看路由信息，可知路由经过 eth0，而非 ycr 接口：

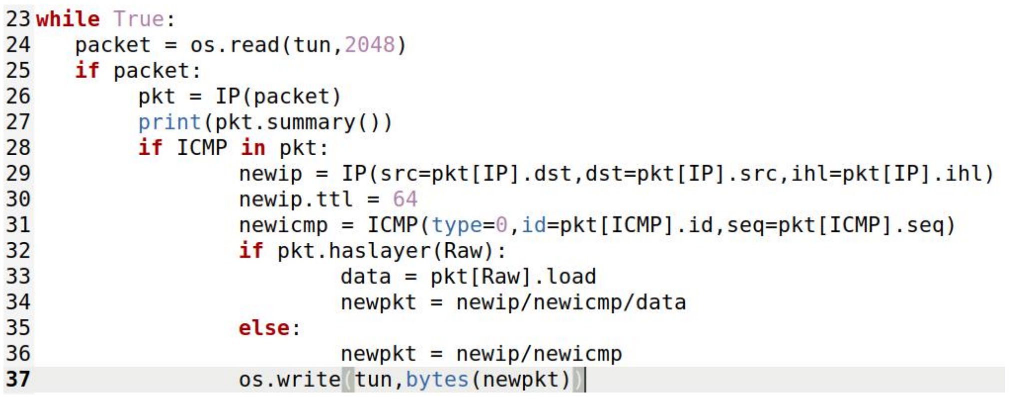


```
root@4ee09fe79651:/# ip route
default via 10.9.0.1 dev eth0
10.9.0.0/24 dev eth0 proto kernel scope link src 10.9.0.5
192.168.53.0/24 dev ycr proto kernel scope link src 192.168.53.99
```

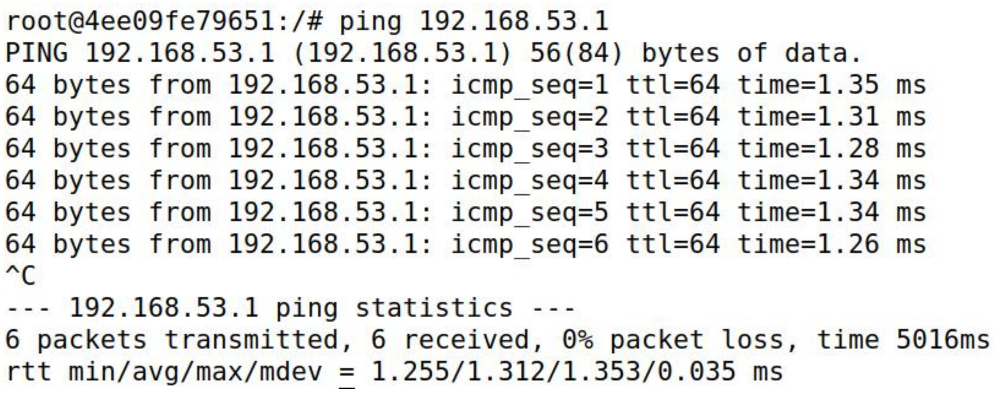
3.4 Task 2.d: Write to the TUN Interface

• After getting a packet from the TUN interface, if this packet is an ICMP echo request packet, construct a corresponding echo reply packet and write it to the TUN interface. Please provide evidence to show that the code works as expected.

用以下代码替换 while 循环部分：

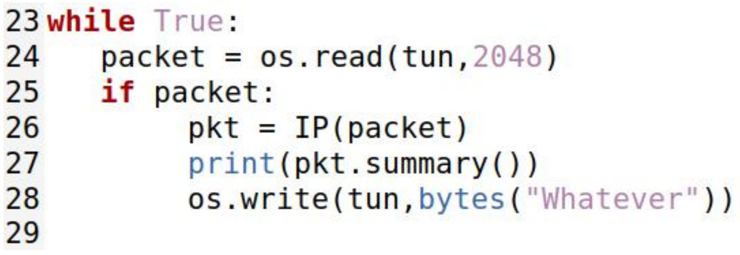

```python
while True:
    packet = os.read(tun,2048)
    if packet:
        pkt = IP(packet)
        print(pkt.summary())
        if ICMP in pkt:
                newip = IP(src=pkt[IP].dst,dst=pkt[IP].src,ihl=pkt[IP].ihl)
                newip.ttl = 64
                newicmp = ICMP(type=0,id=pkt[ICMP].id,seq=pkt[ICMP].seq)
                if pkt.haslayer(Raw):
                        data = pkt[Raw].load
                        newpkt = newip/newicmp/data
                else:
                        newpkt = newip/newicmp
                os.write(tun,bytes(newpkt))
```

在主机 U 端 ping 192.168.53.1，成功连接：

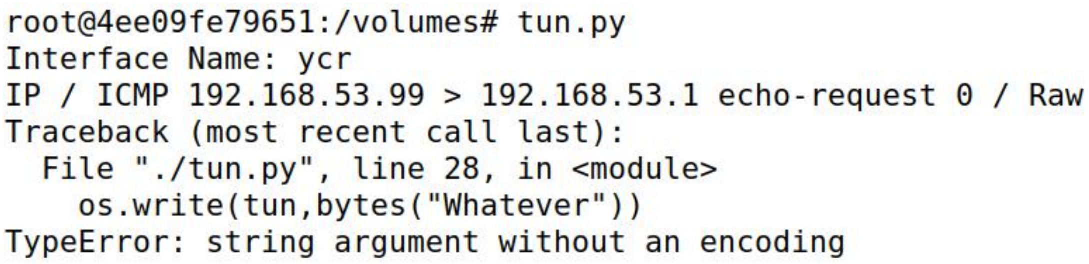

```
root@4ee09fe79651:/# ping 192.168.53.1
PING 192.168.53.1 (192.168.53.1) 56(84) bytes of data.
64 bytes from 192.168.53.1: icmp_seq=1 ttl=64 time=1.35 ms
64 bytes from 192.168.53.1: icmp_seq=2 ttl=64 time=1.31 ms
64 bytes from 192.168.53.1: icmp_seq=3 ttl=64 time=1.28 ms
64 bytes from 192.168.53.1: icmp_seq=4 ttl=64 time=1.34 ms
64 bytes from 192.168.53.1: icmp_seq=5 ttl=64 time=1.34 ms
64 bytes from 192.168.53.1: icmp_seq=6 ttl=64 time=1.26 ms
^C
--- 192.168.53.1 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5016ms
rtt min/avg/max/mdev = 1.255/1.312/1.353/0.035 ms
```

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

用以下代码替换 while 循环部分：

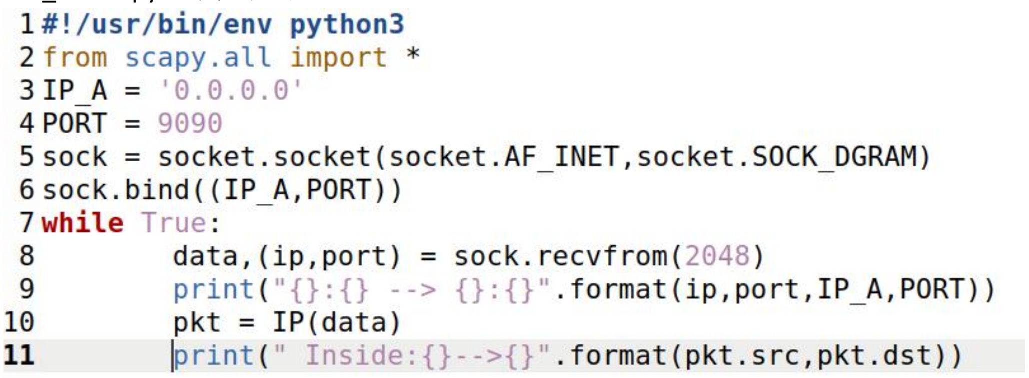

```python
23 while True:
24     packet = os.read(tun,2048)
25     if packet:
26         pkt = IP(packet)
27         print(pkt.summary())
28         os.write(tun,bytes("Whatever"))
29
```

可以看到因为没有 IP 包的相应构造方式，会显示无法解码，造成错误而发生中断。


```
root@4ee09fe79651:/volumes# tun.py
Interface Name: ycr
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
Traceback (most recent call last):
  File "./tun.py", line 28, in <module>
    os.write(tun,bytes("Whatever"))
TypeError: string argument without an encoding
```

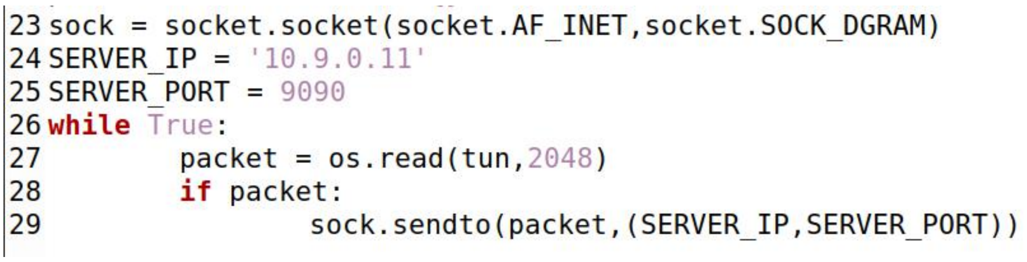
4 Task 3: Send the IP Packet to VPN Server Through a Tunnel

tun_server.py 的代码如下：



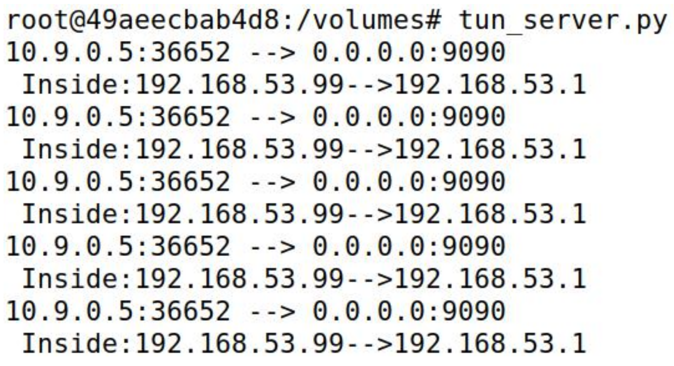
```
#!/usr/bin/env python3
from scapy.all import *
IP_A = '0.0.0.0'
PORT = 9090
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
sock.bind((IP_A,PORT))
while True:
        data,(ip,port) = sock.recvfrom(2048)
        print("{}:{} --> {}:{}".format(ip,port,IP_A,PORT))
        pkt = IP(data)
        print(" Inside:{}-->{}".format(pkt.src,pkt.dst))
```

tun_client.py 的代码的 while True 部分如下：



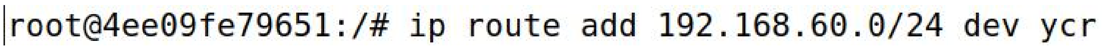
```
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
SERVER_IP = '10.9.0.11'
SERVER_PORT = 9090
while True:
        packet = os.read(tun,2048)
        if packet:
                sock.sendto(packet,(SERVER_IP,SERVER_PORT))
```

在 VPN server 上运行 tun_server.py，在主机 U 上运行 tun_client.py，然后在 U 上 ping 192.168.53.1 网段的 IP，可以看到 VPN server 上收到了相应的报文。但是 ping 192.168.60.0/24 网段的 IP 则没有反应：



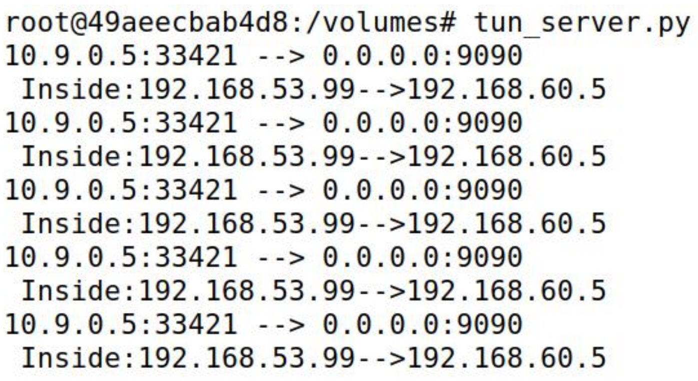
```
root@49aeecbab4d8:/volumes# tun_server.py
10.9.0.5:36652 --> 0.0.0.0:9090
 Inside:192.168.53.99-->192.168.53.1
10.9.0.5:36652 --> 0.0.0.0:9090
 Inside:192.168.53.99-->192.168.53.1
10.9.0.5:36652 --> 0.0.0.0:9090
 Inside:192.168.53.99-->192.168.53.1
10.9.0.5:36652 --> 0.0.0.0:9090
 Inside:192.168.53.99-->192.168.53.1
10.9.0.5:36652 --> 0.0.0.0:9090
 Inside:192.168.53.99-->192.168.53.1
```

为了能够使 60 网段的报文通过 tunnel，还需要增加一条路由：



```
root@4ee09fe79651:/# ip route add 192.168.60.0/24 dev ycr
```

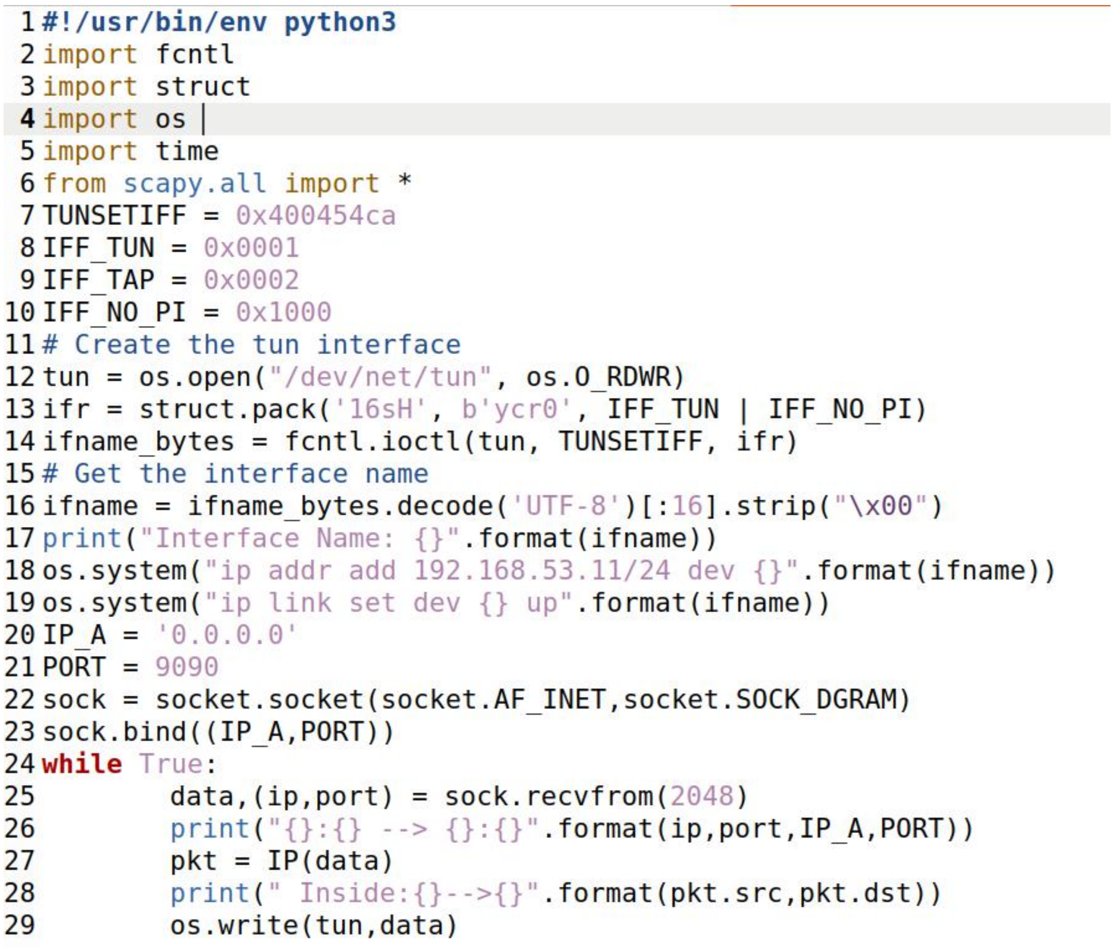
再次重复上述步骤，可以看见相应的报文在 VPN server 端接收：



```
root@49aeecbab4d8:/volumes# tun_server.py
10.9.0.5:33421 --> 0.0.0.0:9090
 Inside:192.168.53.99-->192.168.60.5
10.9.0.5:33421 --> 0.0.0.0:9090
 Inside:192.168.53.99-->192.168.60.5
10.9.0.5:33421 --> 0.0.0.0:9090
 Inside:192.168.53.99-->192.168.60.5
10.9.0.5:33421 --> 0.0.0.0:9090
 Inside:192.168.53.99-->192.168.60.5
10.9.0.5:33421 --> 0.0.0.0:9090
 Inside:192.168.53.99-->192.168.60.5
```

5 Task 4: Set Up the VPN Server

修改 tun_server.py 为如下：



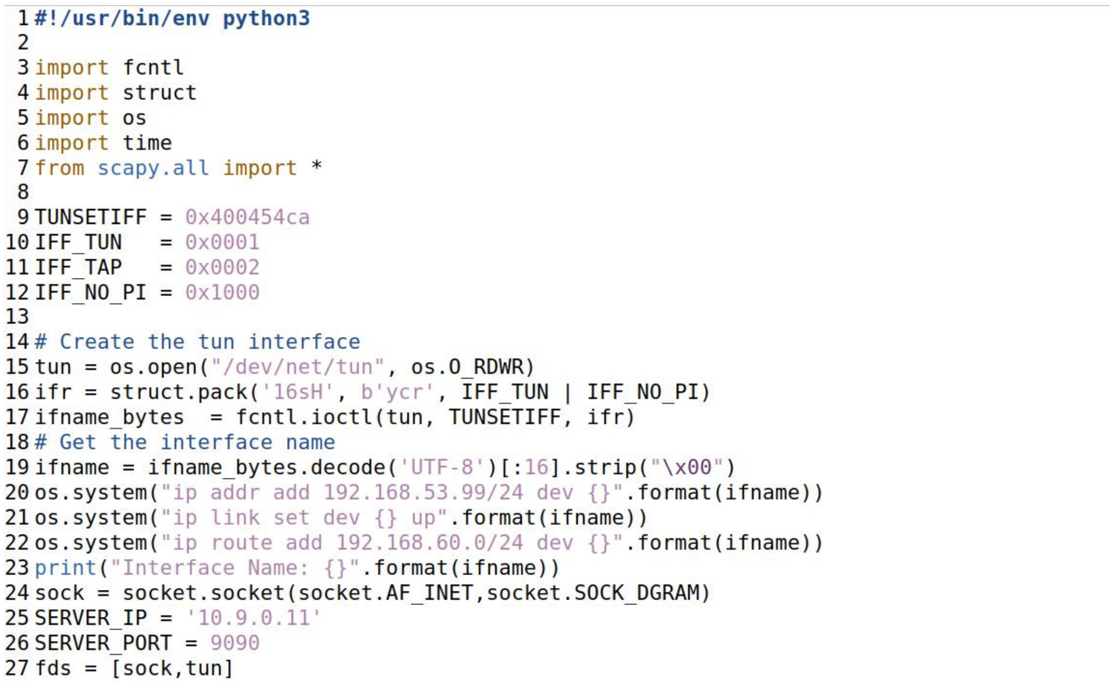
```python
#!/usr/bin/env python3
import fcntl
import struct
import os
import time
from scapy.all import *
TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'ycr0', IFF_TUN | IFF_NO_PI)
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 192.168.53.11/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
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,data)
```

接下来重复上述步骤，在主机 U 上 ping 主机 V，同时在主机 V 上进行 tcpdump，可以看到主机 V 收到了相应的 ICMP request，并发出了 reply：

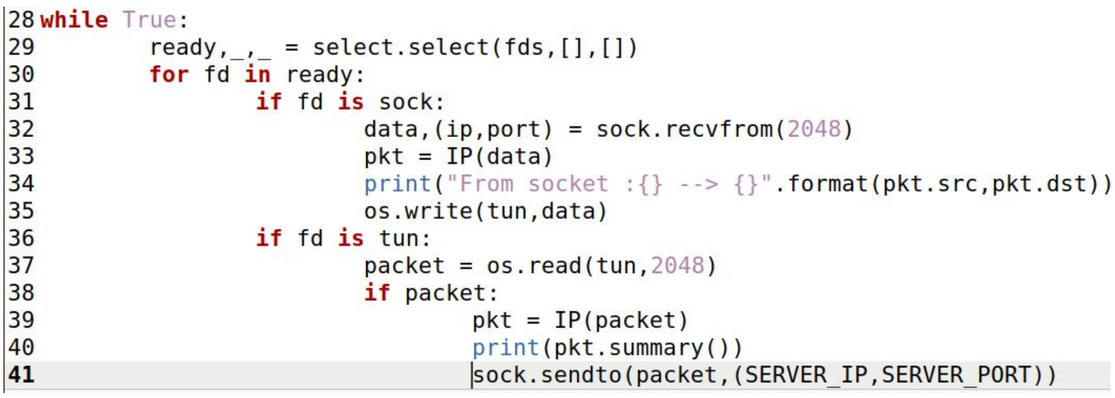
```
root@d8c183cb3c71:/# tcpdump -i eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:53:53.088961 ARP, Request who-has d8c183cb3c71 tell server-router.net-192.168
.60.0, length 28
14:53:53.088979 ARP, Reply d8c183cb3c71 is-at 02:42:c0:a8:3c:05 (oui Unknown), l
ength 28
14:53:53.088989 IP 192.168.53.99 > d8c183cb3c71: ICMP echo request, id 34, seq 1
, length 64
14:53:53.089013 IP d8c183cb3c71 > 192.168.53.99: ICMP echo reply, id 34, seq 1,
length 64
14:53:53.097926 IP d8c183cb3c71.51144 > 192.168.88.2.domain: 26037+ PTR? 99.53.1
68.192.in-addr.arpa. (44)
14:53:54.101449 IP 192.168.53.99 > d8c183cb3c71: ICMP echo request, id 34, seq 2
, length 64
14:53:54.101464 IP d8c183cb3c71 > 192.168.53.99: ICMP echo reply, id 34, seq 2,
length 64
```

6 Task 5: Handling Traffific in Both Directions

修改 tun_client 如下：

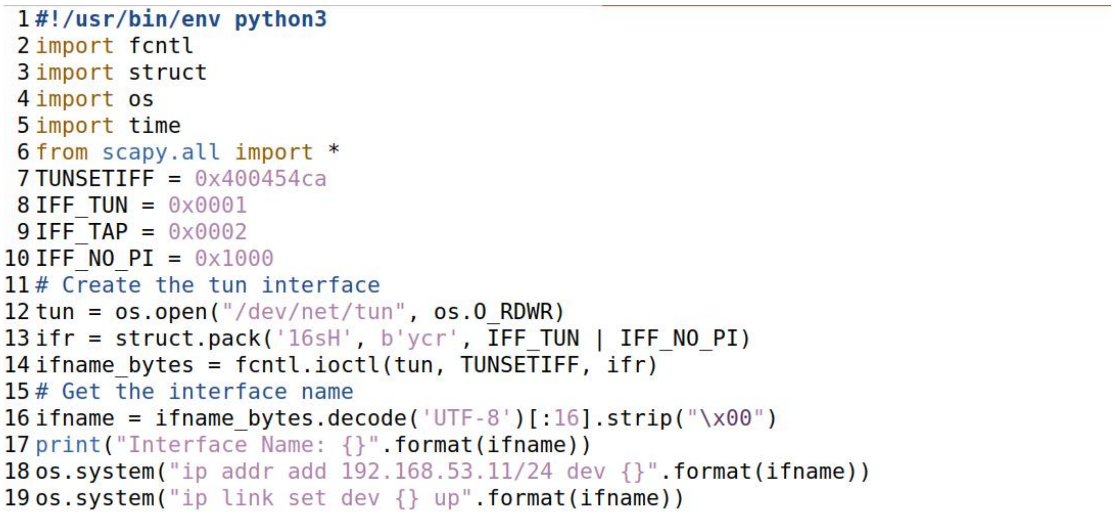


```python
#!/usr/bin/env python3

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'ycr', 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")
os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
os.system("ip route add 192.168.60.0/24 dev {}".format(ifname))
print("Interface Name: {}".format(ifname))
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
SERVER_IP = '10.9.0.11'
SERVER_PORT = 9090
fds = [sock,tun]
```

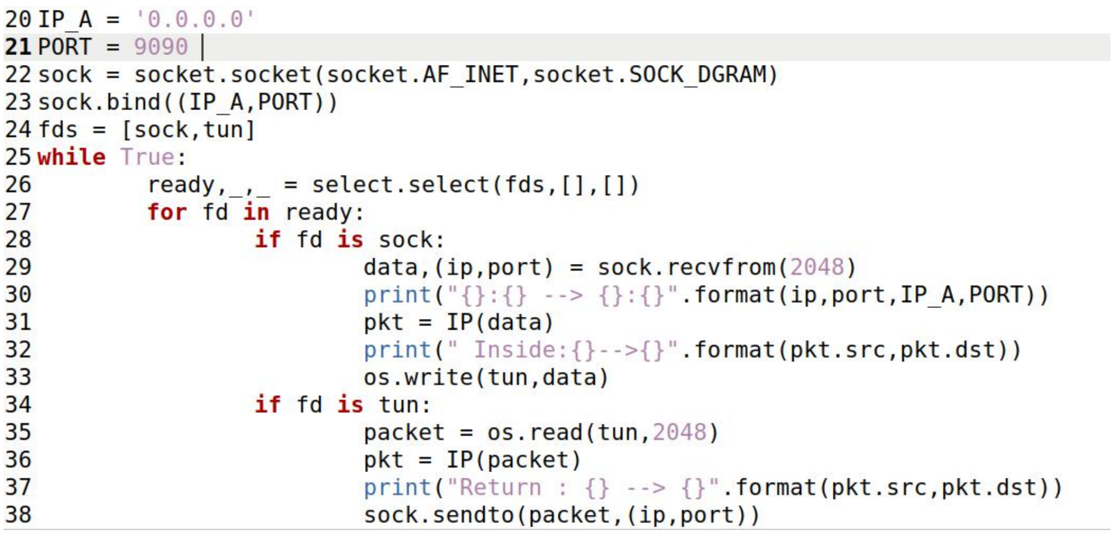


```python
while True:
        ready,_,_ = select.select(fds,[],[])
        for fd in ready:
                if fd is sock:
                        data,(ip,port) = sock.recvfrom(2048)
                        pkt = IP(data)
                        print("From socket :{} --> {}".format(pkt.src,pkt.dst))
                        os.write(tun,data)
                if fd is tun:
                        packet = os.read(tun,2048)
                        if packet:
                                pkt = IP(packet)
                                print(pkt.summary())
                                sock.sendto(packet,(SERVER_IP,SERVER_PORT))
```

修改 tun_server 如下：

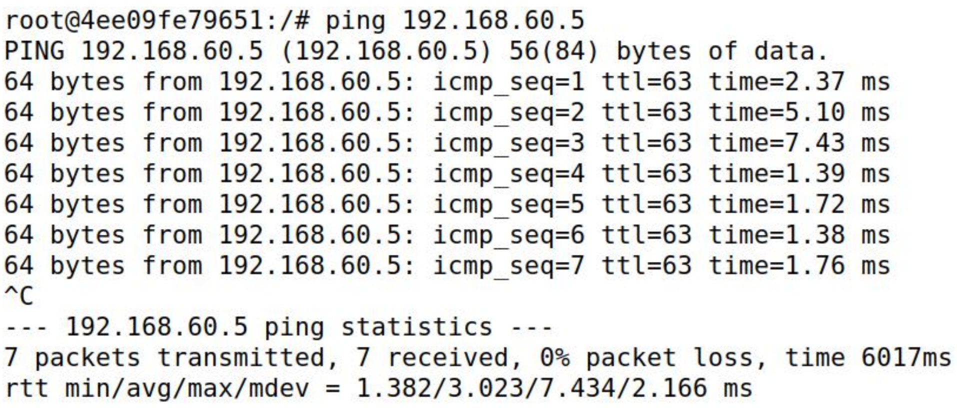


```python
#!/usr/bin/env python3
import fcntl
import struct
import os
import time
from scapy.all import *
TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'ycr', IFF_TUN | IFF_NO_PI)
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 192.168.53.11/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
```



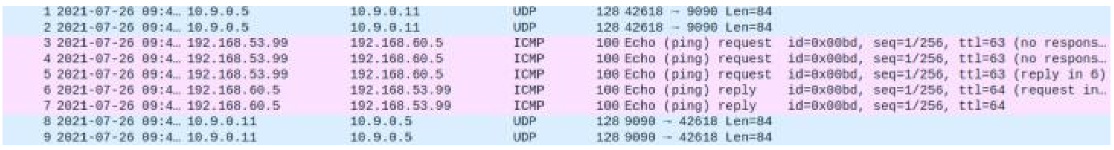
```
IP_A = '0.0.0.0'
PORT = 9090
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
sock.bind((IP_A,PORT))
fds = [sock,tun]
while True:
        ready,_,_ = select.select(fds,[],[])
        for fd in ready:
                if fd is sock:
                        data,(ip,port) = sock.recvfrom(2048)
                        print("{}:{} --> {}:{}".format(ip,port,IP_A,PORT))
                        pkt = IP(data)
                        print(" Inside:{}-->{}".format(pkt.src,pkt.dst))
                        os.write(tun,data)
                if fd is tun:
                        packet = os.read(tun,2048)
                        pkt = IP(packet)
                        print("Return : {} --> {}".format(pkt.src,pkt.dst))
                        sock.sendto(packet,(ip,port))
```

此时进行如上的实验，可以发现可以从主机 U ping 通主机 V：



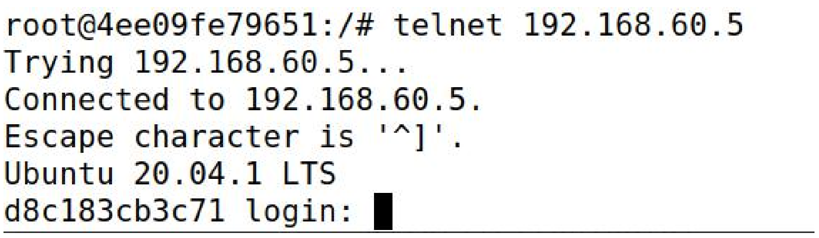
```
root@4ee09fe79651:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=2.37 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=5.10 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=7.43 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=1.39 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=1.72 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=1.38 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=1.76 ms
^C
--- 192.168.60.5 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6017ms
rtt min/avg/max/mdev = 1.382/3.023/7.434/2.166 ms
```

通过 Wireshrk 可以看见更为清晰的 VPN tunneling 过程，先是 10.9.0.5 发送给 10.9.0.11，然后 VPN 变为 192.168.53.99 发往 192.168.60.5，然后再原路径返回：



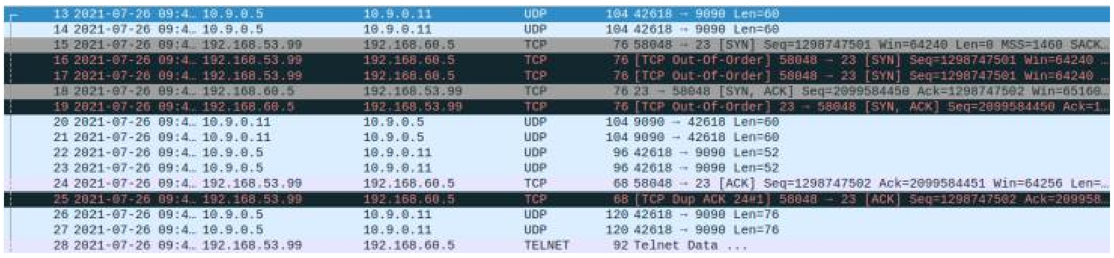
```
1 2021-07-26 09:4… 10.9.0.5        10.9.0.11       UDP     128 42618 → 9090 Len=84
2 2021-07-26 09:4… 10.9.0.5        10.9.0.11       UDP     128 42618 → 9090 Len=84
3 2021-07-26 09:4… 192.168.53.99   192.168.60.5    ICMP    100 Echo (ping) request  id=0x00bd, seq=1/256, ttl=63 (no respons…
4 2021-07-26 09:4… 192.168.53.99   192.168.60.5    ICMP    100 Echo (ping) request  id=0x00bd, seq=1/256, ttl=63 (no respons…
5 2021-07-26 09:4… 192.168.53.99   192.168.60.5    ICMP    100 Echo (ping) request  id=0x00bd, seq=1/256, ttl=63 (reply in 6)
6 2021-07-26 09:4… 192.168.60.5    192.168.53.99   ICMP    100 Echo (ping) reply    id=0x00bd, seq=1/256, ttl=64 (request in…
7 2021-07-26 09:4… 192.168.60.5    192.168.53.99   ICMP    100 Echo (ping) reply    id=0x00bd, seq=1/256, ttl=64
8 2021-07-26 09:4… 10.9.0.11       10.9.0.5        UDP     128 9090 → 42618 Len=84
9 2021-07-26 09:4… 10.9.0.11       10.9.0.5        UDP     128 9090 → 42618 Len=84
```

再次尝试 TELNET 连接，同样成功：



```
root@4ee09fe79651:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
d8c183cb3c71 login: 
```

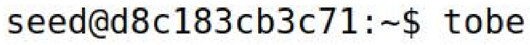
Wireshark 同样可以看见，走的是和 ping 相同的路径，只不过内部变为了 TCP 协议：



```
13 2021-07-26 09:4… 10.9.0.5        10.9.0.11       UDP     104 42618 → 9090 Len=60
14 2021-07-26 09:4… 10.9.0.5        10.9.0.11       UDP     104 42618 → 9090 Len=60
15 2021-07-26 09:4… 192.168.53.99   192.168.60.5    TCP      76 58048 → 23 [SYN] Seq=1298747501 Win=64240 Len=0 MSS=1460 SACK…
16 2021-07-26 09:4… 192.168.53.99   192.168.60.5    TCP      76 [TCP Out-Of-Order] 58048 → 23 [SYN] Seq=1298747501 Win=64240 …
17 2021-07-26 09:4… 192.168.53.99   192.168.60.5    TCP      76 [TCP Out-Of-Order] 58048 → 23 [SYN] Seq=1298747501 Win=64240 …
18 2021-07-26 09:4… 192.168.60.5    192.168.53.99   TCP      76 23 → 58048 [SYN, ACK] Seq=2099584450 Ack=1298747502 Win=65160…
19 2021-07-26 09:4… 192.168.60.5    192.168.53.99   TCP      76 [TCP Out-Of-Order] 23 → 58048 [SYN, ACK] Seq=2099584450 Ack=1…
20 2021-07-26 09:4… 10.9.0.11       10.9.0.5        UDP     104 9090 → 42618 Len=60
21 2021-07-26 09:4… 10.9.0.11       10.9.0.5        UDP     104 9090 → 42618 Len=60
22 2021-07-26 09:4… 10.9.0.5        10.9.0.11       UDP      96 42618 → 9090 Len=52
23 2021-07-26 09:4… 10.9.0.5        10.9.0.11       UDP      96 42618 → 9090 Len=52
24 2021-07-26 09:4… 192.168.53.99   192.168.60.5    TCP      68 58048 → 23 [ACK] Seq=1298747502 Ack=2099584451 Win=64256 Len=…
25 2021-07-26 09:4… 192.168.53.99   192.168.60.5    TCP      68 [TCP Dup ACK 24#1] 58048 → 23 [ACK] Seq=1298747502 Ack=209958…
26 2021-07-26 09:4… 10.9.0.5        10.9.0.11       UDP     120 42618 → 9090 Len=76
27 2021-07-26 09:4… 10.9.0.5        10.9.0.11       UDP     120 42618 → 9090 Len=76
28 2021-07-26 09:4… 192.168.53.99   192.168.60.5    TELNET   92 Telnet Data ...
```

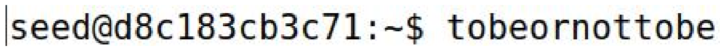
7 Task 6: Tunnel-Breaking Experiment

在 telnet 连接过程中，中断 tun_client.py 程序，此时在主机 V 中的输入没有任何显示：



```
seed@d8c183cb3c71:~$ tobe
```

再次运行 tun_client.py 程序，在中断过程中输入的内容会全部显示出来：



```
seed@d8c183cb3c71:~$ tobeornottobe
```

原因是因为我们的程序是从 tun 接口上来接收和发送报文的，当程序终止时，报文会存储在这些接口的 buffer 上，等待程序运行的时候再来处理。