网络空间安全课程综合设计任务报告七

57117203　姜舒
2020 年 9 月 28 日

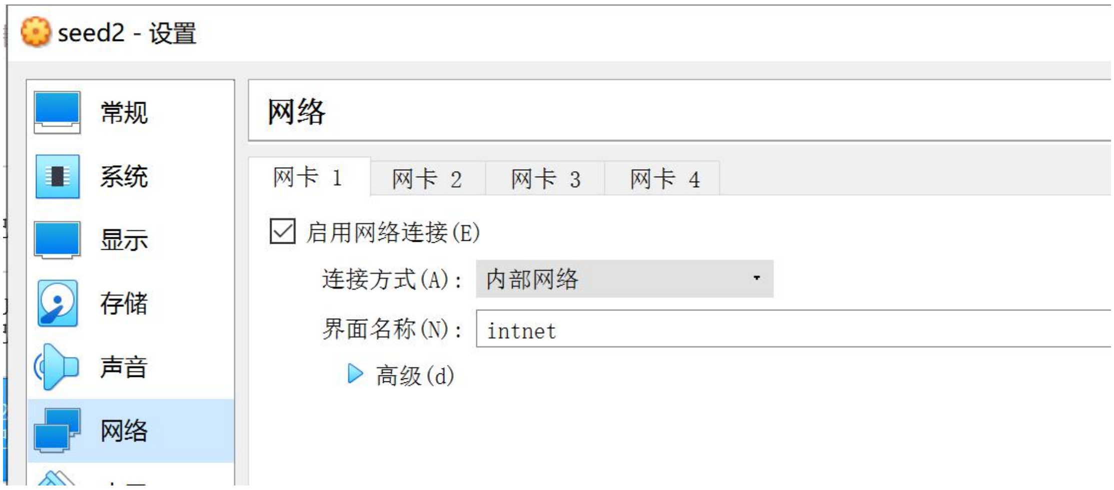
（一）VPN Tunneling Lab

Task 1: Network Setup

各虚拟机 IP 地址：

VPN Client/Host U	虚拟机 VA	10.0.2.4
Host V	虚拟机 VB	192.168.70.101
VPN Server/Gateway	虚拟机 VC	10.0.2.6

将主机 V 的网络设置改为内部网络


seed2 - 设置

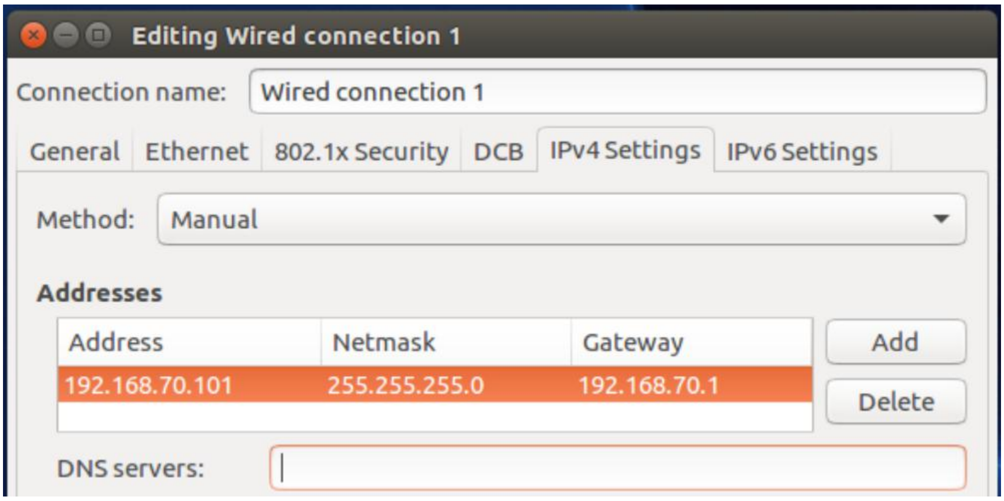
网络

☑ 启用网络连接(E)

连接方式(A)： 内部网络

界面名称(N)： intnet

修改主机 V 的网络设置


Editing Wired connection 1

Connection name: Wired connection 1

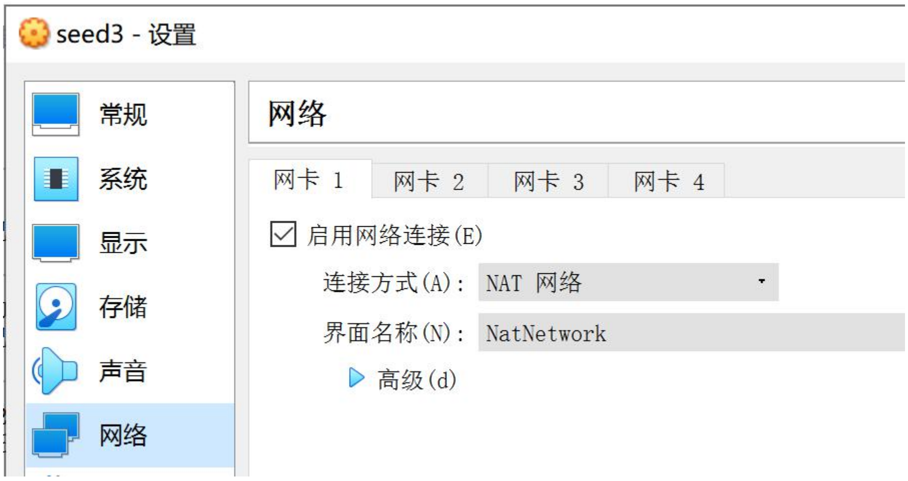
Method: Manual

Addresses

Address	Netmask	Gateway
192.168.70.101	255.255.255.0	192.168.70.1

DNS servers:

将 VPN Server 的网络设置修改，网卡 1 不变，网卡 2 改为内部网络

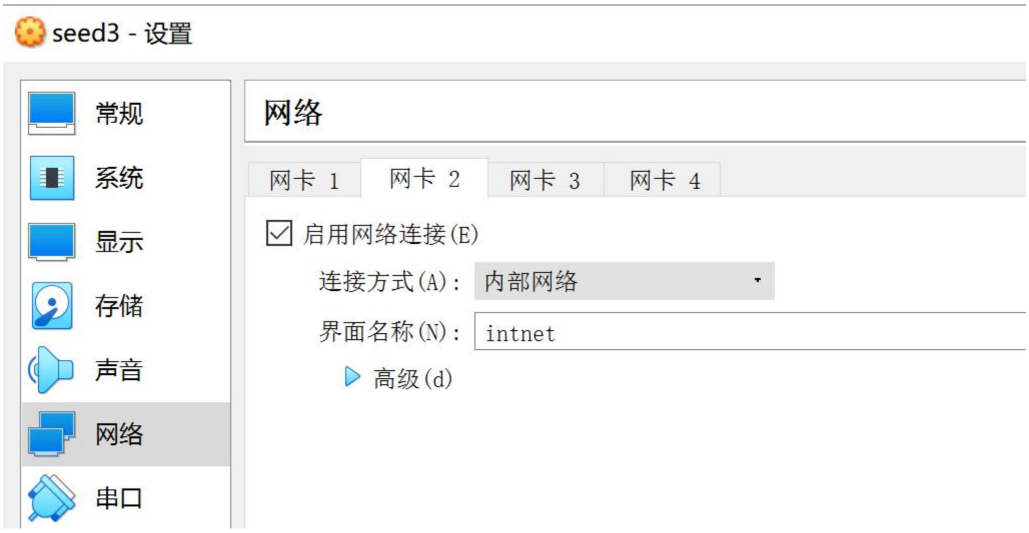


网络

☑ 启用网络连接(E)

连接方式(A)：NAT 网络

界面名称(N)：NatNetwork



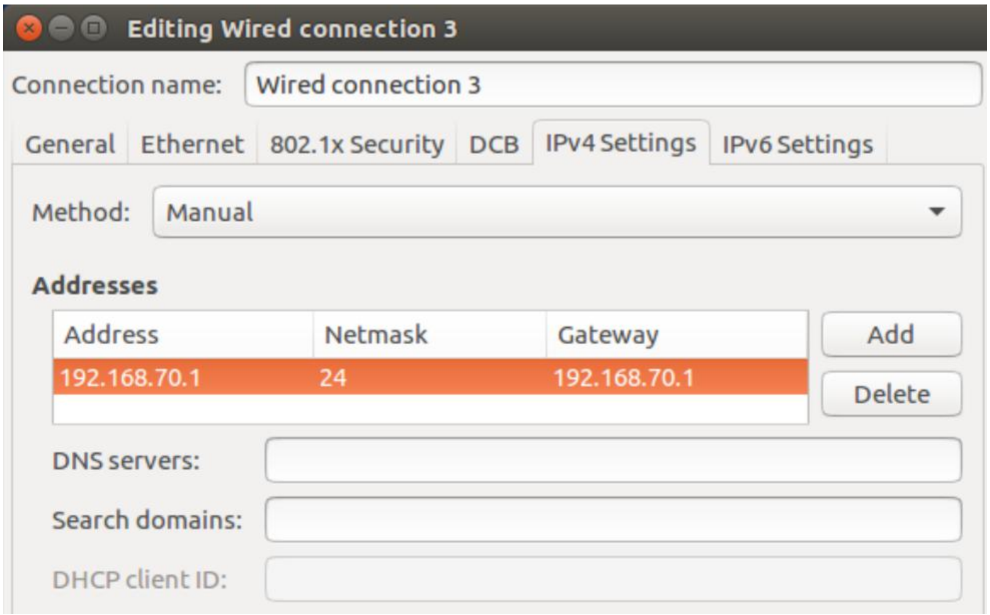
网络

☑ 启用网络连接(E)

连接方式(A)：内部网络

界面名称(N)：intnet

修改 VPN Server 的网络连接设置，将 IP 地址改为 192.168.70.1



Connection name: Wired connection 3

Method: Manual

Addresses

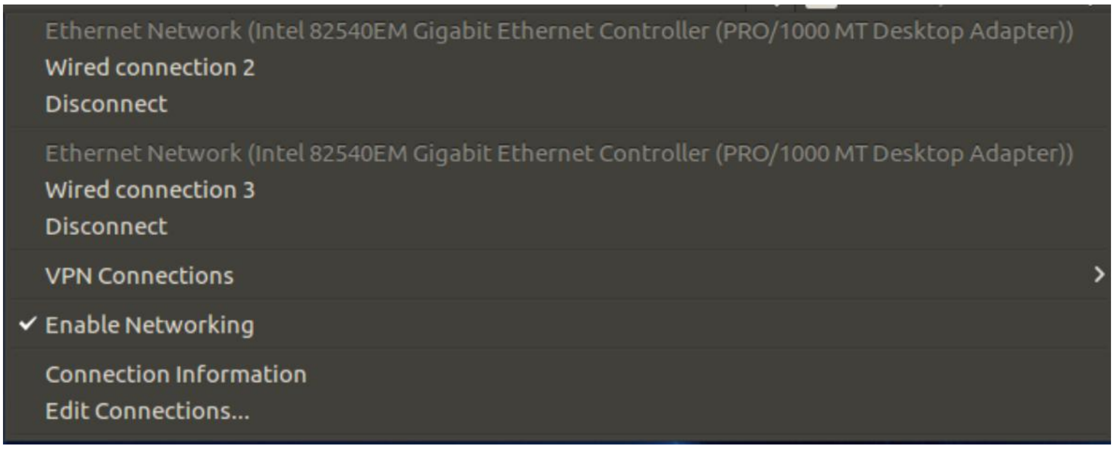
Address	Netmask	Gateway
192.168.70.1	24	192.168.70.1

DNS servers:

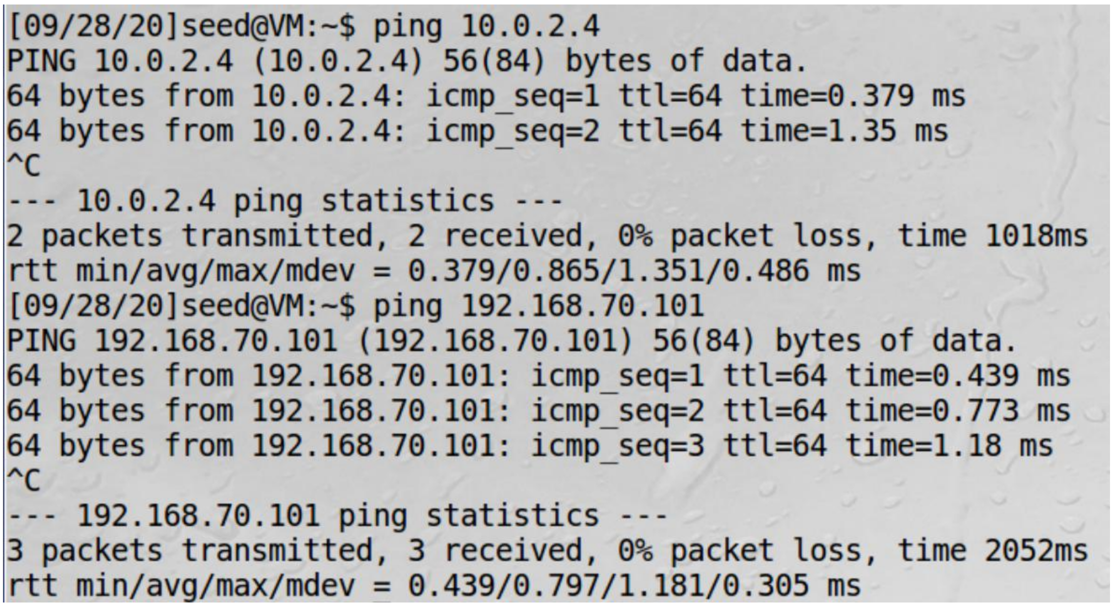
Search domains:

DHCP client ID:

此时主机 VPN Server 有两个网络连接

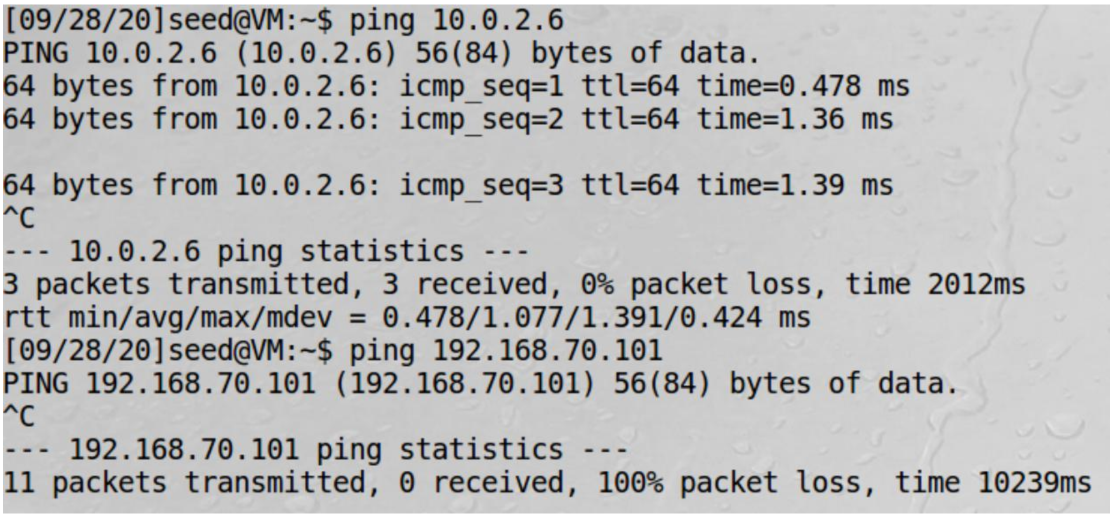

Ethernet Network (Intel 82540EM Gigabit Ethernet Controller (PRO/1000 MT Desktop Adapter))
Wired connection 2
Disconnect
Ethernet Network (Intel 82540EM Gigabit Ethernet Controller (PRO/1000 MT Desktop Adapter))
Wired connection 3
Disconnect
VPN Connections
✓ Enable Networking
Connection Information
Edit Connections...

测试各个主机之间的连接情况。
VPN Server 可以同时 ping 通主机 U 和主机 V



```
[09/28/20]seed@VM:~$ ping 10.0.2.4
PING 10.0.2.4 (10.0.2.4) 56(84) bytes of data.
64 bytes from 10.0.2.4: icmp_seq=1 ttl=64 time=0.379 ms
64 bytes from 10.0.2.4: icmp_seq=2 ttl=64 time=1.35 ms
^C
--- 10.0.2.4 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1018ms
rtt min/avg/max/mdev = 0.379/0.865/1.351/0.486 ms
[09/28/20]seed@VM:~$ ping 192.168.70.101
PING 192.168.70.101 (192.168.70.101) 56(84) bytes of data.
64 bytes from 192.168.70.101: icmp_seq=1 ttl=64 time=0.439 ms
64 bytes from 192.168.70.101: icmp_seq=2 ttl=64 time=0.773 ms
64 bytes from 192.168.70.101: icmp_seq=3 ttl=64 time=1.18 ms
^C
--- 192.168.70.101 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2052ms
rtt min/avg/max/mdev = 0.439/0.797/1.181/0.305 ms
```

主机 U 可以 ping 通 VPN Server，但无法连接主机 V

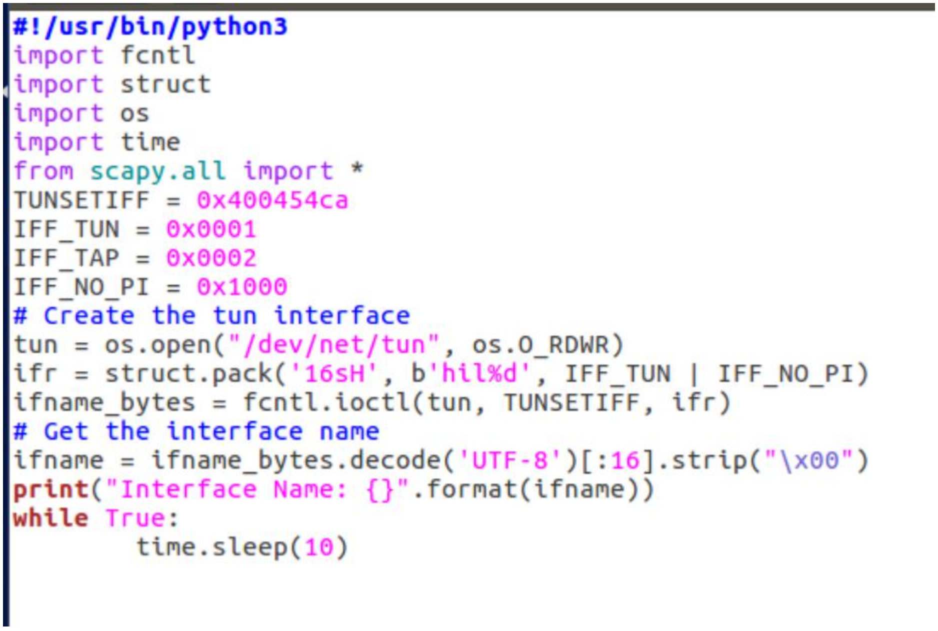


```
[09/28/20]seed@VM:~$ ping 10.0.2.6
PING 10.0.2.6 (10.0.2.6) 56(84) bytes of data.
64 bytes from 10.0.2.6: icmp_seq=1 ttl=64 time=0.478 ms
64 bytes from 10.0.2.6: icmp_seq=2 ttl=64 time=1.36 ms

64 bytes from 10.0.2.6: icmp_seq=3 ttl=64 time=1.39 ms
^C
--- 10.0.2.6 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2012ms
rtt min/avg/max/mdev = 0.478/1.077/1.391/0.424 ms
[09/28/20]seed@VM:~$ ping 192.168.70.101
PING 192.168.70.101 (192.168.70.101) 56(84) bytes of data.
^C
--- 192.168.70.101 ping statistics ---
11 packets transmitted, 0 received, 100% packet loss, time 10239ms
```

Task 2: Create and Configure TUN Interface

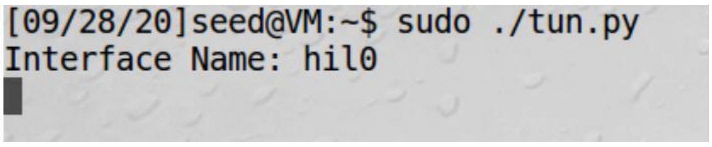
Task 2.a: Name of the Interface

创建文件 tun.py 写入代码，接口名为 hil

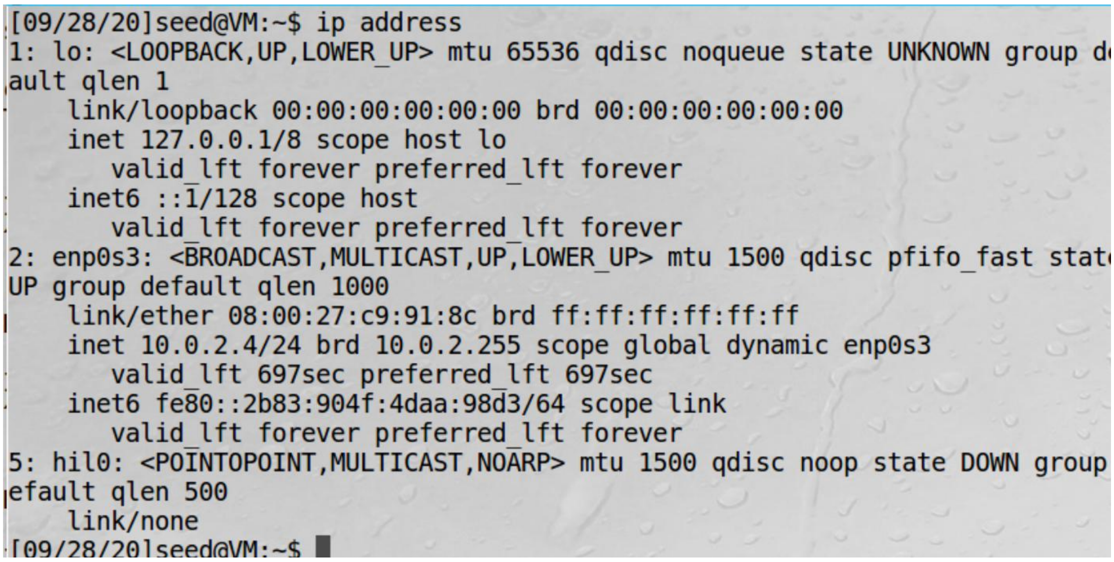


```
#!/usr/bin/python3
import fcntl
import struct
import os
import time
from scapy.all import *
TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'hil%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = 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)
```

运行文件，打印出接口为 hil0



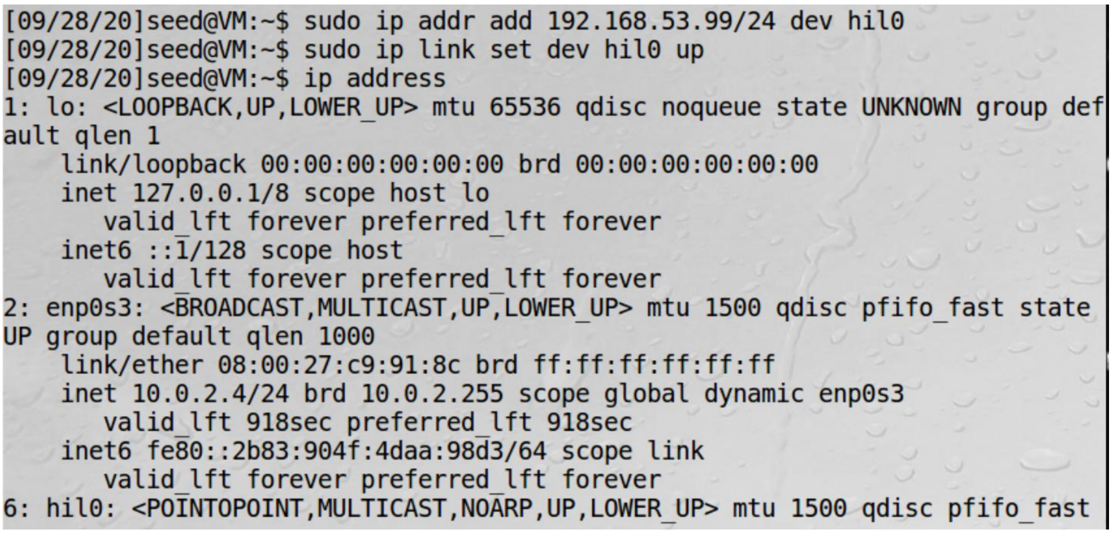
```
[09/28/20]seed@VM:~$ sudo ./tun.py
Interface Name: hil0
```



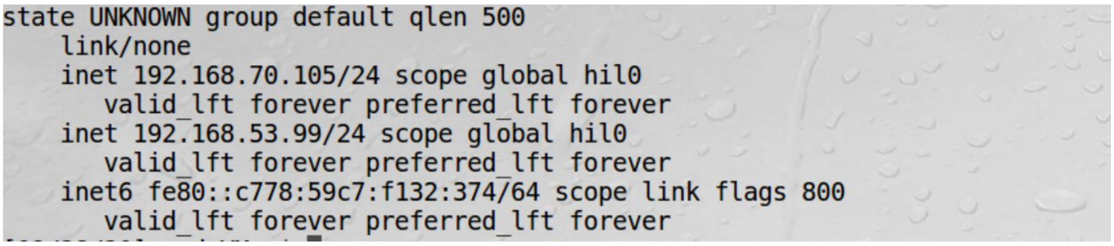
```
[09/28/20]seed@VM:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group d
ault qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast stat
UP group default qlen 1000
    link/ether 08:00:27:c9:91:8c brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.4/24 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 697sec preferred_lft 697sec
    inet6 fe80::2b83:904f:4daa:98d3/64 scope link
       valid_lft forever preferred_lft forever
5: hil0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group
efault qlen 500
    link/none
[09/28/20]seed@VM:~$
```

Task 2.b: Set up the TUN Interface

为接口 hil0 加上 IP 地址 192.168.53.99/24 并打开，使用 ip address 命令查看隧道状态



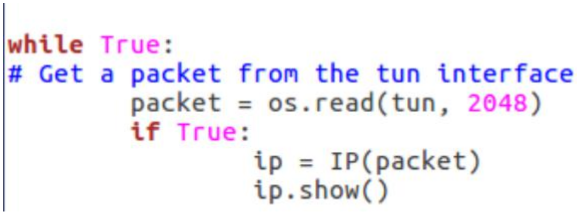
```
[09/28/20]seed@VM:~$ sudo ip addr add 192.168.53.99/24 dev hil0
[09/28/20]seed@VM:~$ sudo ip link set dev hil0 up
[09/28/20]seed@VM:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group def
ault qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    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: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state
UP group default qlen 1000
    link/ether 08:00:27:c9:91:8c brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.4/24 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 918sec preferred_lft 918sec
    inet6 fe80::2b83:904f:4daa:98d3/64 scope link
       valid_lft forever preferred_lft forever
6: hil0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
```



```
state UNKNOWN group default qlen 500
    link/none
    inet 192.168.70.105/24 scope global hil0
       valid_lft forever preferred_lft forever
    inet 192.168.53.99/24 scope global hil0
       valid_lft forever preferred_lft forever
    inet6 fe80::c778:59c7:f132:374/64 scope link flags 800
       valid_lft forever preferred_lft forever
```

Task 2.c: Read from the TUN Interface

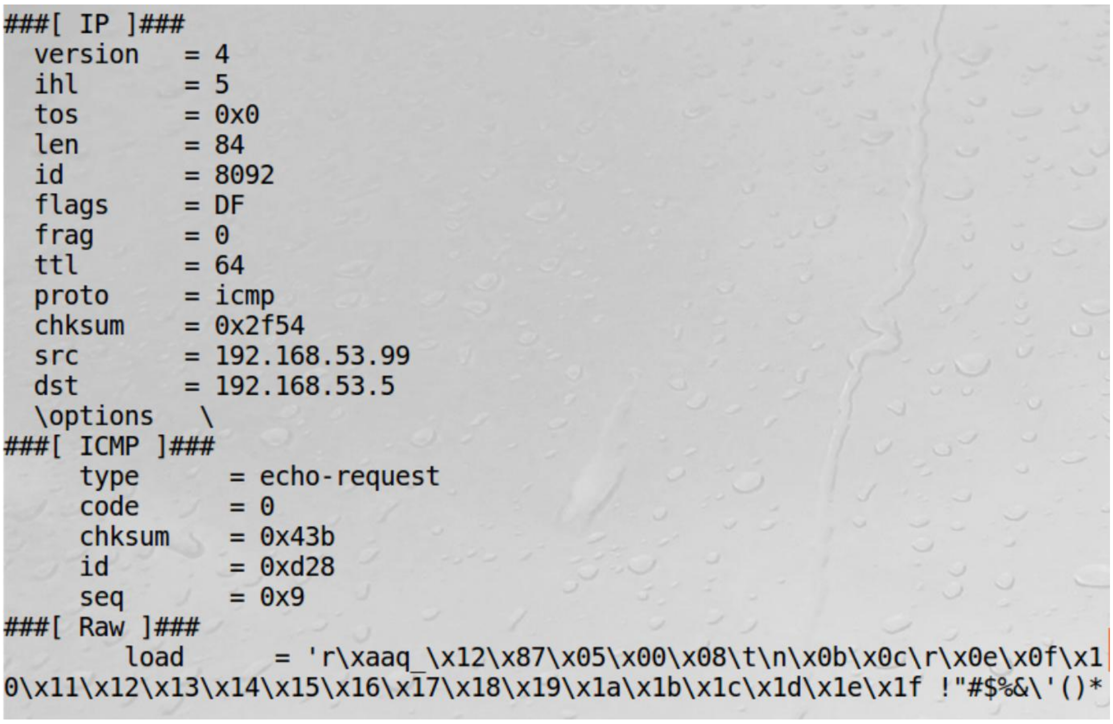
修改 tun.py



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

On Host U, ping a host in the 192.168.53.0/24 network.

重新运行文件，在另一个终端 ping 192.168.53.5，终端打印了以下内容
源地址为 192.168.53.99，目的地址为 192.168.53.5

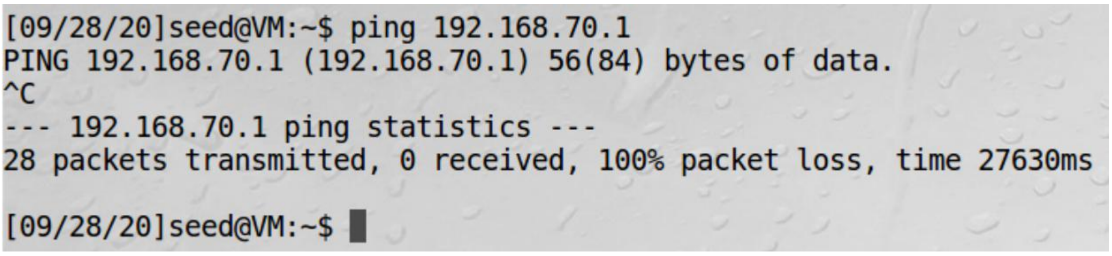


```
###[ IP ]###
  version   = 4
  ihl       = 5
  tos       = 0x0
  len       = 84
  id        = 8092
  flags     = DF
  frag      = 0
  ttl       = 64
  proto     = icmp
  chksum    = 0x2f54
  src       = 192.168.53.99
  dst       = 192.168.53.5
  \options   \
###[ ICMP ]###
     type      = echo-request
     code      = 0
     chksum    = 0x43b
     id        = 0xd28
     seq       = 0x9
###[ Raw ]###
        load      = 'r\xaaq_\x12\x87\x05\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x1
0\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*
```

因为主机 U 有两个不同网段的接口，主机 U 发送报文时，如果接口 hil0 和报文的网段相同，则从该接口发送出去，所以报文的源地址变为 hil0 接口内设置的 IP 地址。

On Host U, ping a host in the internal network 192.168.60.0/24（本机设置为 192.168.70.0/24）

ping 192.168.70.1，没有成功。另一个终端不显示输出。

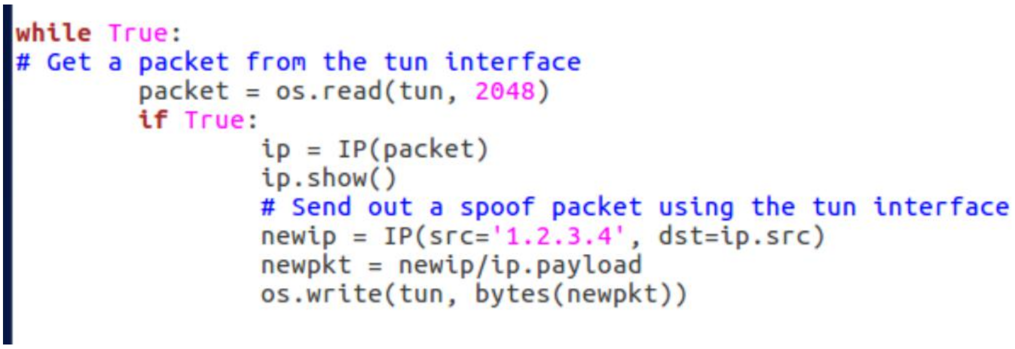


```
[09/28/20]seed@VM:~$ ping 192.168.70.1
PING 192.168.70.1 (192.168.70.1) 56(84) bytes of data.
^C
--- 192.168.70.1 ping statistics ---
28 packets transmitted, 0 received, 100% packet loss, time 27630ms

[09/28/20]seed@VM:~$ 
```

因为 hil0 接口与 VPN 服务器 192.168.70.1 不是一个网段，报文从其他接口发送出去，不能被程序接收到。

Task 2.d: Write to the TUN Interface

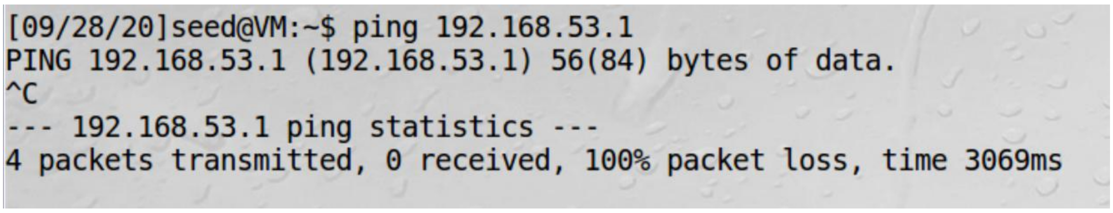
修改 tun.py



```
while True:
# Get a packet from the tun interface
        packet = os.read(tun, 2048)
        if True:
                ip = IP(packet)
                ip.show()
                # Send out a spoof packet using the tun interface
                newip = IP(src='1.2.3.4', dst=ip.src)
                newpkt = newip/ip.payload
                os.write(tun, bytes(newpkt))
```

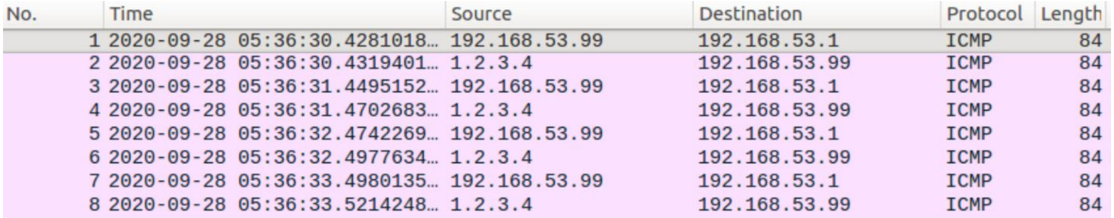
After getting a packet from the TUN interface, send out a new packet to the TUN interface.

修改文件后重新运行，ping 192.168.53.1



```
[09/28/20]seed@VM:~$ 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 ---
4 packets transmitted, 0 received, 100% packet loss, time 3069ms
```

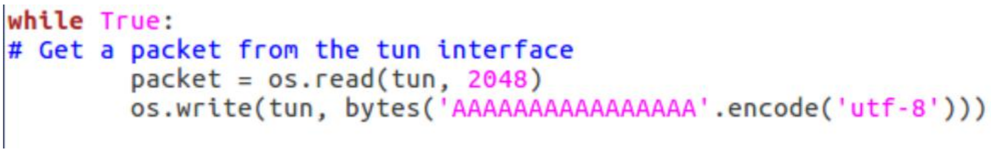
使用 wireshark 查看 hil0 接口，在接收到发往 192.168.53.1 的报文后都会发送一个源地址为 1.2.3.4 的报文



No.	Time	Source	Destination	Protocol	Length
1	2020-09-28 05:36:30.4281018…	192.168.53.99	192.168.53.1	ICMP	84
2	2020-09-28 05:36:30.4319401…	1.2.3.4	192.168.53.99	ICMP	84
3	2020-09-28 05:36:31.4495152…	192.168.53.99	192.168.53.1	ICMP	84
4	2020-09-28 05:36:31.4702683…	1.2.3.4	192.168.53.99	ICMP	84
5	2020-09-28 05:36:32.4742269…	192.168.53.99	192.168.53.1	ICMP	84
6	2020-09-28 05:36:32.4977634…	1.2.3.4	192.168.53.99	ICMP	84
7	2020-09-28 05:36:33.4980135…	192.168.53.99	192.168.53.1	ICMP	84
8	2020-09-28 05:36:33.5214248…	1.2.3.4	192.168.53.99	ICMP	84

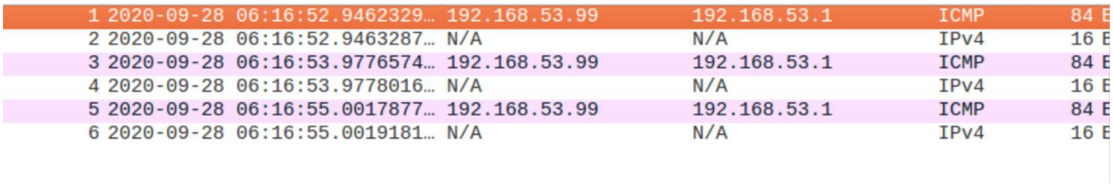
Instead of writing an IP packet to the interface, write some arbitrary data to the interface, and report your observation.

修改 tun.py，重新运行



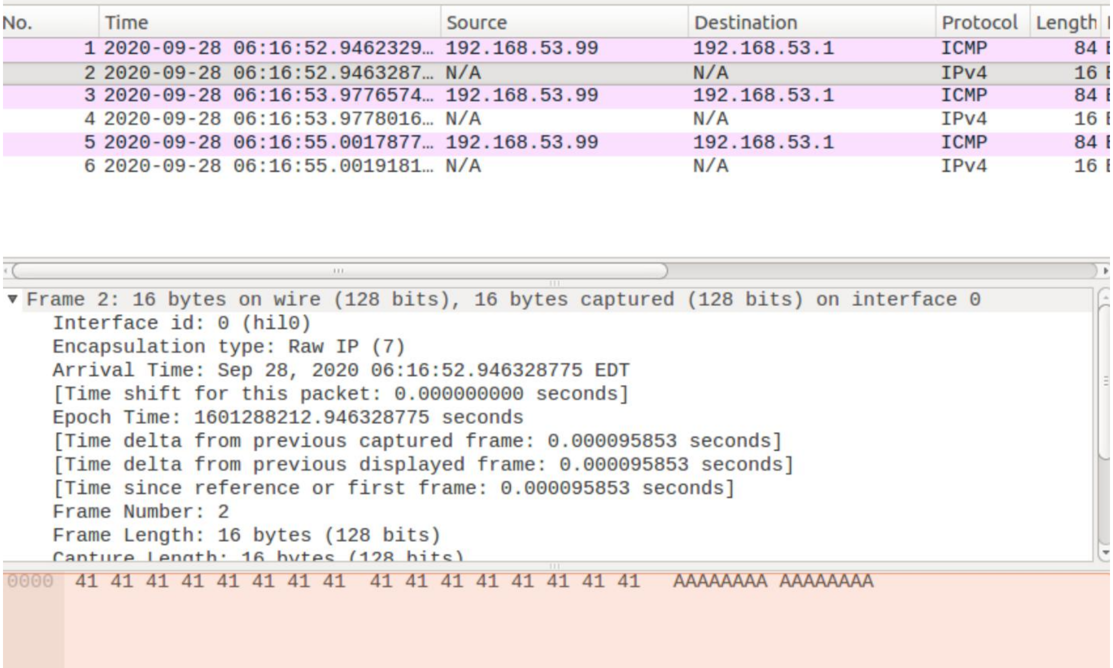
```
while True:
# Get a packet from the tun interface
        packet = os.read(tun, 2048)
        os.write(tun, bytes('AAAAAAAAAAAAAAAA'.encode('utf-8')))
```

发送 ping 报文后，用 wireshark 抓包获得许多源地址和目的地址为 N/A 的报文



1	2020-09-28 06:16:52.9462329…	192.168.53.99	192.168.53.1	ICMP	84 E
2	2020-09-28 06:16:52.9463287…	N/A	N/A	IPv4	16 E
3	2020-09-28 06:16:53.9776574…	192.168.53.99	192.168.53.1	ICMP	84 E
4	2020-09-28 06:16:53.9778016…	N/A	N/A	IPv4	16 E
5	2020-09-28 06:16:55.0017877…	192.168.53.99	192.168.53.1	ICMP	84 E
6	2020-09-28 06:16:55.0019181…	N/A	N/A	IPv4	16 E

报文的内容为 AAAA



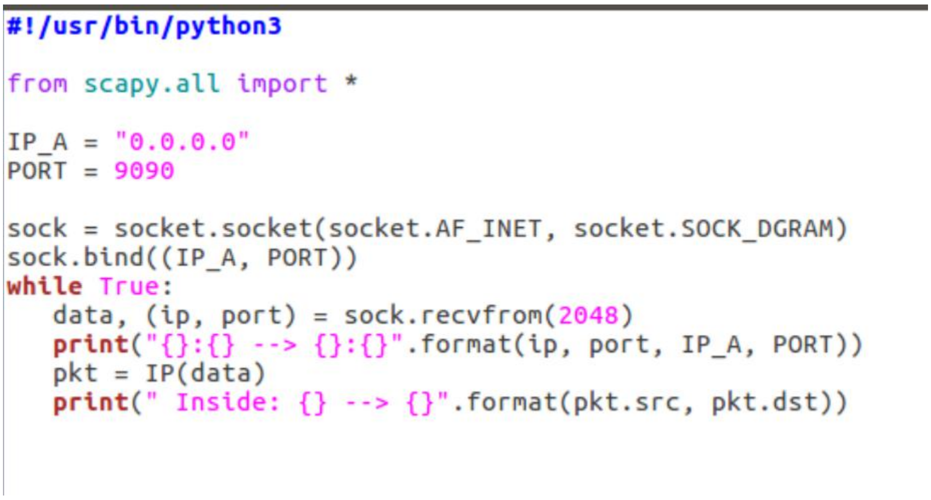
No.	Time	Source	Destination	Protocol	Length
1	2020-09-28 06:16:52.9462329…	192.168.53.99	192.168.53.1	ICMP	84
2	2020-09-28 06:16:52.9463287…	N/A	N/A	IPv4	16
3	2020-09-28 06:16:53.9776574…	192.168.53.99	192.168.53.1	ICMP	84
4	2020-09-28 06:16:53.9778016…	N/A	N/A	IPv4	16
5	2020-09-28 06:16:55.0017877…	192.168.53.99	192.168.53.1	ICMP	84
6	2020-09-28 06:16:55.0019181…	N/A	N/A	IPv4	16

```
▾ Frame 2: 16 bytes on wire (128 bits), 16 bytes captured (128 bits) on interface 0
    Interface id: 0 (hil0)
    Encapsulation type: Raw IP (7)
    Arrival Time: Sep 28, 2020 06:16:52.946328775 EDT
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1601288212.946328775 seconds
    [Time delta from previous captured frame: 0.000095853 seconds]
    [Time delta from previous displayed frame: 0.000095853 seconds]
    [Time since reference or first frame: 0.000095853 seconds]
    Frame Number: 2
    Frame Length: 16 bytes (128 bits)
    Capture Length: 16 bytes (128 bits)
```

```
0000  41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41   AAAAAAAA AAAAAAAA
```

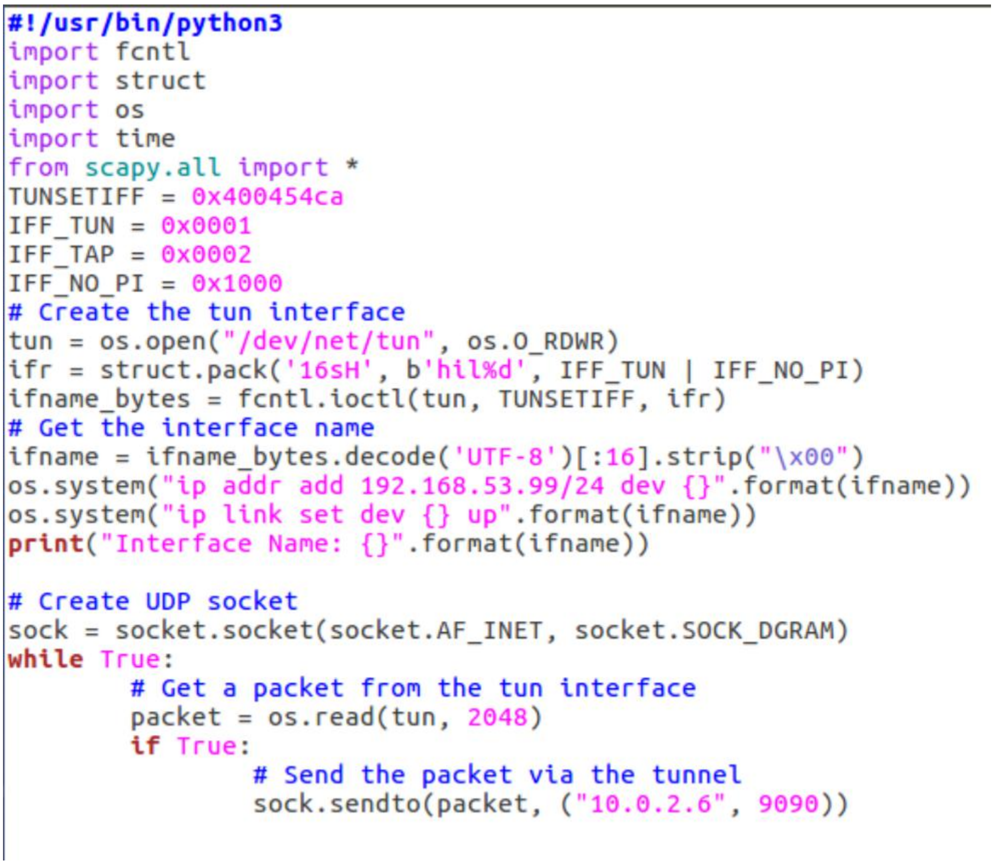
Task 3: Send the IP Packet to VPN Server Through a Tunnel

在 VPN Server 上编写程序 server.py



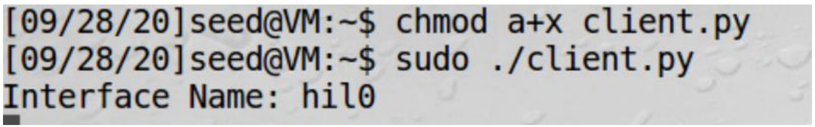
```
#!/usr/bin/python3

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))
```

在 VPN Client 主机 U 上编写程序 client.py

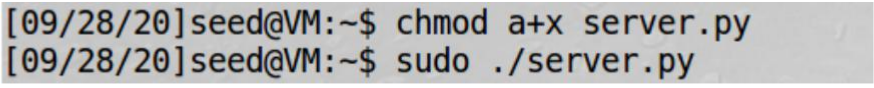


```
#!/usr/bin/python3
import fcntl
import struct
import os
import time
from scapy.all import *
TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'hil%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
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)
while True:
        # Get a packet from the tun interface
        packet = os.read(tun, 2048)
        if True:
                # Send the packet via the tunnel
                sock.sendto(packet, ("10.0.2.6", 9090))
```

在服务器和主机 U（客户端）分别运行 py 文件



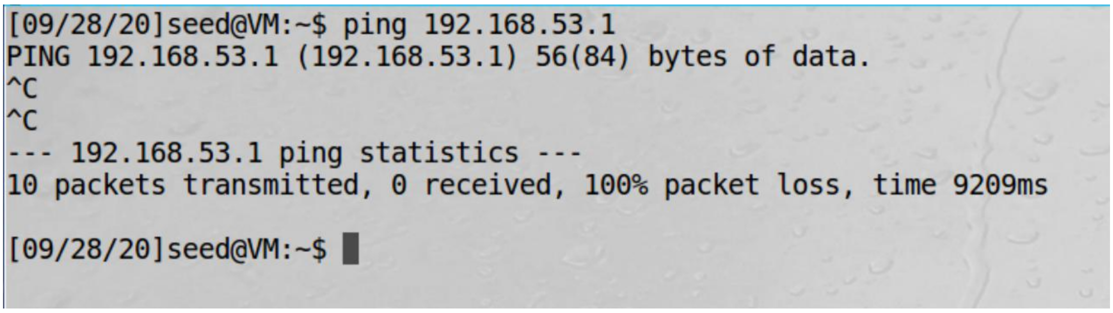
```
[09/28/20]seed@VM:~$ chmod a+x client.py
[09/28/20]seed@VM:~$ sudo ./client.py
Interface Name: hil0
```



```
[09/28/20]seed@VM:~$ chmod a+x server.py
[09/28/20]seed@VM:~$ sudo ./server.py
```

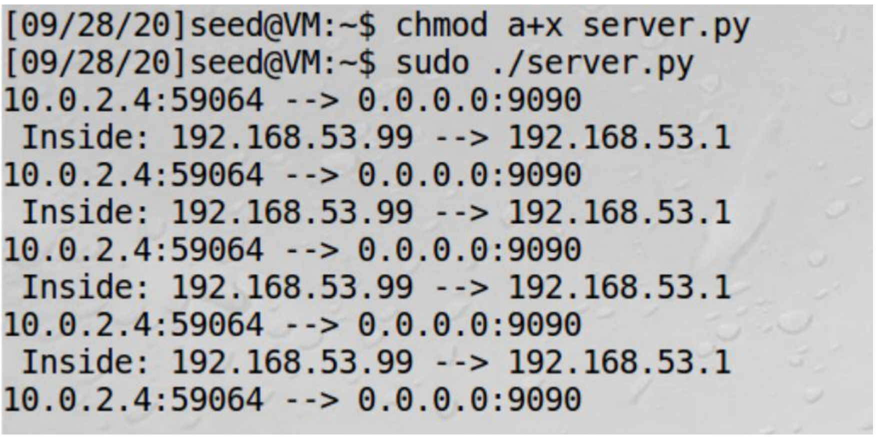
To test whether the tunnel works or not, ping any IP address belonging to the 192.168.53.0/24
network. What is printed out on VPN Server? Why?

在客户端 ping 192.168.53.0/24 网络内的地址



```
[09/28/20]seed@VM:~$ ping 192.168.53.1
PING 192.168.53.1 (192.168.53.1) 56(84) bytes of data.
^C
^C
--- 192.168.53.1 ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time 9209ms

[09/28/20]seed@VM:~$ 
```

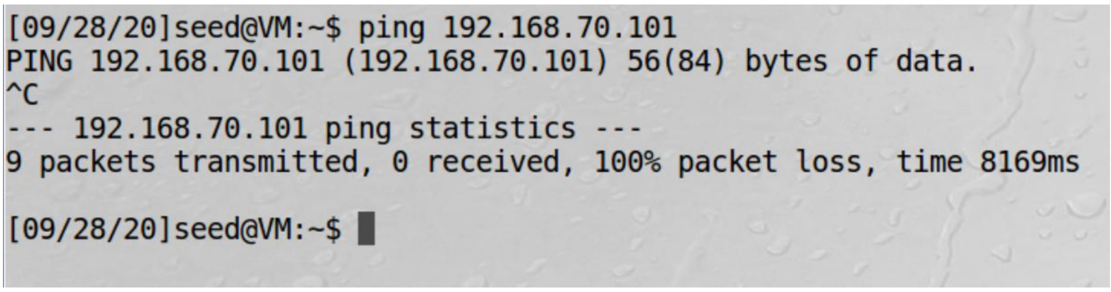
此时服务器上打印出了客户端发送的报文信息，因为服务器监听了报文传输过程，可以打印出客户端发送的报文



```
[09/28/20]seed@VM:~$ chmod a+x server.py
[09/28/20]seed@VM:~$ sudo ./server.py
10.0.2.4:59064 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
10.0.2.4:59064 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
10.0.2.4:59064 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
10.0.2.4:59064 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
10.0.2.4:59064 --> 0.0.0.0:9090
```

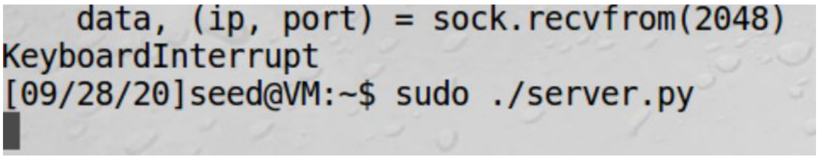
Let us ping Host V, and see whether the ICMP packet is sent to VPN Server through the tunnel.

在主机 U 上 ping 主机 V



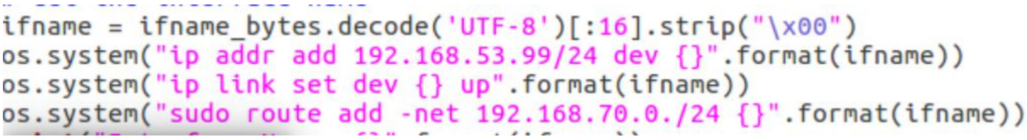
```
[09/28/20]seed@VM:~$ ping 192.168.70.101
PING 192.168.70.101 (192.168.70.101) 56(84) bytes of data.
^C
--- 192.168.70.101 ping statistics ---
9 packets transmitted, 0 received, 100% packet loss, time 8169ms

[09/28/20]seed@VM:~$ 
```

服务器端没有输出。说明 ping 主机 V 的报文没有通过我们设置的隧道和接口，所以程序没有打印这些隧道以外的报文。



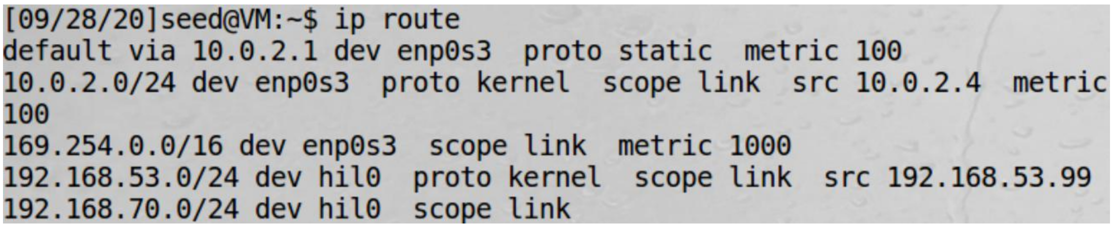
```
    data, (ip, port) = sock.recvfrom(2048)
KeyboardInterrupt
[09/28/20]seed@VM:~$ sudo ./server.py
```

为了解决这一问题，我们需要在 client.py 中加一条静态路由规则，将发往网络 192.168.70.0/24 的报文从添加的接口 hil0 发送出去，经过隧道传到服务器端。



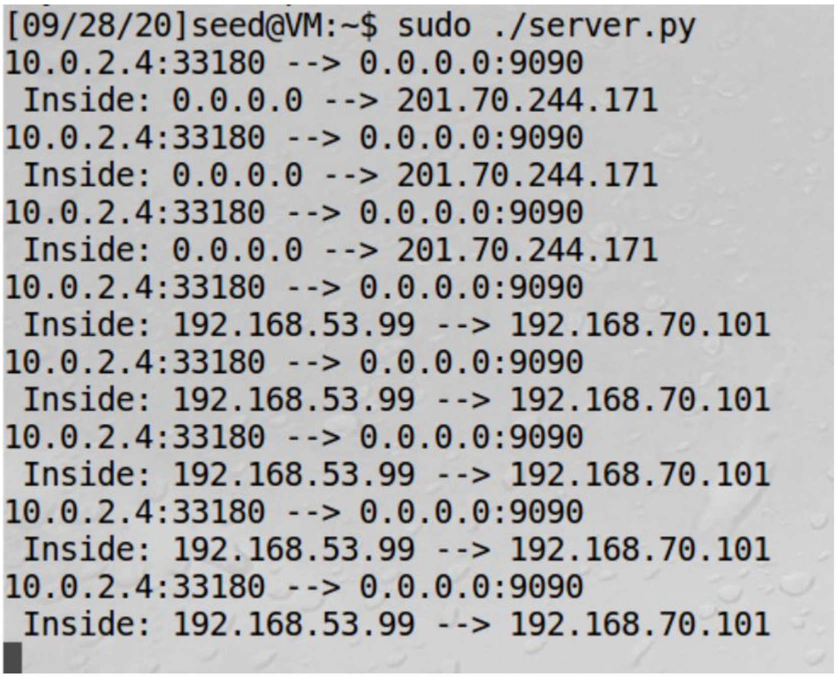
```
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
os.system("sudo route add -net 192.168.70.0./24 {}".format(ifname))
```

在主机 U 上重新运行 client.py，查看路由表，路由已添加



```
[09/28/20]seed@VM:~$ ip route
default via 10.0.2.1 dev enp0s3  proto static  metric 100
10.0.2.0/24 dev enp0s3  proto kernel  scope link  src 10.0.2.4  metric
100
169.254.0.0/16 dev enp0s3  scope link  metric 1000
192.168.53.0/24 dev hil0  proto kernel  scope link  src 192.168.53.99
192.168.70.0/24 dev hil0  scope link
```

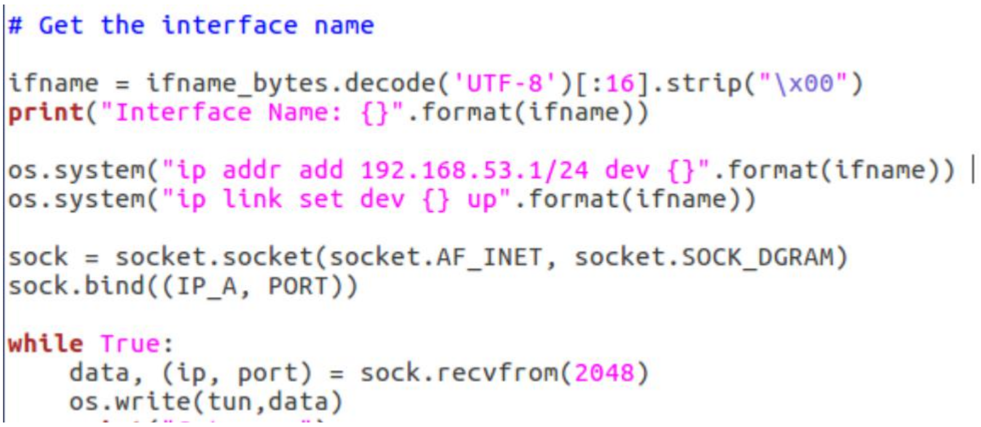
再在主机 U 上 ping 主机 V，在服务器端打印出了发送到 192.168.70.101 的报文信息



```
[09/28/20]seed@VM:~$ sudo ./server.py
10.0.2.4:33180 --> 0.0.0.0:9090
 Inside: 0.0.0.0 --> 201.70.244.171
10.0.2.4:33180 --> 0.0.0.0:9090
 Inside: 0.0.0.0 --> 201.70.244.171
10.0.2.4:33180 --> 0.0.0.0:9090
 Inside: 0.0.0.0 --> 201.70.244.171
10.0.2.4:33180 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.70.101
10.0.2.4:33180 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.70.101
10.0.2.4:33180 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.70.101
10.0.2.4:33180 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.70.101
10.0.2.4:33180 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.70.101
```

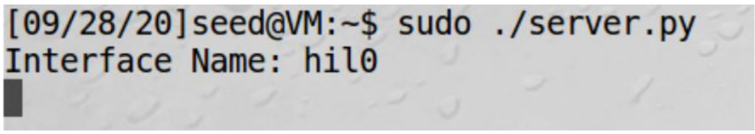
Task 4: Set Up the VPN Server

修改 server.py

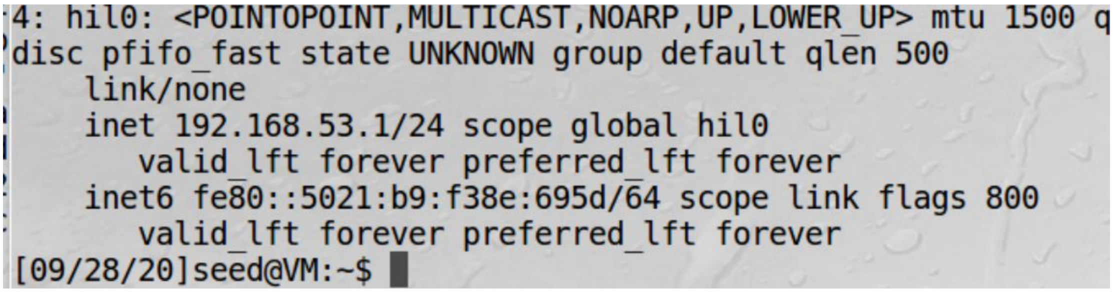


```
# Get the interface name

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

os.system("ip addr add 192.168.53.1/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))

while True:
    data, (ip, port) = sock.recvfrom(2048)
    os.write(tun,data)
```

在服务器端运行 server.py，新接口 hil0 已设置添加

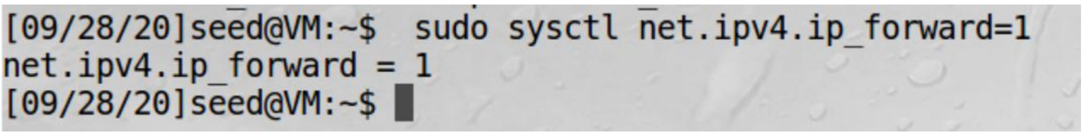


```
[09/28/20]seed@VM:~$ sudo ./server.py
Interface Name: hil0
```



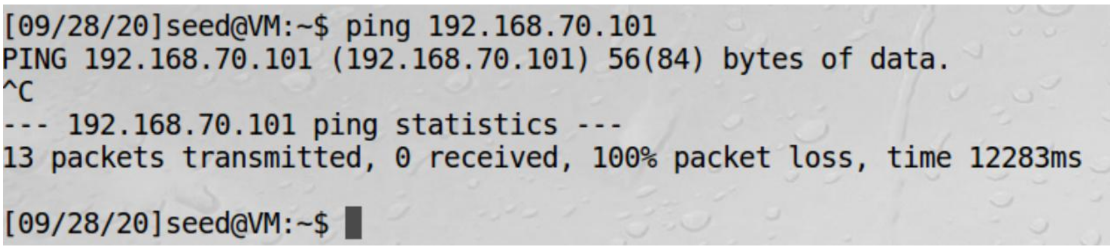
```
4: hil0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 q
disc pfifo_fast state UNKNOWN group default qlen 500
    link/none
    inet 192.168.53.1/24 scope global hil0
       valid_lft forever preferred_lft forever
    inet6 fe80::5021:b9:f38e:695d/64 scope link flags 800
       valid_lft forever preferred_lft forever
[09/28/20]seed@VM:~$
```

打开 IP 转发功能



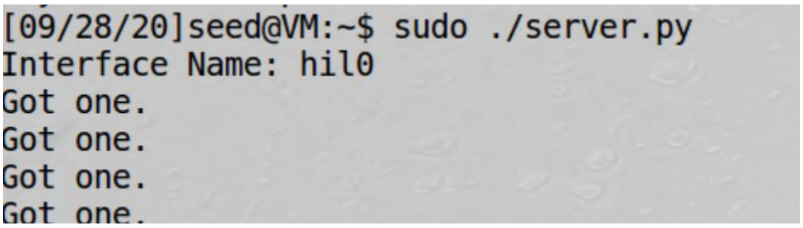
```
[09/28/20]seed@VM:~$  sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
[09/28/20]seed@VM:~$
```

在主机 U 上 ping 主机 V



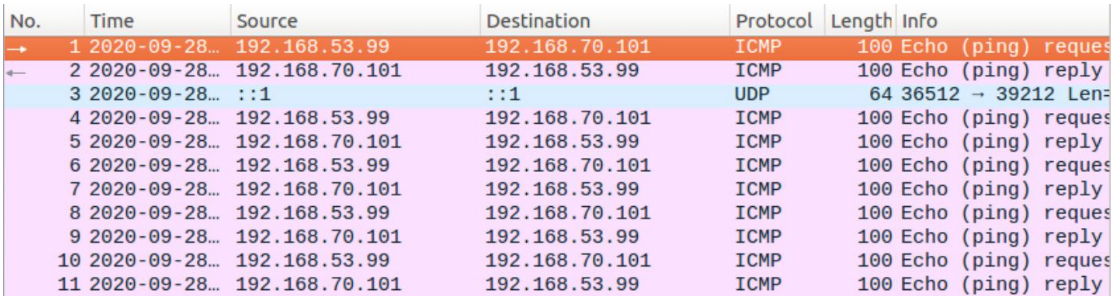
```
[09/28/20]seed@VM:~$ ping 192.168.70.101
PING 192.168.70.101 (192.168.70.101) 56(84) bytes of data.
^C
--- 192.168.70.101 ping statistics ---
13 packets transmitted, 0 received, 100% packet loss, time 12283ms

[09/28/20]seed@VM:~$
```

服务器端显示接收到报文



```
[09/28/20]seed@VM:~$ sudo ./server.py
Interface Name: hil0
Got one.
Got one.
Got one.
Got one.
```

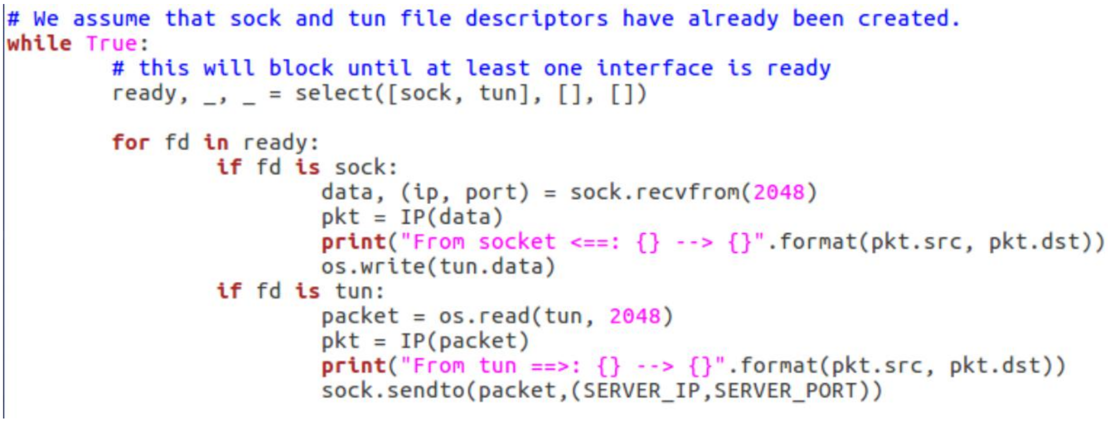
主机 V 上使用 wireshark 抓包到了来自 192.168.53.99 的 ICMP 报文



No.	Time	Source	Destination	Protocol	Length	Info
1	2020-09-28…	192.168.53.99	192.168.70.101	ICMP	100	Echo (ping) reques
2	2020-09-28…	192.168.70.101	192.168.53.99	ICMP	100	Echo (ping) reply
3	2020-09-28…	::1	::1	UDP	64	36512 → 39212 Len=
4	2020-09-28…	192.168.53.99	192.168.70.101	ICMP	100	Echo (ping) reques
5	2020-09-28…	192.168.70.101	192.168.53.99	ICMP	100	Echo (ping) reply
6	2020-09-28…	192.168.53.99	192.168.70.101	ICMP	100	Echo (ping) reques
7	2020-09-28…	192.168.70.101	192.168.53.99	ICMP	100	Echo (ping) reply
8	2020-09-28…	192.168.53.99	192.168.70.101	ICMP	100	Echo (ping) reques
9	2020-09-28…	192.168.70.101	192.168.53.99	ICMP	100	Echo (ping) reply
10	2020-09-28…	192.168.53.99	192.168.70.101	ICMP	100	Echo (ping) reques
11	2020-09-28…	192.168.70.101	192.168.53.99	ICMP	100	Echo (ping) reply

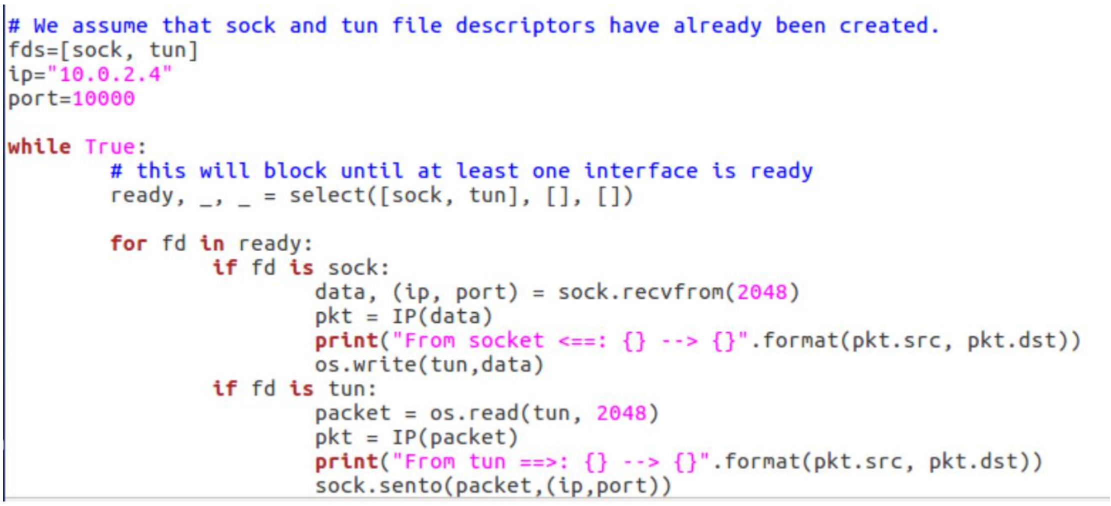
Task 5: Handling Traffic in Both Directions

修改 client.py 文件



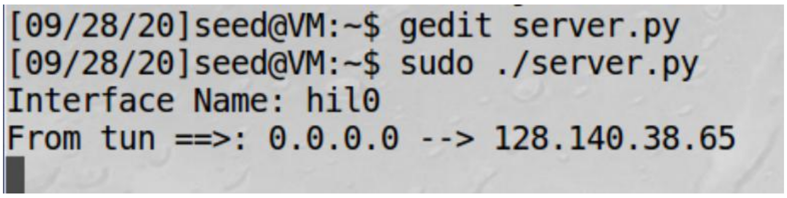
```
# We assume that sock and tun file descriptors have already been created.
while True:
        # this will block until at least one interface is ready
        ready, _, _ = select([sock, tun], [], [])

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

修改 server.py 文件

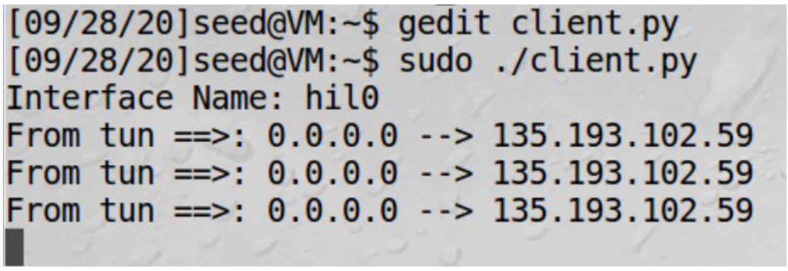


```
# We assume that sock and tun file descriptors have already been created.
fds=[sock, tun]
ip="10.0.2.4"
port=10000

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

分别运行 client.py 和 server.py 文件

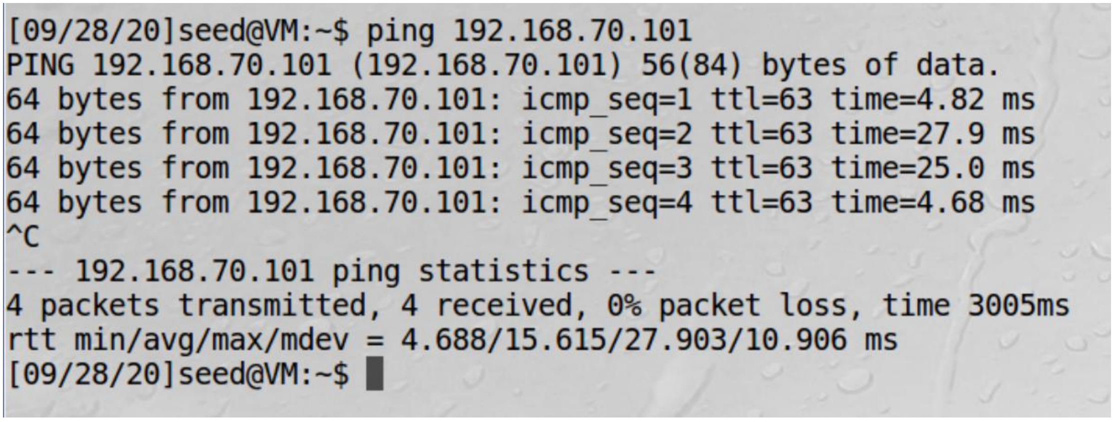


```
[09/28/20]seed@VM:~$ gedit server.py
[09/28/20]seed@VM:~$ sudo ./server.py
Interface Name: hil0
From tun ==>: 0.0.0.0 --> 128.140.38.65
```



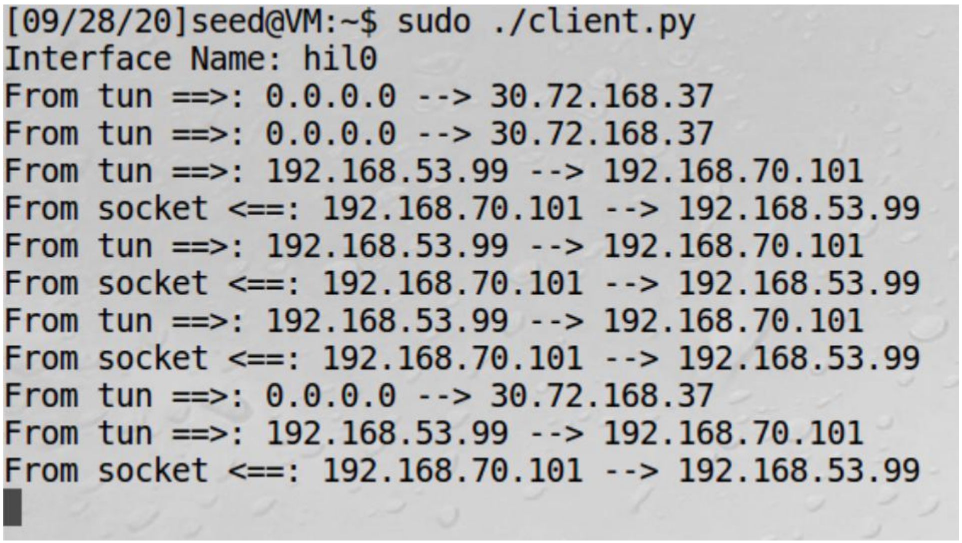
```
[09/28/20]seed@VM:~$ gedit client.py
[09/28/20]seed@VM:~$ sudo ./client.py
Interface Name: hil0
From tun ==>: 0.0.0.0 --> 135.193.102.59
From tun ==>: 0.0.0.0 --> 135.193.102.59
From tun ==>: 0.0.0.0 --> 135.193.102.59
```

此时主机 U 可以 ping 通主机 V

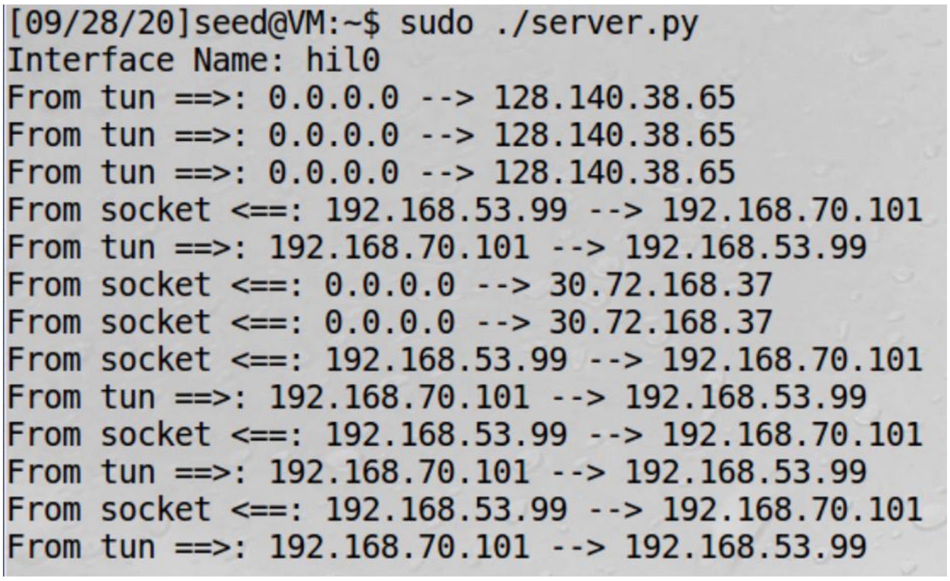


```
[09/28/20]seed@VM:~$ ping 192.168.70.101
PING 192.168.70.101 (192.168.70.101) 56(84) bytes of data.
64 bytes from 192.168.70.101: icmp_seq=1 ttl=63 time=4.82 ms
64 bytes from 192.168.70.101: icmp_seq=2 ttl=63 time=27.9 ms
64 bytes from 192.168.70.101: icmp_seq=3 ttl=63 time=25.0 ms
64 bytes from 192.168.70.101: icmp_seq=4 ttl=63 time=4.68 ms
^C
--- 192.168.70.101 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 4.688/15.615/27.903/10.906 ms
[09/28/20]seed@VM:~$
```

客户端和服务器端运行文件后都接收到传输的报文



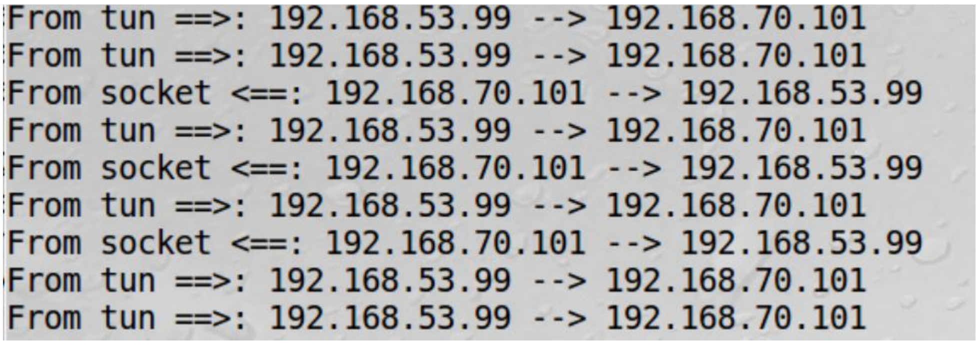
```
[09/28/20]seed@VM:~$ sudo ./client.py
Interface Name: hil0
From tun ==>: 0.0.0.0 --> 30.72.168.37
From tun ==>: 0.0.0.0 --> 30.72.168.37
From tun ==>: 192.168.53.99 --> 192.168.70.101
From socket <==: 192.168.70.101 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.70.101
From socket <==: 192.168.70.101 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.70.101
From socket <==: 192.168.70.101 --> 192.168.53.99
From tun ==>: 0.0.0.0 --> 30.72.168.37
From tun ==>: 192.168.53.99 --> 192.168.70.101
From socket <==: 192.168.70.101 --> 192.168.53.99
```



```
[09/28/20]seed@VM:~$ sudo ./server.py
Interface Name: hil0
From tun ==>: 0.0.0.0 --> 128.140.38.65
From tun ==>: 0.0.0.0 --> 128.140.38.65
From tun ==>: 0.0.0.0 --> 128.140.38.65
From socket <==: 192.168.53.99 --> 192.168.70.101
From tun ==>: 192.168.70.101 --> 192.168.53.99
From socket <==: 0.0.0.0 --> 30.72.168.37
From socket <==: 0.0.0.0 --> 30.72.168.37
From socket <==: 192.168.53.99 --> 192.168.70.101
From tun ==>: 192.168.70.101 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.70.101
From tun ==>: 192.168.70.101 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.70.101
From tun ==>: 192.168.70.101 --> 192.168.53.99
```

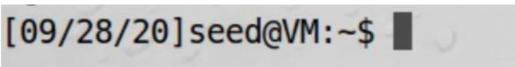
Task 6: Tunnel-Breaking Experiment

在主机 U 上向主机 V 建立 telent 连接，客户端和服务端都显示了报文



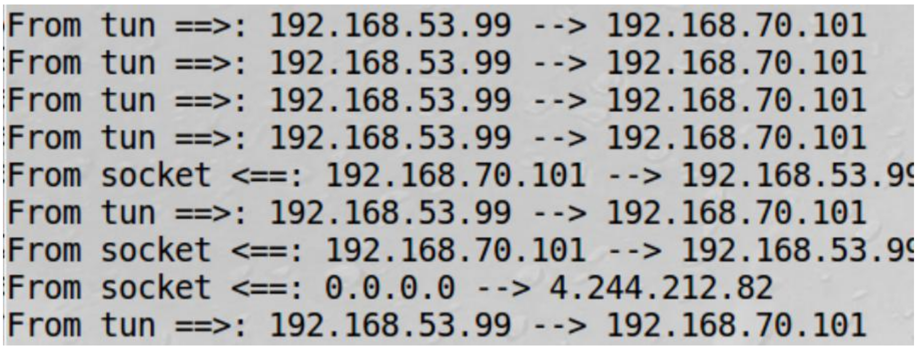
```
From tun ==>: 192.168.53.99 --> 192.168.70.101
From tun ==>: 192.168.53.99 --> 192.168.70.101
From socket <==: 192.168.70.101 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.70.101
From socket <==: 192.168.70.101 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.70.101
From socket <==: 192.168.70.101 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.70.101
From tun ==>: 192.168.53.99 --> 192.168.70.101
```

关闭 server.py 停止运行，此时主机 U 无法输入命令



```
[09/28/20]seed@VM:~$ 
```

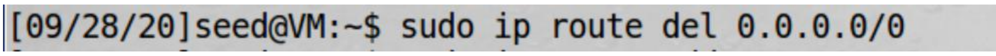
运行 server.py，主机 U 中命令出现，连接恢复



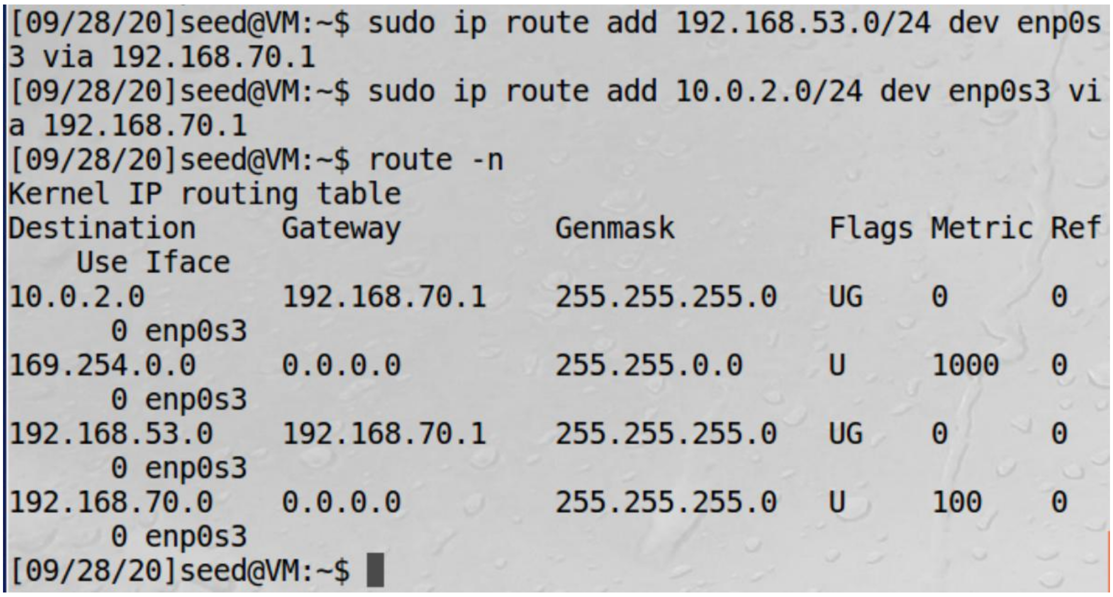
```
From tun ==>: 192.168.53.99 --> 192.168.70.101
From tun ==>: 192.168.53.99 --> 192.168.70.101
From tun ==>: 192.168.53.99 --> 192.168.70.101
From tun ==>: 192.168.53.99 --> 192.168.70.101
From socket <==: 192.168.70.101 --> 192.168.53.9
From tun ==>: 192.168.53.99 --> 192.168.70.101
From socket <==: 192.168.70.101 --> 192.168.53.9
From socket <==: 0.0.0.0 --> 4.244.212.82
From tun ==>: 192.168.53.99 --> 192.168.70.101
```

Task 7: Routing Experiment on Host V

在主机 V 上配置路由，使用 route -n 查看配置情况

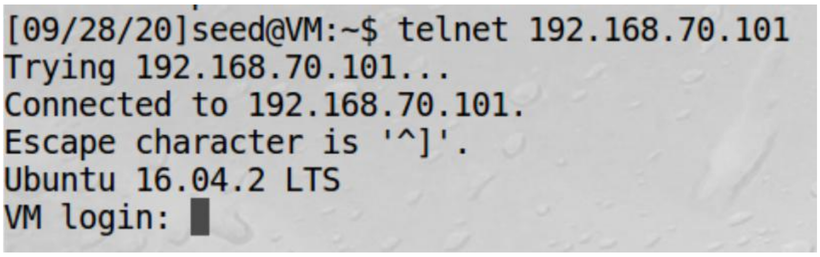


```
[09/28/20]seed@VM:~$ sudo ip route del 0.0.0.0/0
```



```
[09/28/20]seed@VM:~$ sudo ip route add 192.168.53.0/24 dev enp0s
3 via 192.168.70.1
[09/28/20]seed@VM:~$ sudo ip route add 10.0.2.0/24 dev enp0s3 vi
a 192.168.70.1
[09/28/20]seed@VM:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref
    Use Iface
10.0.2.0        192.168.70.1    255.255.255.0   UG    0      0
     0 enp0s3
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0
     0 enp0s3
192.168.53.0    192.168.70.1    255.255.255.0   UG    0      0
     0 enp0s3
192.168.70.0    0.0.0.0         255.255.255.0   U     100    0
     0 enp0s3
[09/28/20]seed@VM:~$ 
```

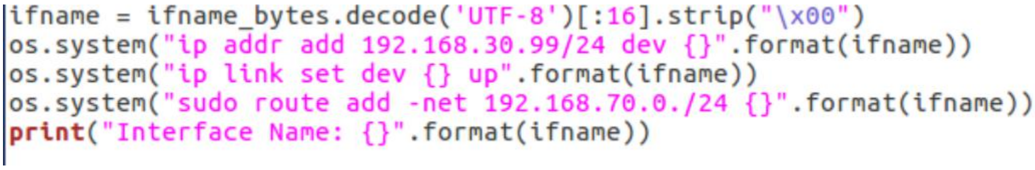
在主机 U 上向主机 V 建立 telnet 连接，连接成功



```
[09/28/20]seed@VM:~$ telnet 192.168.70.101
Trying 192.168.70.101...
Connected to 192.168.70.101.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: 
```

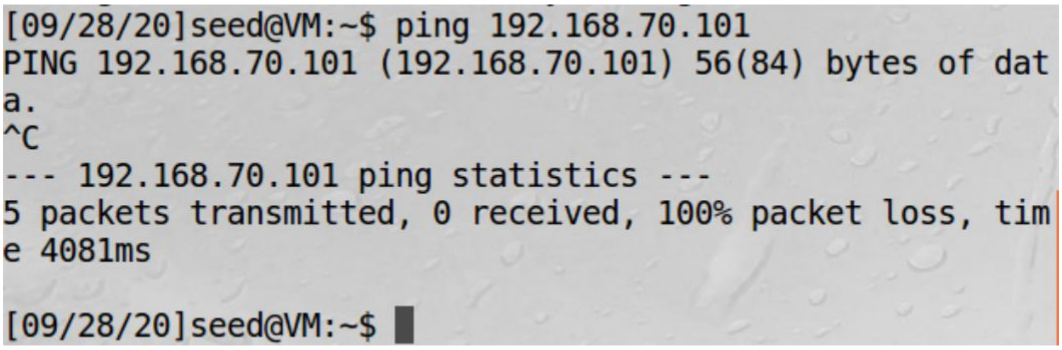
Task 8: Experiment with the TUN IP Address

将客户端的接口 IP 地址改为 192.168.30.99



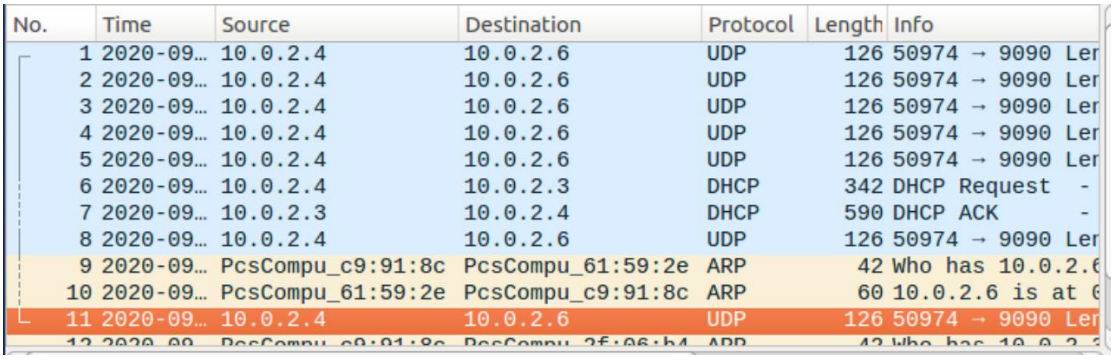
```
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
os.system("ip addr add 192.168.30.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
os.system("sudo route add -net 192.168.70.0./24 {}".format(ifname))
print("Interface Name: {}".format(ifname))
```

主机 U 无法 ping 通主机 V

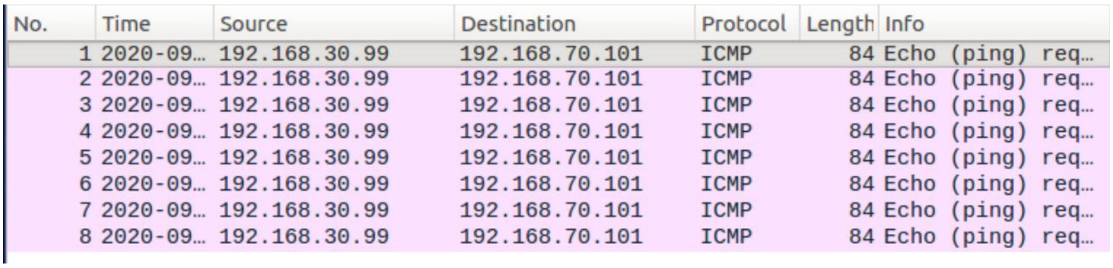


```
[09/28/20]seed@VM:~$ ping 192.168.70.101
PING 192.168.70.101 (192.168.70.101) 56(84) bytes of dat
a.
^C
--- 192.168.70.101 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, tim
e 4081ms

[09/28/20]seed@VM:~$ 
```

在主机 U 上打开 wireshark，enp0s3 接口内显示了从 10.0.2.4 发往 10.0.2.6 的 UDP 报文，hil0 接口内有 192.168.30.99 发往主机 V 的报文

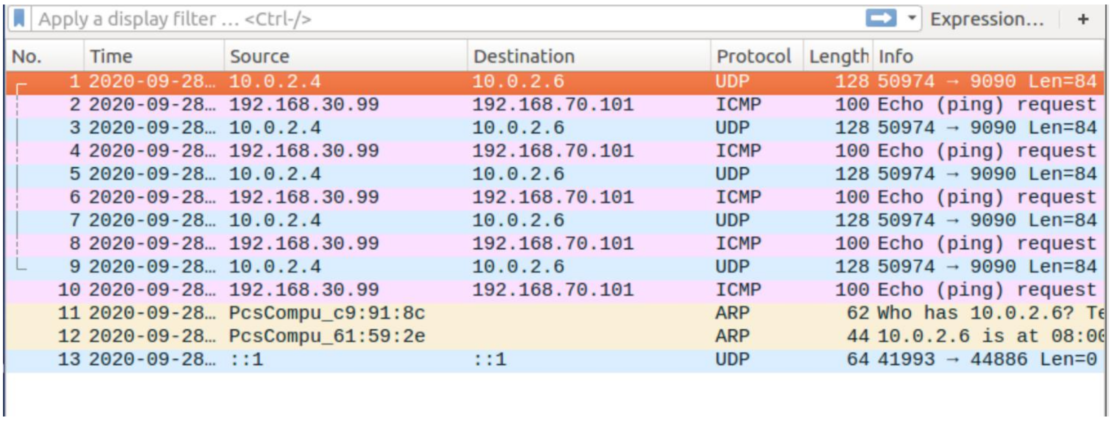


No.	Time	Source	Destination	Protocol	Length	Info
1	2020-09…	10.0.2.4	10.0.2.6	UDP	126	50974 → 9090 Ler
2	2020-09…	10.0.2.4	10.0.2.6	UDP	126	50974 → 9090 Ler
3	2020-09…	10.0.2.4	10.0.2.6	UDP	126	50974 → 9090 Ler
4	2020-09…	10.0.2.4	10.0.2.6	UDP	126	50974 → 9090 Ler
5	2020-09…	10.0.2.4	10.0.2.6	UDP	126	50974 → 9090 Ler
6	2020-09…	10.0.2.4	10.0.2.3	DHCP	342	DHCP Request -
7	2020-09…	10.0.2.3	10.0.2.4	DHCP	590	DHCP ACK -
8	2020-09…	10.0.2.4	10.0.2.6	UDP	126	50974 → 9090 Ler
9	2020-09…	PcsCompu_c9:91:8c	PcsCompu_61:59:2e	ARP	42	Who has 10.0.2.6
10	2020-09…	PcsCompu_61:59:2e	PcsCompu_c9:91:8c	ARP	60	10.0.2.6 is at 0
11	2020-09…	10.0.2.4	10.0.2.6	UDP	126	50974 → 9090 Ler
12	2020-09…	PcsCompu_c9:91:8c	PcsCompu_2f:06:b4	ARP	42	Who has 10.0.2.3



No.	Time	Source	Destination	Protocol	Length	Info
1	2020-09…	192.168.30.99	192.168.70.101	ICMP	84	Echo (ping) req…
2	2020-09…	192.168.30.99	192.168.70.101	ICMP	84	Echo (ping) req…
3	2020-09…	192.168.30.99	192.168.70.101	ICMP	84	Echo (ping) req…
4	2020-09…	192.168.30.99	192.168.70.101	ICMP	84	Echo (ping) req…
5	2020-09…	192.168.30.99	192.168.70.101	ICMP	84	Echo (ping) req…
6	2020-09…	192.168.30.99	192.168.70.101	ICMP	84	Echo (ping) req…
7	2020-09…	192.168.30.99	192.168.70.101	ICMP	84	Echo (ping) req…
8	2020-09…	192.168.30.99	192.168.70.101	ICMP	84	Echo (ping) req…

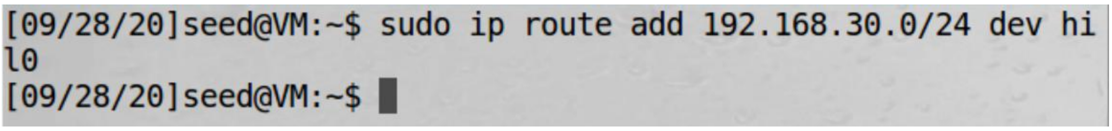
服务器端 wireshark 可以看到两个接口发送的报文



```
No.  Time             Source              Destination      Protocol Length Info
   1 2020-09-28…  10.0.2.4            10.0.2.6         UDP       128 50974 → 9090 Len=84
   2 2020-09-28…  192.168.30.99       192.168.70.101   ICMP      100 Echo (ping) request
   3 2020-09-28…  10.0.2.4            10.0.2.6         UDP       128 50974 → 9090 Len=84
   4 2020-09-28…  192.168.30.99       192.168.70.101   ICMP      100 Echo (ping) request
   5 2020-09-28…  10.0.2.4            10.0.2.6         UDP       128 50974 → 9090 Len=84
   6 2020-09-28…  192.168.30.99       192.168.70.101   ICMP      100 Echo (ping) request
   7 2020-09-28…  10.0.2.4            10.0.2.6         UDP       128 50974 → 9090 Len=84
   8 2020-09-28…  192.168.30.99       192.168.70.101   ICMP      100 Echo (ping) request
   9 2020-09-28…  10.0.2.4            10.0.2.6         UDP       128 50974 → 9090 Len=84
  10 2020-09-28…  192.168.30.99       192.168.70.101   ICMP      100 Echo (ping) request
  11 2020-09-28…  PcsCompu_c9:91:8c                    ARP        62 Who has 10.0.2.6? Te
  12 2020-09-28…  PcsCompu_61:59:2e                    ARP        44 10.0.2.6 is at 08:00
  13 2020-09-28…  ::1                 ::1              UDP        64 41993 → 44886 Len=0
```

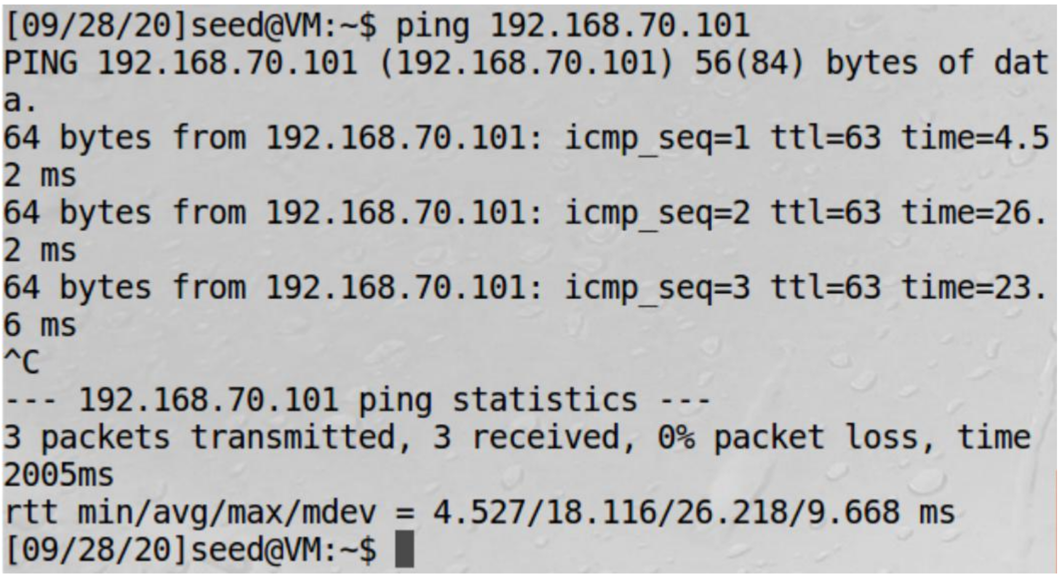
但是服务器端没有返回的 UDP 报文。因为 Linux 中的反向路径过滤机制会对收到的 IP 数据报的源 IP 进行反向路由查找，若 IP 数据报不是来源于该接口，则认为结果不匹配，丢弃该报文。

解决方法是在服务器端添加 192.168.30.0/24 网络与接口 hil0 关联的路由规则，让服务器反向路由查找时查找到设置的接口。



```
[09/28/20]seed@VM:~$ sudo ip route add 192.168.30.0/24 dev hi
l0
[09/28/20]seed@VM:~$ █
```

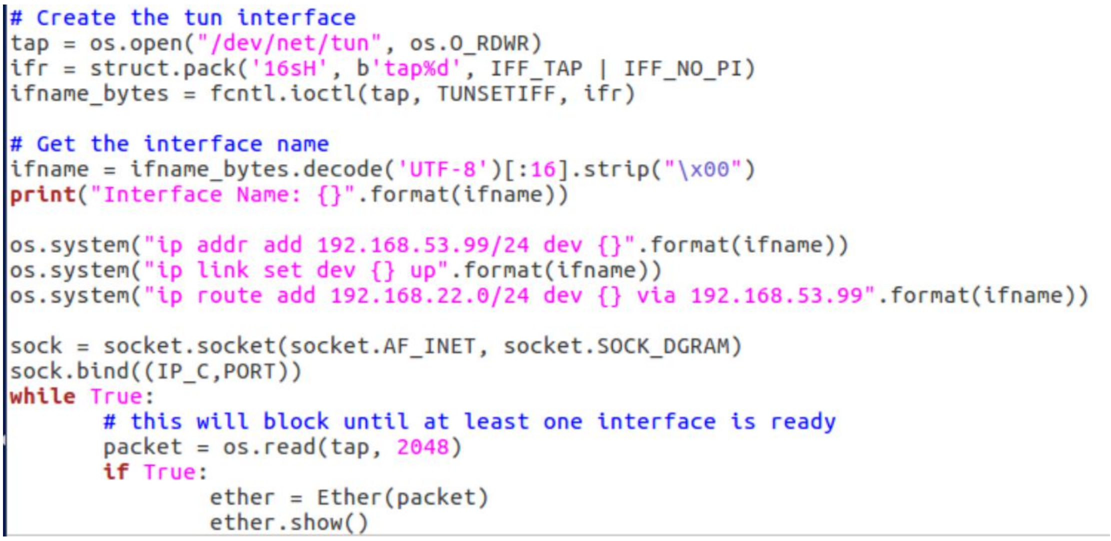
此时主机 U 可以 ping 通主机 V



```
[09/28/20]seed@VM:~$ ping 192.168.70.101
PING 192.168.70.101 (192.168.70.101) 56(84) bytes of dat
a.
64 bytes from 192.168.70.101: icmp_seq=1 ttl=63 time=4.5
2 ms
64 bytes from 192.168.70.101: icmp_seq=2 ttl=63 time=26.
2 ms
64 bytes from 192.168.70.101: icmp_seq=3 ttl=63 time=23.
6 ms
^C
--- 192.168.70.101 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time
2005ms
rtt min/avg/max/mdev = 4.527/18.116/26.218/9.668 ms
[09/28/20]seed@VM:~$ █
```

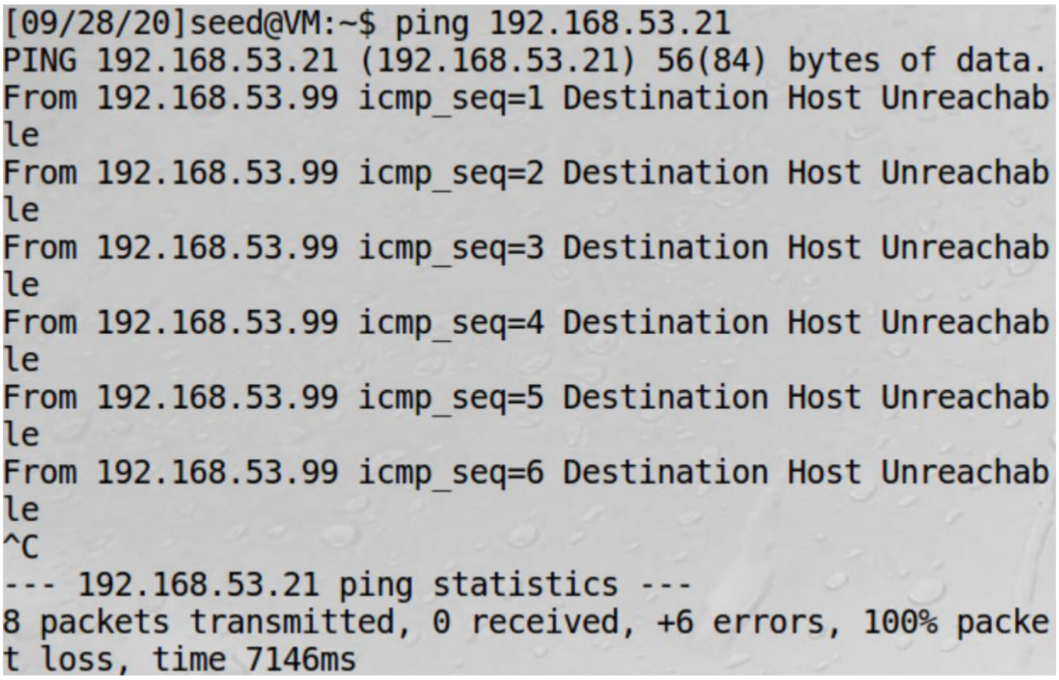
Task 9: Experiment with the TAP Interface

修改 client.py



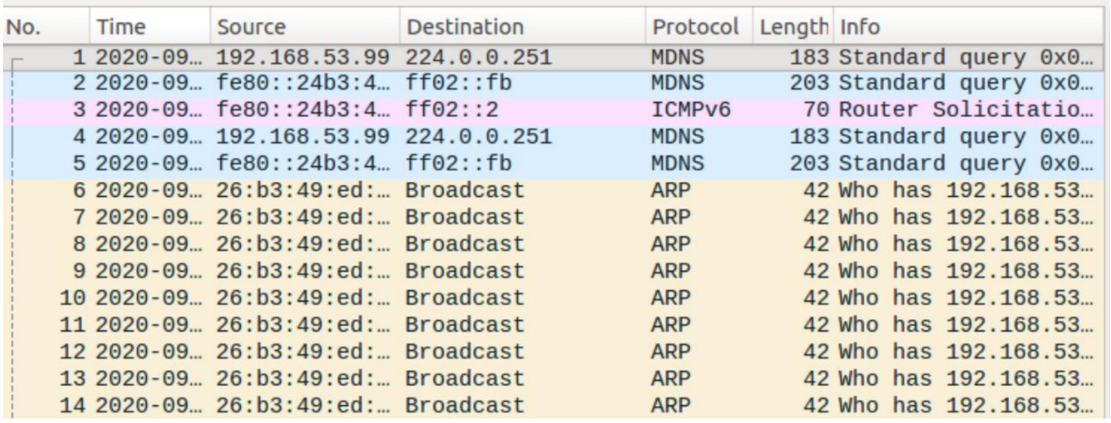
```
# Create the tun interface
tap = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'tap%d', IFF_TAP | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tap, TUNSETIFF, ifr)

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

os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
os.system("ip route add 192.168.22.0/24 dev {} via 192.168.53.99".format(ifname))

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_C,PORT))
while True:
        # this will block until at least one interface is ready
        packet = os.read(tap, 2048)
        if True:
                ether = Ether(packet)
                ether.show()
```

在主机 U 上 ping IP 网段 192.168.53.99/24 里的地址，显示目的主机不可达



```
[09/28/20]seed@VM:~$ ping 192.168.53.21
PING 192.168.53.21 (192.168.53.21) 56(84) bytes of data.
From 192.168.53.99 icmp_seq=1 Destination Host Unreachab
le
From 192.168.53.99 icmp_seq=2 Destination Host Unreachab
le
From 192.168.53.99 icmp_seq=3 Destination Host Unreachab
le
From 192.168.53.99 icmp_seq=4 Destination Host Unreachab
le
From 192.168.53.99 icmp_seq=5 Destination Host Unreachab
le
From 192.168.53.99 icmp_seq=6 Destination Host Unreachab
le
^C
--- 192.168.53.21 ping statistics ---
8 packets transmitted, 0 received, +6 errors, 100% packe
t loss, time 7146ms
```

wireshark 上 tap0 接口显示了许多广播的 ARP 报文



No.	Time	Source	Destination	Protocol	Length	Info
1	2020-09…	192.168.53.99	224.0.0.251	MDNS	183	Standard query 0x0…
2	2020-09…	fe80::24b3:4…	ff02::fb	MDNS	203	Standard query 0x0…
3	2020-09…	fe80::24b3:4…	ff02::2	ICMPv6	70	Router Solicitatio…
4	2020-09…	192.168.53.99	224.0.0.251	MDNS	183	Standard query 0x0…
5	2020-09…	fe80::24b3:4…	ff02::fb	MDNS	203	Standard query 0x0…
6	2020-09…	26:b3:49:ed:…	Broadcast	ARP	42	Who has 192.168.53…
7	2020-09…	26:b3:49:ed:…	Broadcast	ARP	42	Who has 192.168.53…
8	2020-09…	26:b3:49:ed:…	Broadcast	ARP	42	Who has 192.168.53…
9	2020-09…	26:b3:49:ed:…	Broadcast	ARP	42	Who has 192.168.53…
10	2020-09…	26:b3:49:ed:…	Broadcast	ARP	42	Who has 192.168.53…
11	2020-09…	26:b3:49:ed:…	Broadcast	ARP	42	Who has 192.168.53…
12	2020-09…	26:b3:49:ed:…	Broadcast	ARP	42	Who has 192.168.53…
13	2020-09…	26:b3:49:ed:…	Broadcast	ARP	42	Who has 192.168.53…
14	2020-09…	26:b3:49:ed:…	Broadcast	ARP	42	Who has 192.168.53…

tap0 接口广播 ARP 报文查询 192.168.53.21 的 MAC 地址，而 enp0s3 没有回复。因为这是不存在的虚拟网络，所以 ARP 请求不能收到响应。