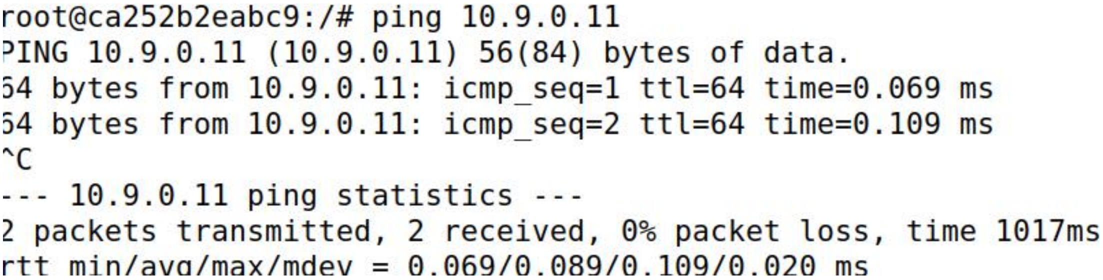
VPN Lab: The Container V ersion

57118224 邱龙

Task 1: Network Setup

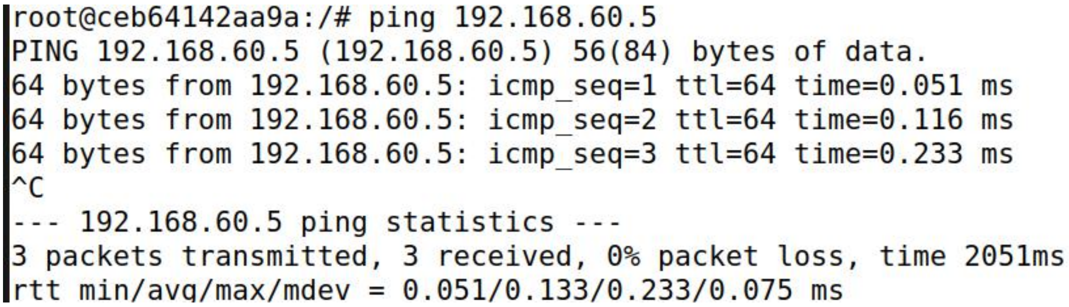
1.在主机 V 上 ping VPN 服务器：



```
root@ca252b2eabc9:/# 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=0.069 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.109 ms
^C
--- 10.9.0.11 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1017ms
rtt min/avg/max/mdev = 0.069/0.089/0.109/0.020 ms
```

正常通信。

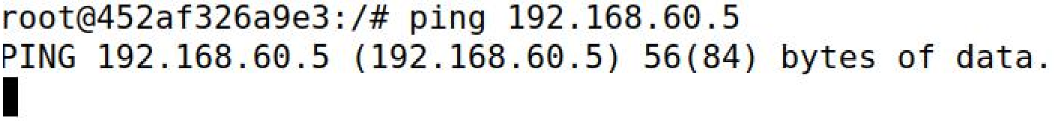
2.在 VPN 服务器上 ping 主机 V:



```
root@ceb64142aa9a:/# 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=0.051 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=64 time=0.116 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=64 time=0.233 ms
^C
--- 192.168.60.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2051ms
rtt min/avg/max/mdev = 0.051/0.133/0.233/0.075 ms
```

正常通信。

3.主机 U 上 ping 主机 V:

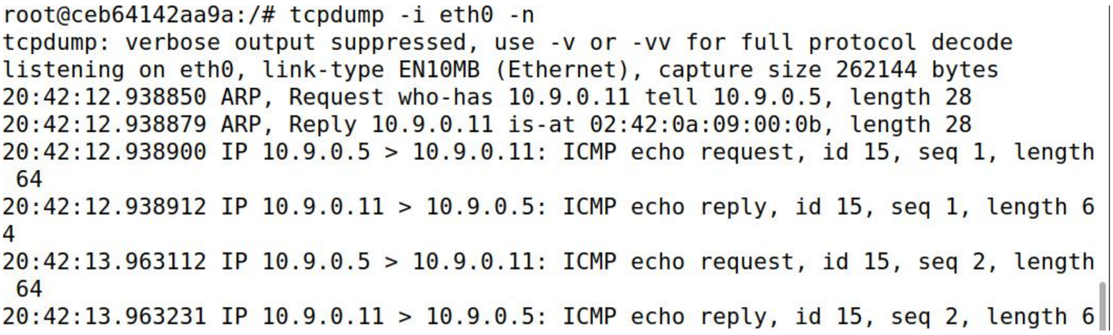


```
root@452af326a9e3:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
```

不能进行通信。

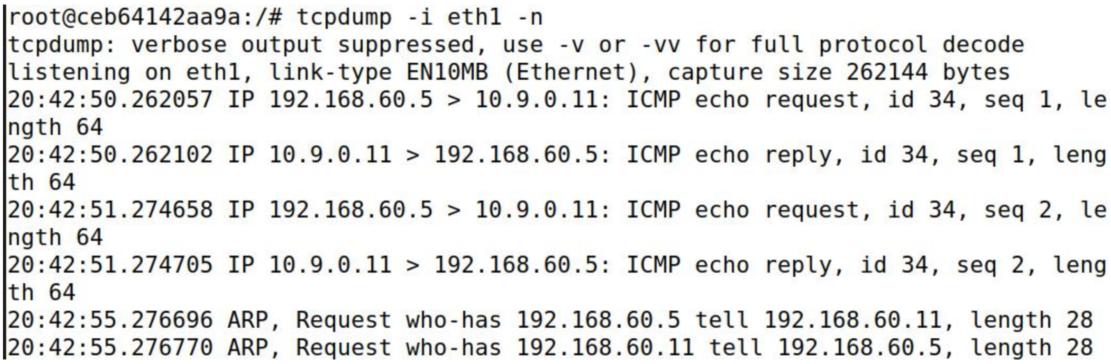
4. 路由器上运行 tcpdump：

嗅探接口 eth0:



```
root@ceb64142aa9a:/# 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
20:42:12.938850 ARP, Request who-has 10.9.0.11 tell 10.9.0.5, length 28
20:42:12.938879 ARP, Reply 10.9.0.11 is-at 02:42:0a:09:00:0b, length 28
20:42:12.938900 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 15, seq 1, length
 64
20:42:12.938912 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 15, seq 1, length 6
4
20:42:13.963112 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 15, seq 2, length
 64
20:42:13.963231 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 15, seq 2, length 6
```

嗅探接口 eth1:



```
root@ceb64142aa9a:/# 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
20:42:50.262057 IP 192.168.60.5 > 10.9.0.11: ICMP echo request, id 34, seq 1, le
ngth 64
20:42:50.262102 IP 10.9.0.11 > 192.168.60.5: ICMP echo reply, id 34, seq 1, leng
th 64
20:42:51.274658 IP 192.168.60.5 > 10.9.0.11: ICMP echo request, id 34, seq 2, le
ngth 64
20:42:51.274705 IP 10.9.0.11 > 192.168.60.5: ICMP echo reply, id 34, seq 2, leng
th 64
20:42:55.276696 ARP, Request who-has 192.168.60.5 tell 192.168.60.11, length 28
20:42:55.276770 ARP, Request who-has 192.168.60.11 tell 192.168.60.5, length 28
```

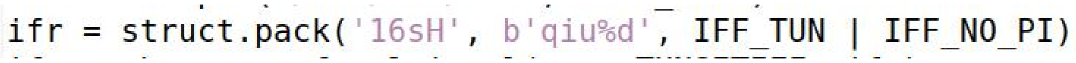
网络流量都可以正常嗅探。

配置正常。

Task 2: Create and Configure TUN Interface

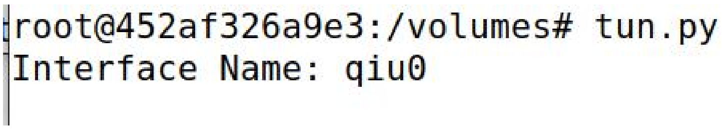
Task 2.a: Name of the Interface

在代码中修改端口名为“qiu”：



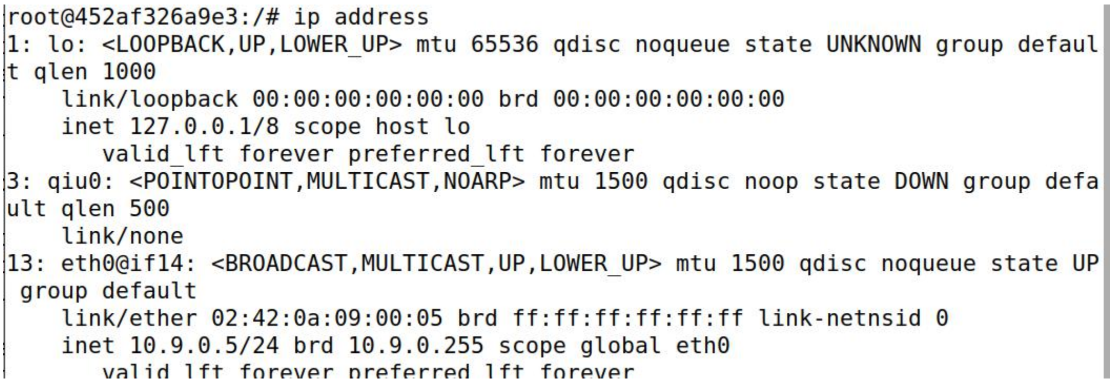
```
ifr = struct.pack('16sH', b'qiu%d', IFF_TUN | IFF_NO_PI)
```

在主机 U 上运行程序：



```
root@452af326a9e3:/volumes# tun.py
Interface Name: qiu0
```

打开另一个终端查看：

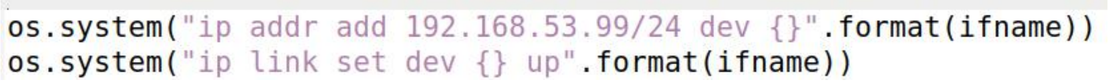


```
root@452af326a9e3:/# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
3: qiu0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group defa
ult qlen 500
    link/none
13: eth0@if14: <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
```

端口名成功修改为 qiu0。

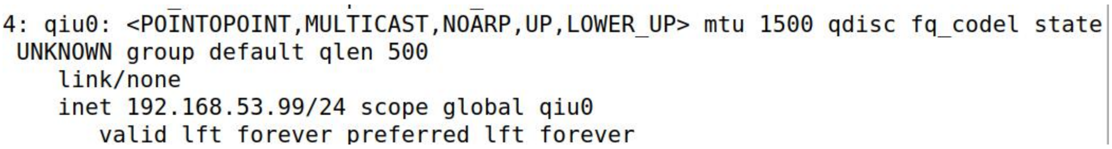
Task 2.b: Set up the TUN Interface

程序中添加两行代码给端口 qiu0 自动分配 ip 地址：



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

再次运行程序，并执行 ip address 命令：

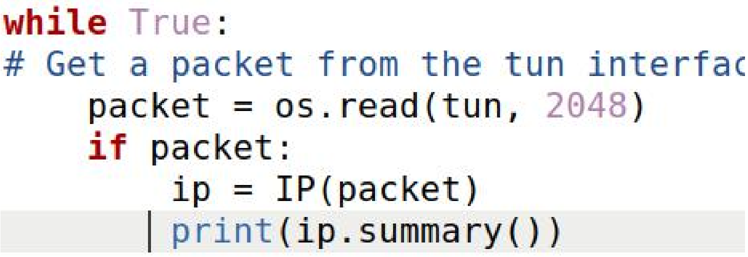


```
4: qiu0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state
 UNKNOWN group default qlen 500
    link/none
    inet 192.168.53.99/24 scope global qiu0
       valid_lft forever preferred_lft forever
```

此时端口已经被成功分配了 ip 地址。

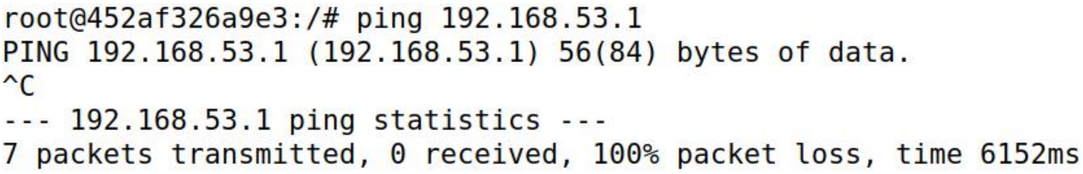
Task 2.c: Read from the TUN Interface

修改程序中的 while 循环：

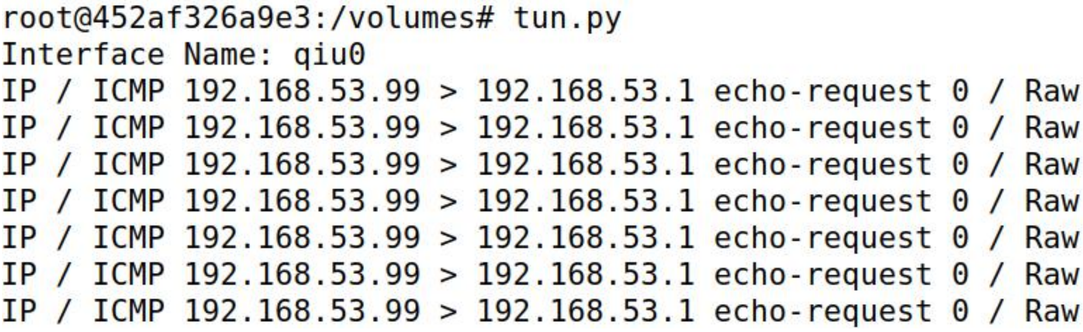


```
while True:
# Get a packet from the tun interfac
    packet = os.read(tun, 2048)
    if packet:
        ip = IP(packet)
        print(ip.summary())
```

再次执行程序。并 ping 192.168.53.0/24 网段中任一主机，这里我们 ping 192.168.53.1：



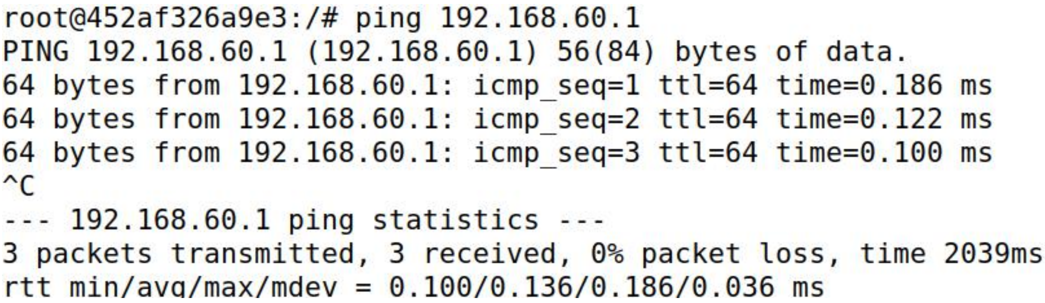
```
root@452af326a9e3:/# ping 192.168.53.1
PING 192.168.53.1 (192.168.53.1) 56(84) bytes of data.
^C
--- 192.168.53.1 ping statistics ---
7 packets transmitted, 0 received, 100% packet loss, time 6152ms
```



```
root@452af326a9e3:/volumes# tun.py
Interface Name: qiu0
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
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
```

此时 ping 不通，从 run.py 程序的输出可以知道 ICMP 请求报文都被端口捕获了，因为发送给 192.168.53.0/24 的数据包是从 qiu0 端口发出。

Ping 192.168.60.1 时：

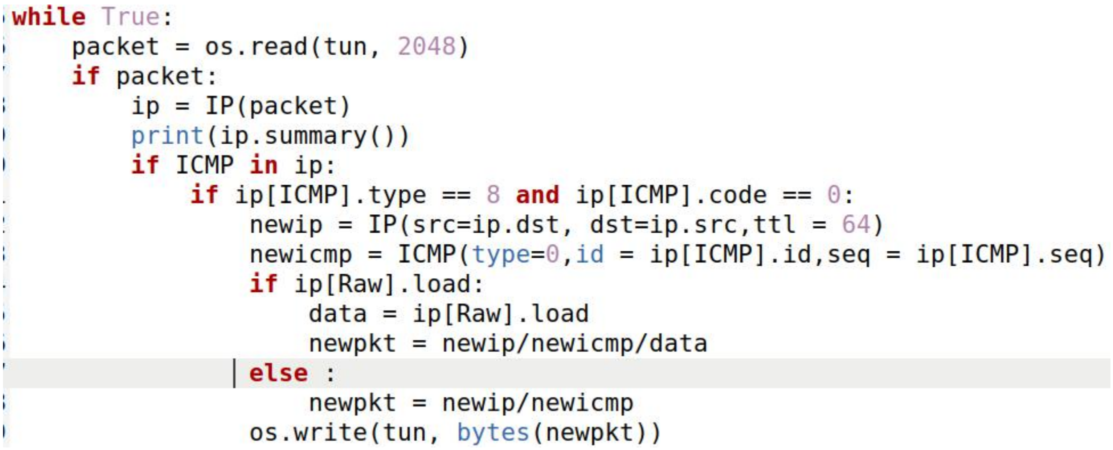


```
root@452af326a9e3:/# ping 192.168.60.1
PING 192.168.60.1 (192.168.60.1) 56(84) bytes of data.
64 bytes from 192.168.60.1: icmp_seq=1 ttl=64 time=0.186 ms
64 bytes from 192.168.60.1: icmp_seq=2 ttl=64 time=0.122 ms
64 bytes from 192.168.60.1: icmp_seq=3 ttl=64 time=0.100 ms
^C
--- 192.168.60.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2039ms
rtt min/avg/max/mdev = 0.100/0.136/0.186/0.036 ms
```

此时能 ping 通，且此时程序没有输出。这是因为发送给 192.168.60.1 的报文不经过 qiu0 端口，所以没有捕获报文。

Task 2.d: Write to the TUN Interface

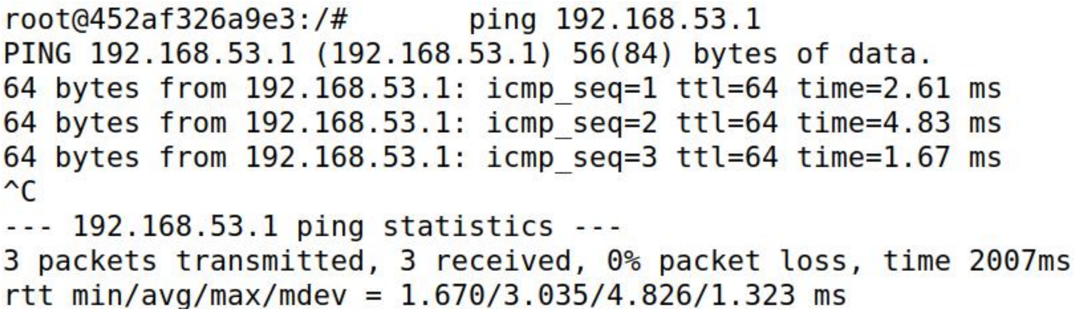
修改 while 循环如下：



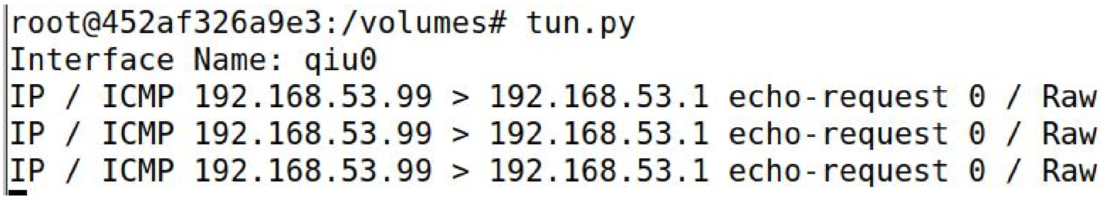
```
while True:
    packet = os.read(tun, 2048)
    if packet:
        ip = IP(packet)
        print(ip.summary())
        if ICMP in ip:
            if ip[ICMP].type == 8 and ip[ICMP].code == 0:
                newip = IP(src=ip.dst, dst=ip.src,ttl = 64)
                newicmp = ICMP(type=0,id = ip[ICMP].id,seq = ip[ICMP].seq)
                if ip[Raw].load:
                    data = ip[Raw].load
                    newpkt = newip/newicmp/data
                else :
                    newpkt = newip/newicmp
                os.write(tun, bytes(newpkt))
```

这里判断是否为 Echo request 包，然后将请求包源地址和目的地址交换，构造响应包，负载为原来数据包的负载。

运行程序，然后再次 ping 192.168.53.1：



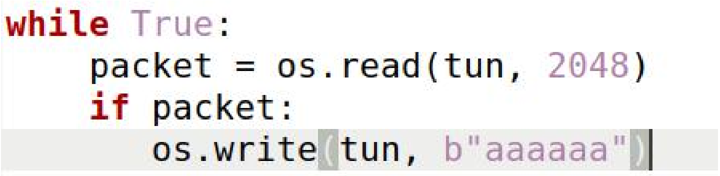
```
root@452af326a9e3:/#       ping 192.168.53.1
PING 192.168.53.1 (192.168.53.1) 56(84) bytes of data.
64 bytes from 192.168.53.1: icmp_seq=1 ttl=64 time=2.61 ms
64 bytes from 192.168.53.1: icmp_seq=2 ttl=64 time=4.83 ms
64 bytes from 192.168.53.1: icmp_seq=3 ttl=64 time=1.67 ms
^C
--- 192.168.53.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2007ms
rtt min/avg/max/mdev = 1.670/3.035/4.826/1.323 ms
```



```
root@452af326a9e3:/volumes# tun.py
Interface Name: qiu0
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
```

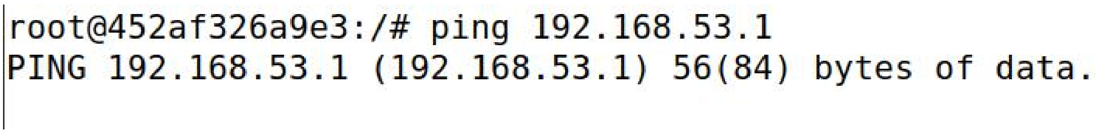
此时能够 ping 通，说明我们伪造响应包成功。

修改 while 循环，不写入 ip 数据包而是任意数据：

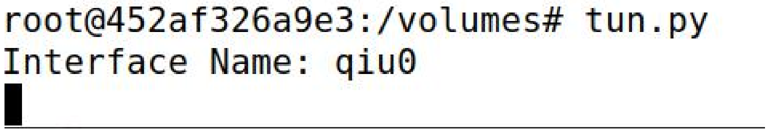


```
while True:
    packet = os.read(tun, 2048)
    if packet:
        os.write(tun, b"aaaaaa")
```

运行程序，然后再次 ping 192.168.53.1：

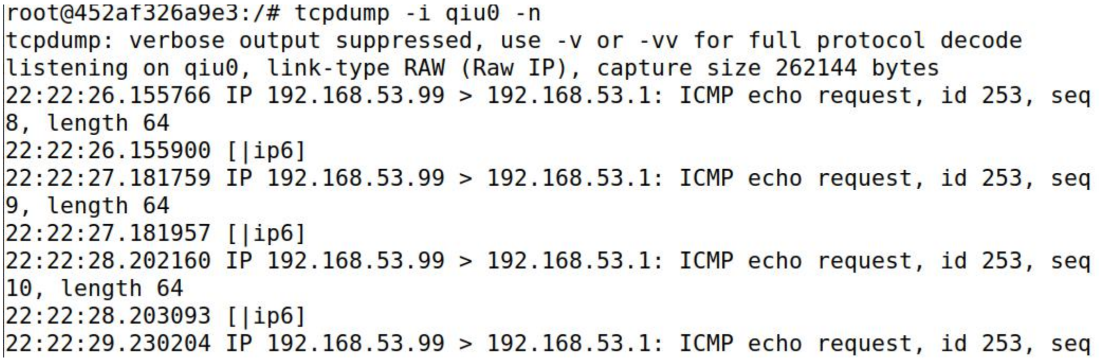


```
root@452af326a9e3:/# ping 192.168.53.1
PING 192.168.53.1 (192.168.53.1) 56(84) bytes of data.
```



```
root@452af326a9e3:/volumes# tun.py
Interface Name: qiu0
```

Ping 不通，且程序无输出，执行 tcpdump -i qiu0 -n 命令：

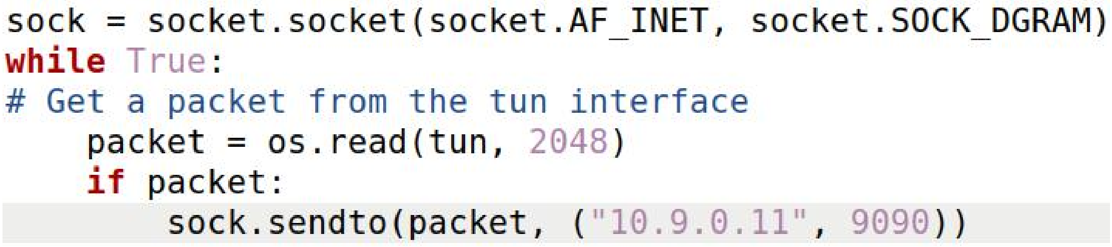


```
root@452af326a9e3:/# tcpdump -i qiu0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on qiu0, link-type RAW (Raw IP), capture size 262144 bytes
22:22:26.155766 IP 192.168.53.99 > 192.168.53.1: ICMP echo request, id 253, seq
8, length 64
22:22:26.155900 [|ip6]
22:22:27.181759 IP 192.168.53.99 > 192.168.53.1: ICMP echo request, id 253, seq
9, length 64
22:22:27.181957 [|ip6]
22:22:28.202160 IP 192.168.53.99 > 192.168.53.1: ICMP echo request, id 253, seq
10, length 64
22:22:28.203093 [|ip6]
22:22:29.230204 IP 192.168.53.99 > 192.168.53.1: ICMP echo request, id 253, seq
```

可以看到发送的任意数据确实发送出去了，但是其不符合报文格式，没有什么用。

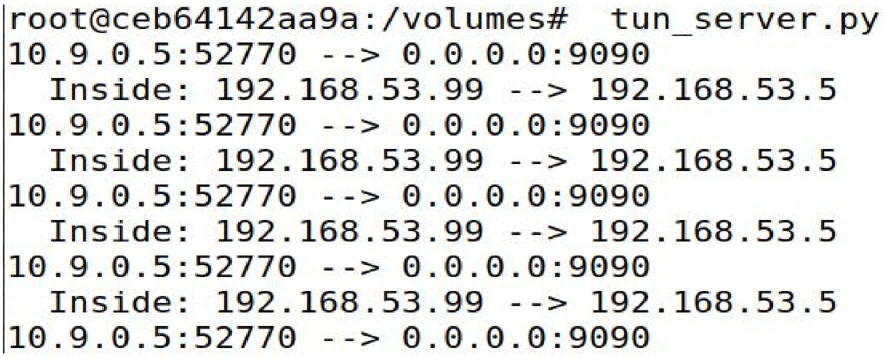
Task 3: Send the IP Packet to VPN Server Through a Tunnel

将 run.py 程序的 while 循环修改为如下代码即为 tun_client.py：



```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
while True:
# Get a packet from the tun interface
      packet = os.read(tun, 2048)
      if packet:
            sock.sendto(packet, ("10.9.0.11", 9090))
```

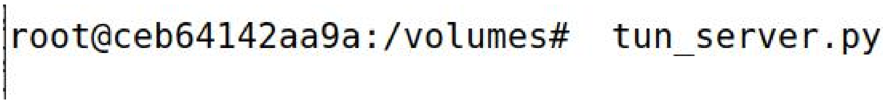
在主机 U 上运行 tun_client.py，在 VPN 服务器上运行 tun_server.py。然后在主机 U 中 ping 192.168.53.5，VPN 服务器输出如下：



```
root@ceb64142aa9a:/volumes#  tun_server.py
10.9.0.5:52770 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:52770 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:52770 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:52770 --> 0.0.0.0:9090
   Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:52770 --> 0.0.0.0:9090
```

此时 VPN 服务器成功捕获到了报文。这是因为 tun_client.py 程序将捕获的报文发给了 VPN 服务器的 9090 端口。

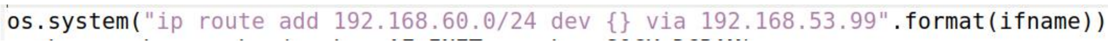
在主机 U 上 Ping 主机 V:



```
root@ceb64142aa9a:/volumes#  tun_server.py
```

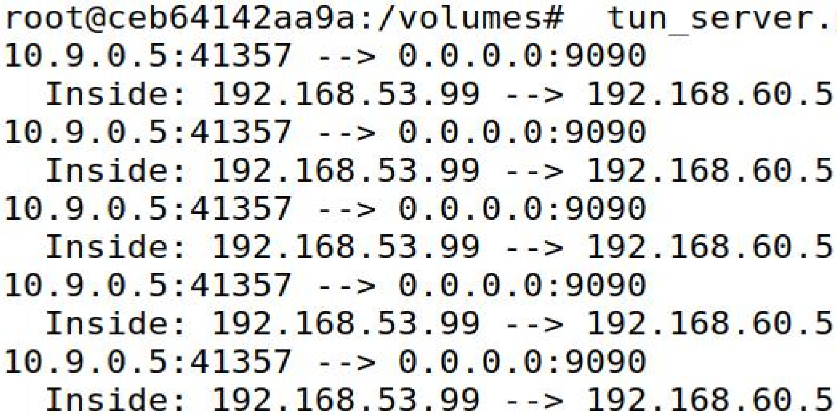
此时 VPN 服务器没有输出，这是因为此时主机 U 上没有去往 192.168.60.0/24 的路由，报文不会从 tun 端口发出。

在 tun_client.py 中添加如下代码用于自动配置路由：



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

重复操作：

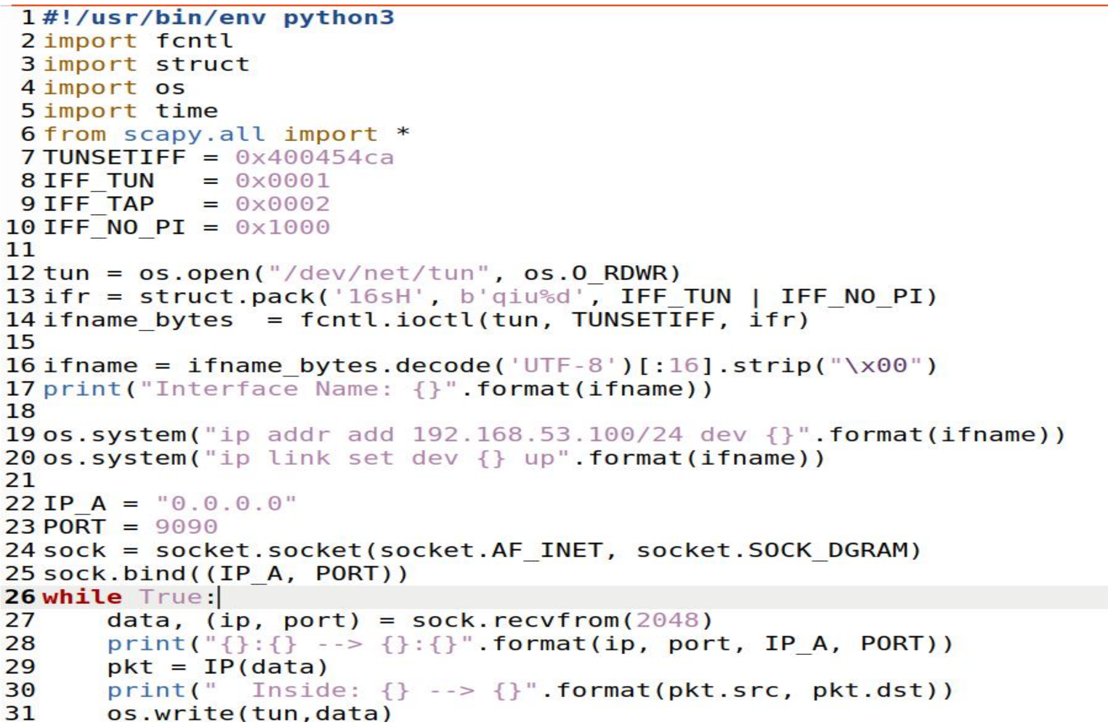


```
root@ceb64142aa9a:/volumes#  tun_server.py
10.9.0.5:41357 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:41357 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:41357 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:41357 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:41357 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.60.5
```

此时 VPN 服务器有输出，说明 tun_server.py 通过隧道接收到报文，实验成功。

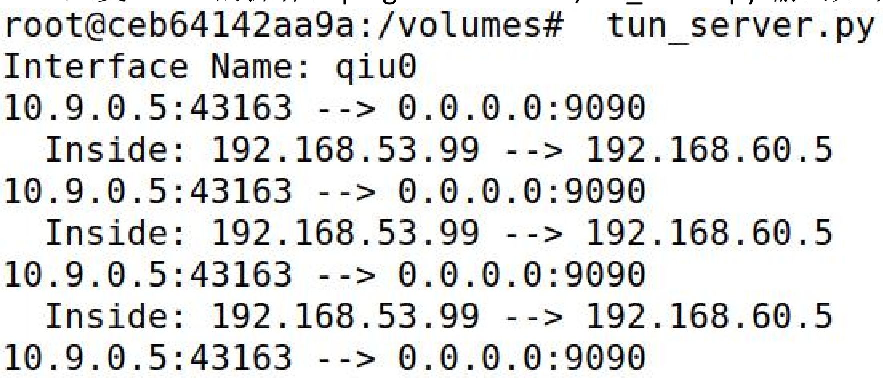
Task 4: Set Up the VPN Server

修改 tun_server.py 代码使得建立一个 tun 接口将数据包路由到最终目的地，增加的代码类似于 task2：



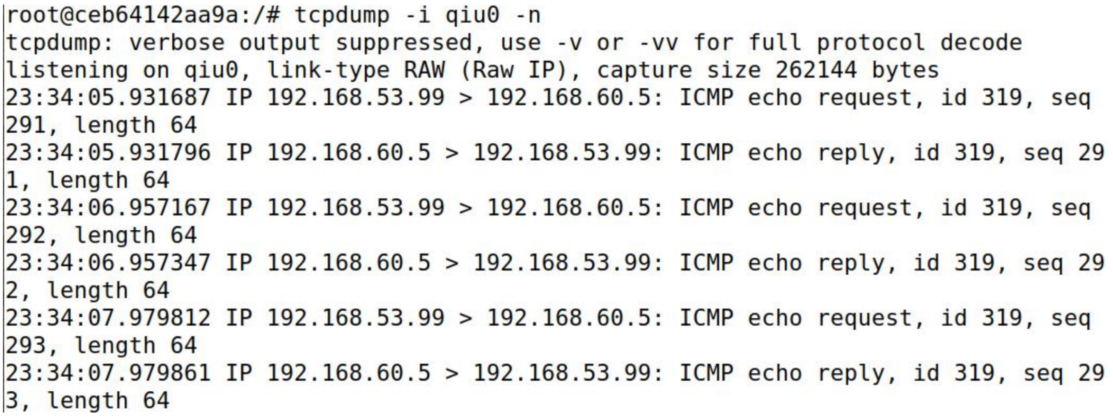
```python
#!/usr/bin/env python3
import fcntl
import struct
import os
import time
from scapy.all import *
TUNSETIFF = 0x400454ca
IFF_TUN   = 0x0001
IFF_TAP   = 0x0002
IFF_NO_PI = 0x1000

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)

ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface Name: {}".format(ifname))

os.system("ip addr add 192.168.53.100/24 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)
```

重复 task3 的操作，ping 192.168.60.5,tun_server.py 输出如下：



```
root@ceb64142aa9a:/volumes#  tun_server.py
Interface Name: qiu0
10.9.0.5:43163 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:43163 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:43163 --> 0.0.0.0:9090
  Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:43163 --> 0.0.0.0:9090
```

我们可以在 VPN 服务器上嗅探 qiu0 端口：



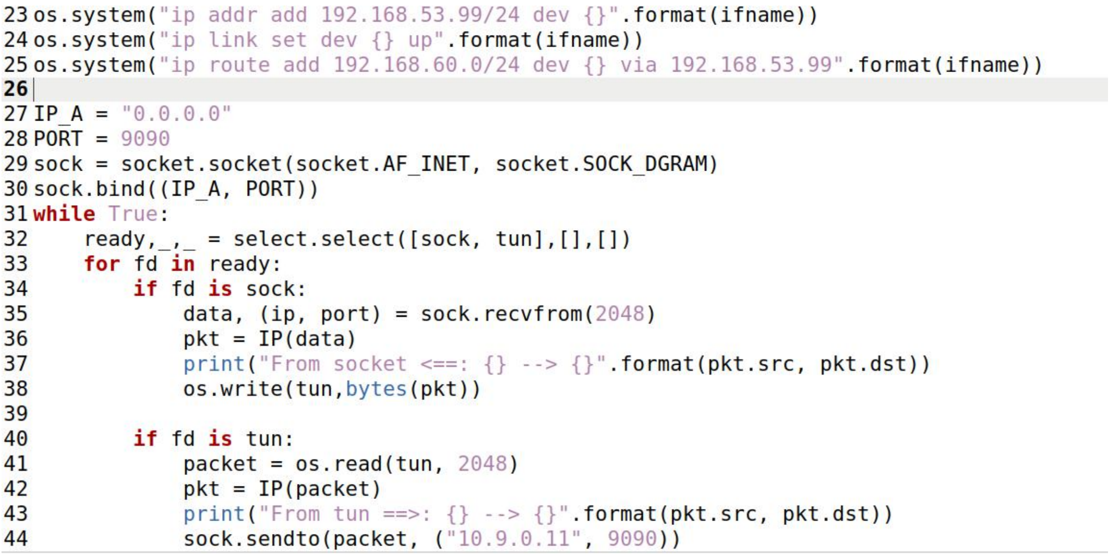
```
root@ceb64142aa9a:/# tcpdump -i qiu0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on qiu0, link-type RAW (Raw IP), capture size 262144 bytes
23:34:05.931687 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 319, seq
291, length 64
23:34:05.931796 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 319, seq 29
1, length 64
23:34:06.957167 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 319, seq
292, length 64
23:34:06.957347 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 319, seq 29
2, length 64
23:34:07.979812 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 319, seq
293, length 64
23:34:07.979861 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 319, seq 29
3, length 64
```

发现 ICMP 请求包成功通过隧道到达主机 V,且受到了主机 V 的 ICMP 响应包。但是此时还没有设置完成，此时隧道只有一个方向，故响应包无法到达主机 U。

Task 5: Handling Traffic in Both Directions

为建立另一个方向的隧道，我们修改代码中的 while 部分：

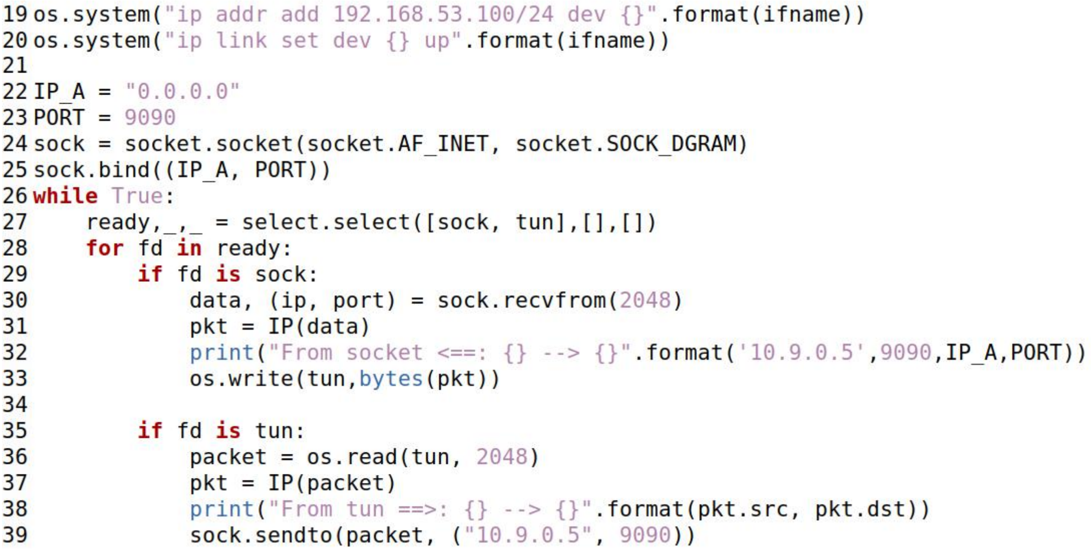
tun_client.py:



```
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 {} via 192.168.53.99".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:
    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))
            os.write(tun,bytes(pkt))

        if fd is tun:
            packet = os.read(tun, 2048)
            pkt = IP(packet)
            print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))
            sock.sendto(packet, ("10.9.0.11", 9090))
```

如果数据包来自 tun 接口，则发给主机 U，如果数据包来自 socket 接口，则发给隧道。

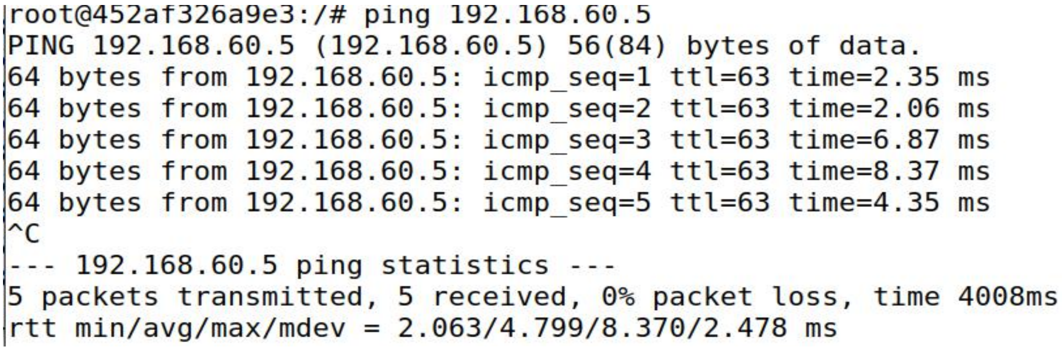
Tun_server.py:



```
os.system("ip addr add 192.168.53.100/24 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:
    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('10.9.0.5',9090,IP_A,PORT))
            os.write(tun,bytes(pkt))

        if fd is tun:
            packet = os.read(tun, 2048)
            pkt = IP(packet)
            print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))
            sock.sendto(packet, ("10.9.0.5", 9090))
```

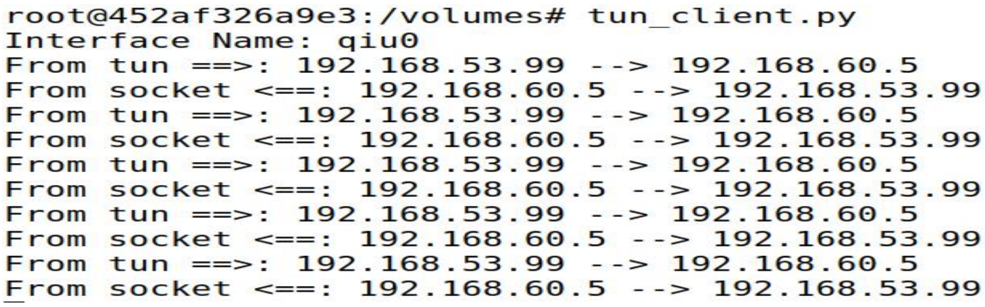
如果数据包来自 tun 接口，则发给主机 V，如果数据包来自 socket 接口，则发给隧道。

重复之前的操作, ping 192.168.60.5:

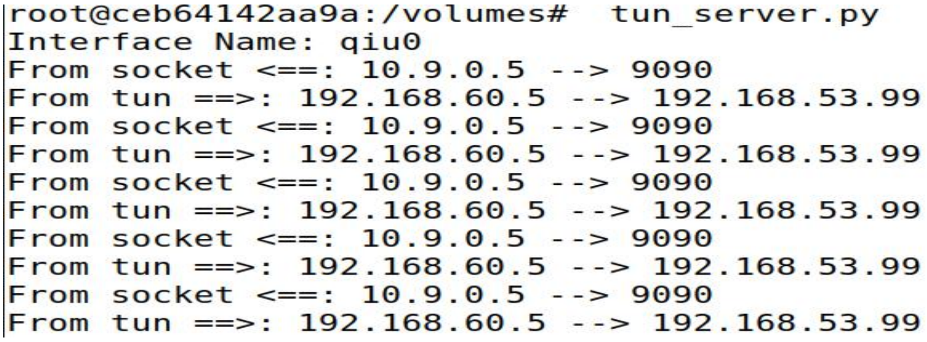


```
root@452af326a9e3:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=2.35 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=2.06 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=6.87 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=8.37 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=4.35 ms
^C
--- 192.168.60.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4008ms
rtt min/avg/max/mdev = 2.063/4.799/8.370/2.478 ms
```

成功 ping 通。Tun_client.py 和 tun_server.py 输出如下：

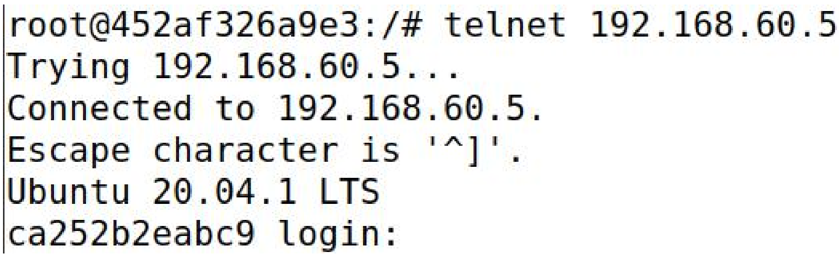


```
root@452af326a9e3:/volumes# tun_client.py
Interface Name: qiu0
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
```



```
root@ceb64142aa9a:/volumes#  tun_server.py
Interface Name: qiu0
From socket <==: 10.9.0.5 --> 9090
From tun ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 10.9.0.5 --> 9090
From tun ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 10.9.0.5 --> 9090
From tun ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 10.9.0.5 --> 9090
From tun ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 10.9.0.5 --> 9090
From tun ==>: 192.168.60.5 --> 192.168.53.99
```

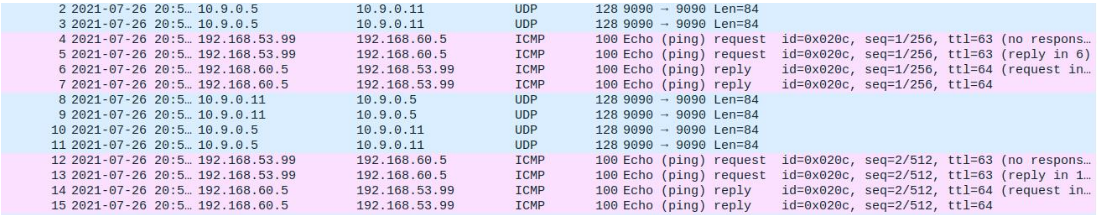
建立 Telnet 连接：



```
root@452af326a9e3:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
ca252b2eabc9 login:
```

Telnet 连接也成功建立。

捕获 ping 过程中的数据包，查看 wireshark：

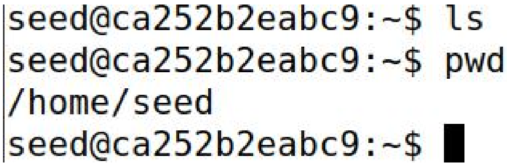


```
 2 2021-07-26 20:5… 10.9.0.5        10.9.0.11       UDP     128 9090 → 9090 Len=84
 3 2021-07-26 20:5… 10.9.0.5        10.9.0.11       UDP     128 9090 → 9090 Len=84
 4 2021-07-26 20:5… 192.168.53.99   192.168.60.5    ICMP    100 Echo (ping) request  id=0x020c, seq=1/256, ttl=63 (no respons…
 5 2021-07-26 20:5… 192.168.53.99   192.168.60.5    ICMP    100 Echo (ping) request  id=0x020c, seq=1/256, ttl=63 (reply in 6)
 6 2021-07-26 20:5… 192.168.60.5    192.168.53.99   ICMP    100 Echo (ping) reply    id=0x020c, seq=1/256, ttl=64 (request in…
 7 2021-07-26 20:5… 192.168.60.5    192.168.53.99   ICMP    100 Echo (ping) reply    id=0x020c, seq=1/256, ttl=64
 8 2021-07-26 20:5… 10.9.0.11       10.9.0.5        UDP     128 9090 → 9090 Len=84
 9 2021-07-26 20:5… 10.9.0.11       10.9.0.5        UDP     128 9090 → 9090 Len=84
10 2021-07-26 20:5… 10.9.0.5        10.9.0.11       UDP     128 9090 → 9090 Len=84
11 2021-07-26 20:5… 10.9.0.5        10.9.0.11       UDP     128 9090 → 9090 Len=84
12 2021-07-26 20:5… 192.168.53.99   192.168.60.5    ICMP    100 Echo (ping) request  id=0x020c, seq=2/512, ttl=63 (no respons…
13 2021-07-26 20:5… 192.168.53.99   192.168.60.5    ICMP    100 Echo (ping) request  id=0x020c, seq=2/512, ttl=63 (reply in 1…
14 2021-07-26 20:5… 192.168.60.5    192.168.53.99   ICMP    100 Echo (ping) reply    id=0x020c, seq=2/512, ttl=64 (request in…
15 2021-07-26 20:5… 192.168.60.5    192.168.53.99   ICMP    100 Echo (ping) reply    id=0x020c, seq=2/512, ttl=64
```

数据报文从主机 U 发向主机 V，报文先通过 tun 到达 VPN 服务器，然后 VPN 服务器通过 tun 发往主机 V 报文，然后主机 V 返回响应报文通过 tun 达到 VPN 服务器，VPN 服务器又通过 tun 将响应报文发给主机 U，从而完成主机 U 和主机 V 之间的通信。

Task 6: Tunnel-Breaking Experiment

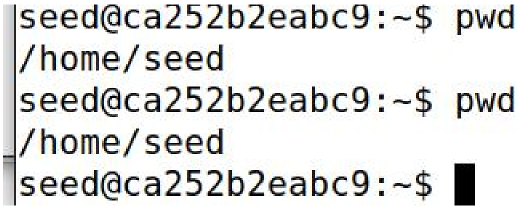
主机 U 向主机 V 建立 Telnet 连接，然后终止程序，发现无法输入任何字符：



```
seed@ca252b2eabc9:~$ ls
seed@ca252b2eabc9:~$ pwd
/home/seed
seed@ca252b2eabc9:~$ █
```

这是因为停止程序后隧道中断，数据包无法到达。

短时间内再次执行程序：



```
seed@ca252b2eabc9:~$ pwd
/home/seed
seed@ca252b2eabc9:~$ pwd
/home/seed
seed@ca252b2eabc9:~$ █
```

如果此时很快地执行程序恢复隧道，会发现前面中断程序时没能显示的输入会再次显示，Telnet 连接恢复。因为断开程序时的输入会在缓存区中一直发送报文，如果恢复连接比较快速，前面的输入仍然会显示。但是较长时间还没再次执行程序就不能显示之前的输入了。