LAB7: VPN Lab

学号：57118227　　**姓名：**孙浩　　**日期：**2021年7月26日

Container names & IDs

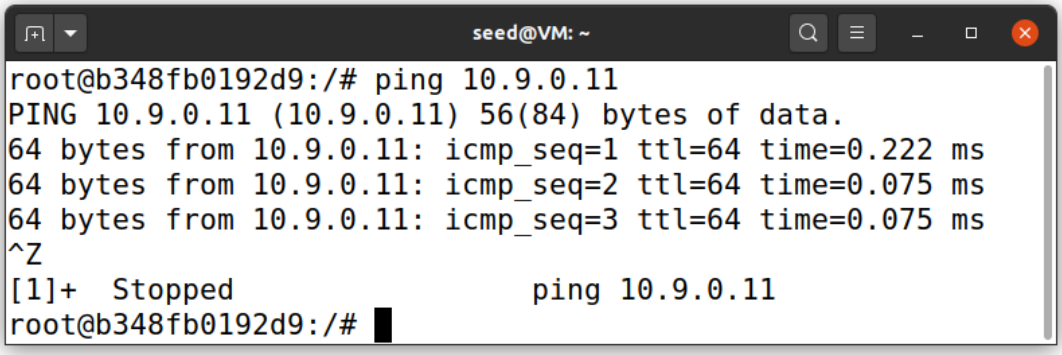
本实验报告在展示结果时用到了较多的命令行终端的窗口截图，为了区分是在哪个Dokcer容器上运行的结果，在这里对这些容器的名字和ID进行一个说明。

name	ID
Host U-10.9.0.5	b348fb0192d9
Host V-192.168.60.5	c05182b10b67
server-router	3cb0dbc80085
192.168.60.6	88cb2edf43e2

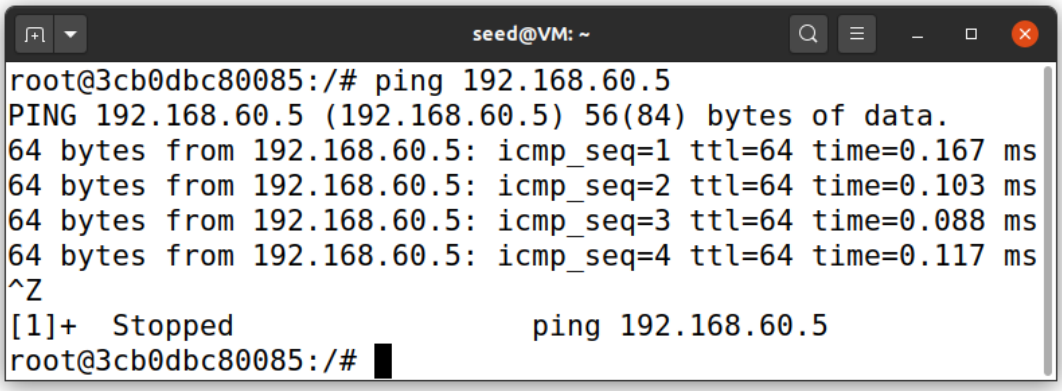
Task 1: Network Setup

Setup步骤同前6次实验相同，接下来进行测试。

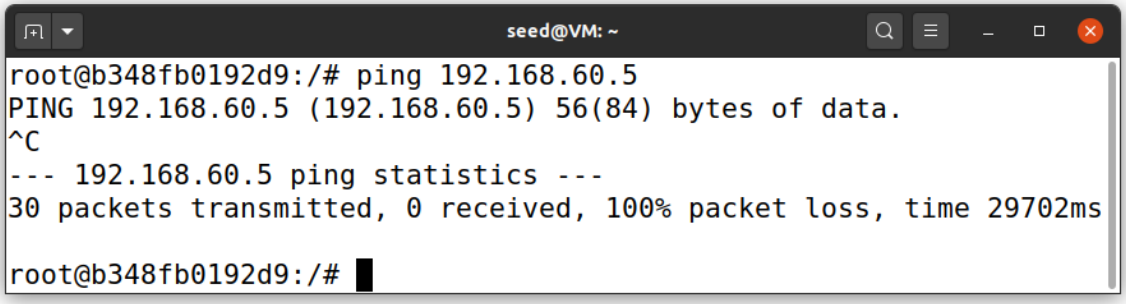
- Host *U* can communicate with VPN Server.



- VPN Server can communicate with Host *V*

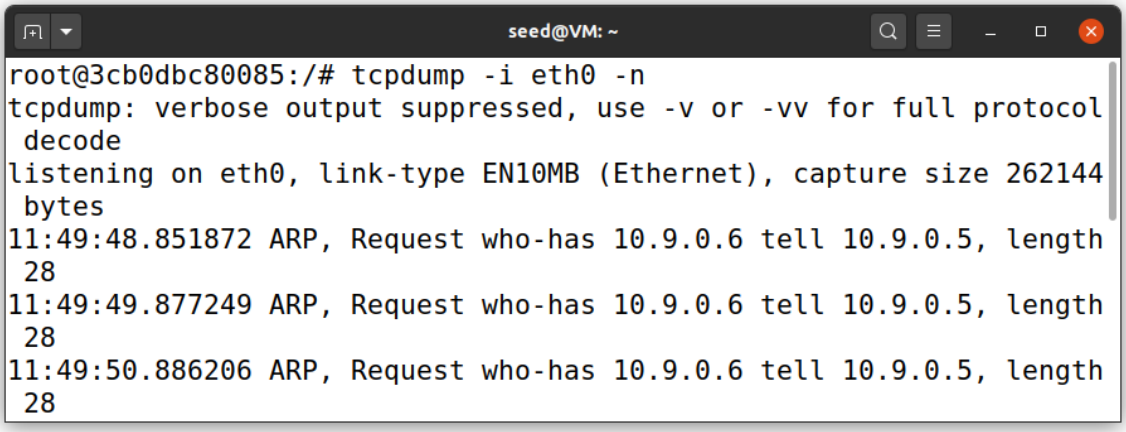


- Host *U* should not be able to communicate with Host *V*.

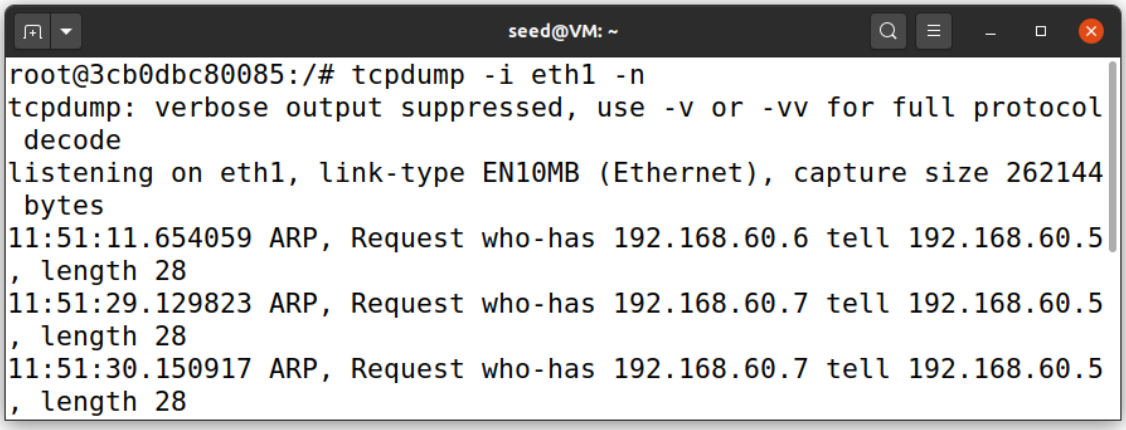


- Run *tcpdump* on the router, and sniff the traffic on each of the network. Show that you can capture packets

10.9.0.0子网：



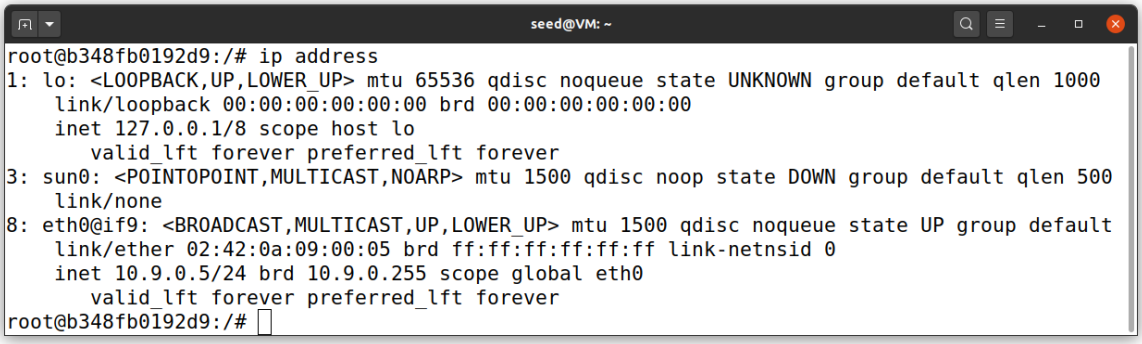
192.168.60.0子网：



Task 2: Create and Configure TUN Interface

Task 2.a: Name of the Interface

在Host U上运行修改后的tun.py，然后查看网络接口，可以看到一个叫sun0的接口。

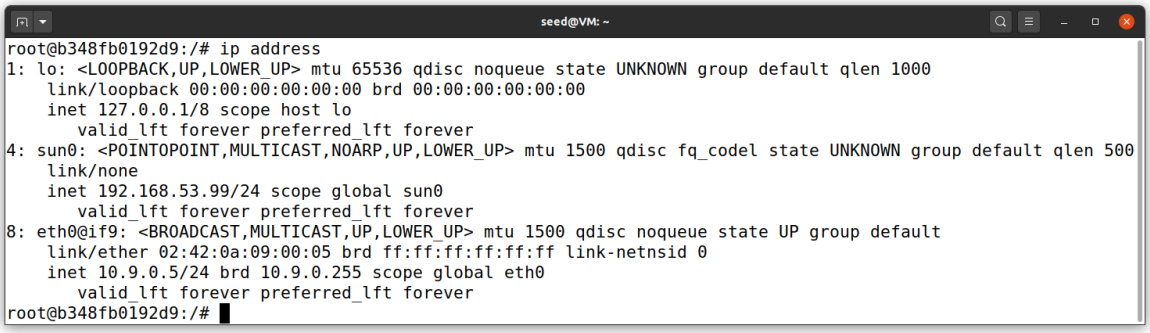


Task 2.b: Set up the TUN Interface

在tun.py末尾增加这样两行代码。

```
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，可以看见这个接口已经具备了IP地址，并且不处于DOWN状态了。



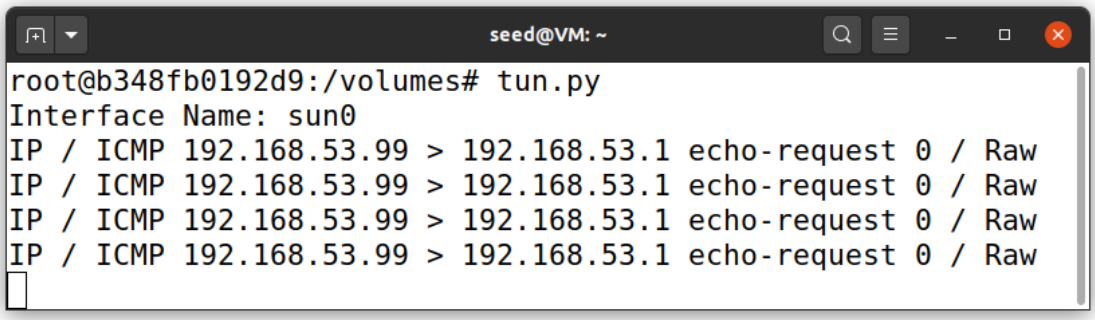
Task 2.c: Read from the TUN Interface

对tun.py原代码最后的while循环部分进行一个修改。

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

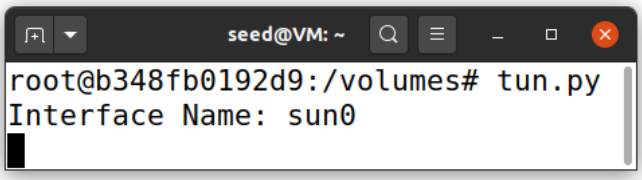
- On Host U, ping a host in the 192.168.53.0/24 netwrok

可以听到响应的报文。



- On Host U, ping a host in the internal network 192.168.60.0/24

不可以听到相应的报文。



这是因为相应报文的目的IP不在TUN接口的网段内。

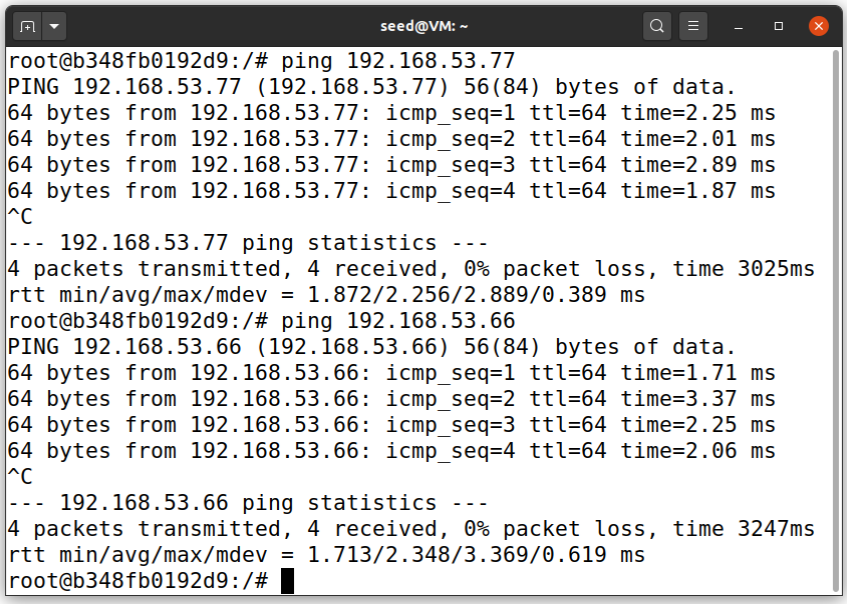
Task 2.d: Write to the TUN Interface

- 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.

更改代码的while True部分如下：

```
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))
```

可以看到此时PING192.168.53.0/24的IP都可以收到回复。

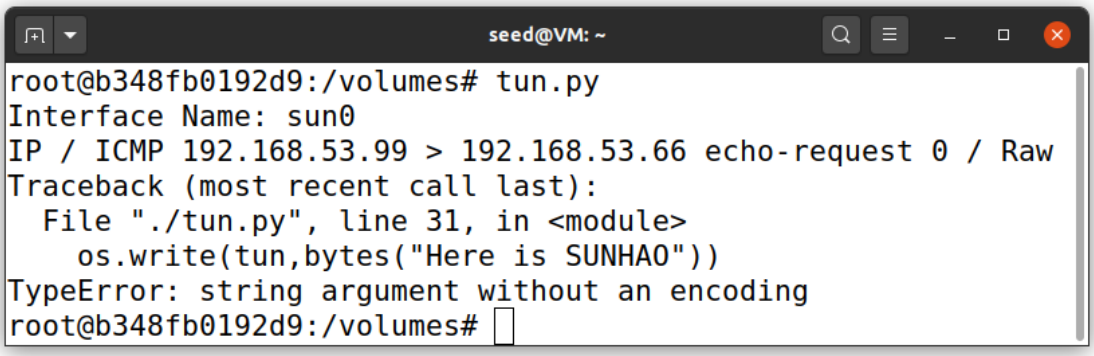


- Instead of writing an IP packet to the interface, write some arbitrary data to the interface, and report your observation

更改while True部分如下。

```
while True:
    packet = os.read(tun,2048)
    if packet:
        pkt = IP(packet)
        print(pkt.summary())
        os.write(tun,bytes("Here is SUNHAO"))
```

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。



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

tun_server.py的代码如下。

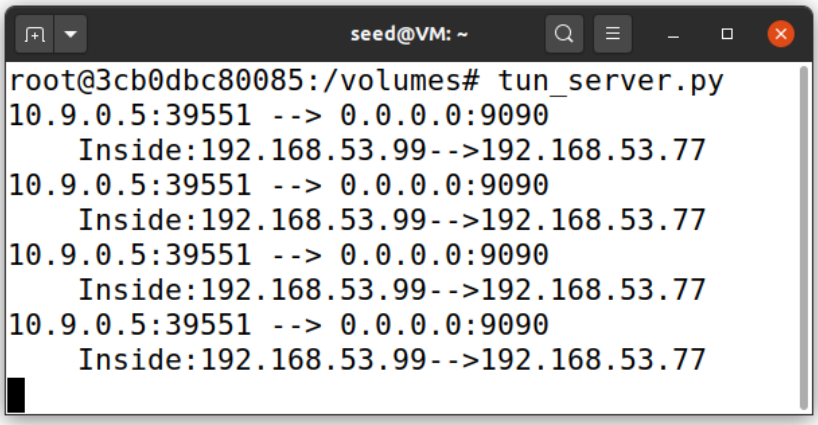
```
#!/usr/bin/env python3
from scapy.all import *

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))
```

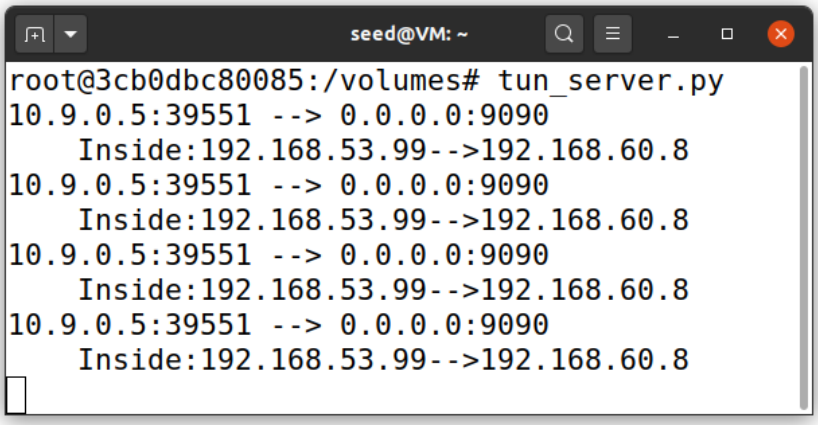
在VPN server上运行tun_server.py，在Host U上运行tun_client.py，然后在U上ping192.168.53.0/24网段的IP，可以看到VPN server上收到了相应的报文。但是ping 192.168.60.0/24网段的IP则没有反应。



为了能够使60网段的报文通过tunnel，还需要增加一条路由。

```
# ip route add 192.168.60.0/24 dev sun0
```

再次重复上述步骤，可以看见相应的报文在VPN server端接收。

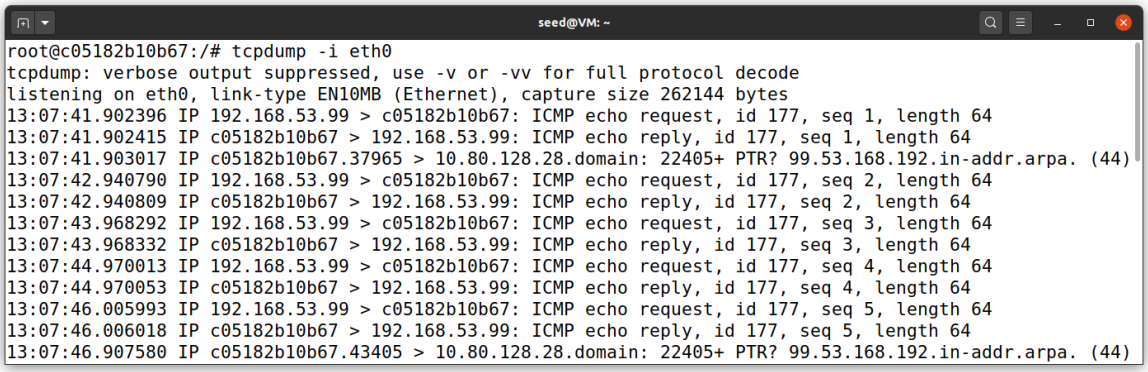


Task 4: Set Up the VPN Server

修改tun_server.py为如下。

```
#!/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'sun%d', IFF_TUN | IFF_NO_PI)
ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)

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

os.system("ip addr add 192.168.53.11/24 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，这一次我们在Host U上ping Host V，同时在Host V上进行tcpdump，可以看到Host V收到了相应的ICMP request，并发出了reply。



Task 5: Handing Traffic in Both Directions

最终版本的tun_client.py如下。

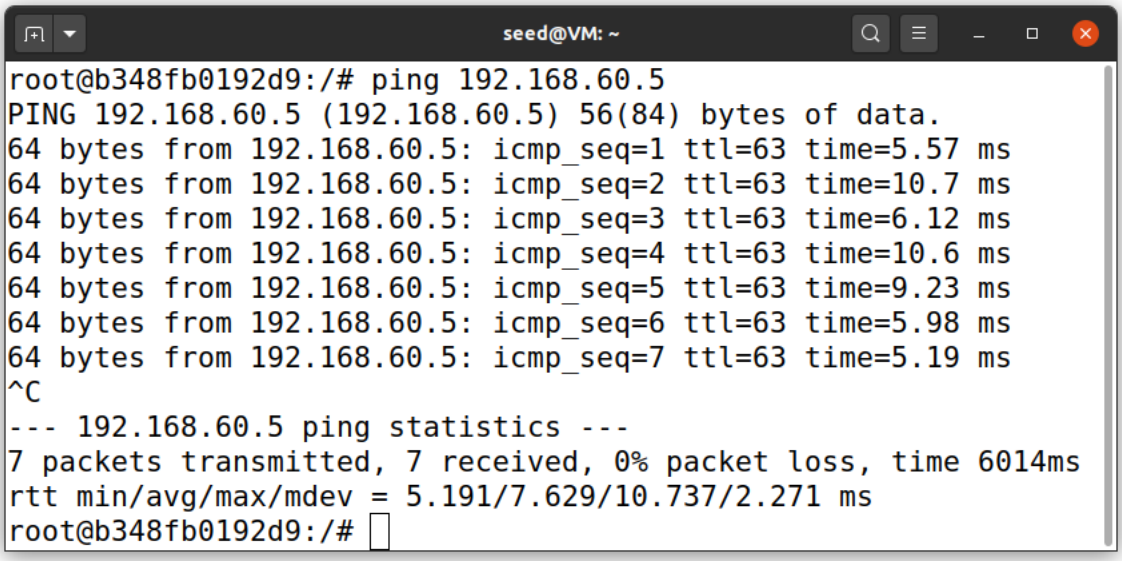
```
#!/usr/bin/env python3

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'sun%d', IFF_TUN | IFF_NO_PI)
ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)

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

os.system("ip addr add 192.168.53.99/24 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))
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)

SERVER_IP = '10.9.0.11'
SERVER_PORT = 9090
fds = [sock,tun]
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))
```

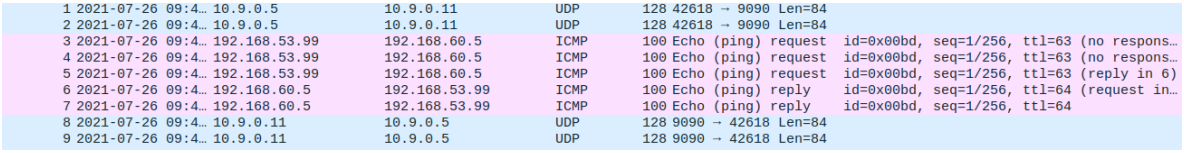
最终版的tun_server.py如下。

```
#!/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'sun%d', IFF_TUN | IFF_NO_PI)
ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)

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

os.system("ip addr add 192.168.53.11/24 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))
```

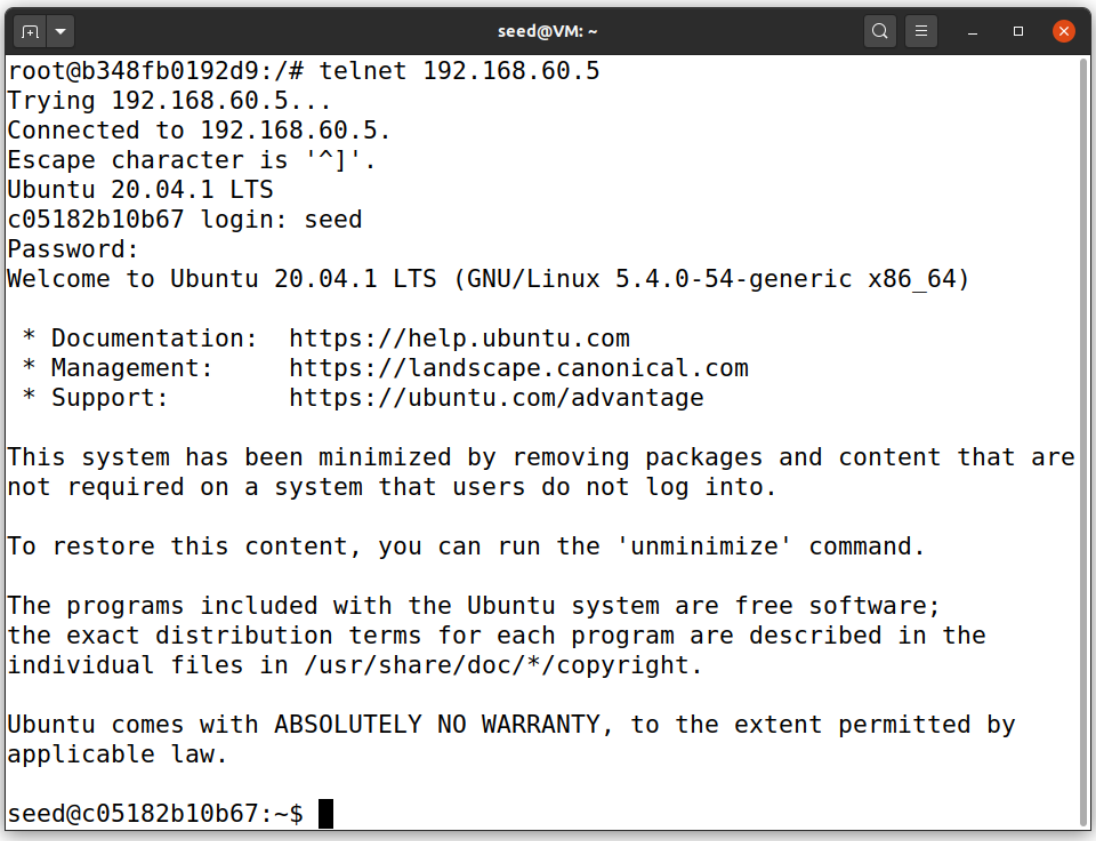
在Host U上ping Host V，可以看到这时已经顺利成功了。



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。



在Host U上telnet Host V，同样成功了。

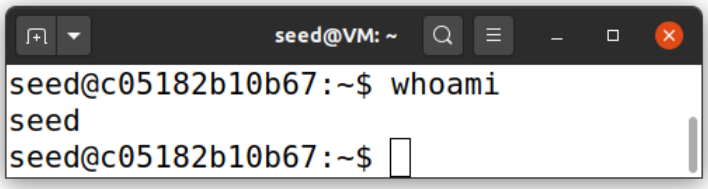


Wireshark同样可以看见，走的是和ping相同的路径，只不过内部变为了TCP协议。

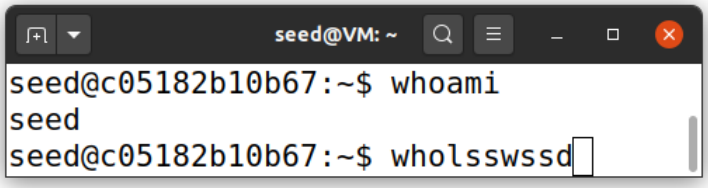
13	2021-07-26 09:4…	10.9.0.5	10.9.0.11	UDP	104 42618 → 9090 Len=60
14	2021-07-26 09:4…	10.9.0.5	10.9.0.11	UDP	104 42618 → 9090 Len=60
15	2021-07-26 09:4…	192.168.53.99	192.168.60.5	TCP	76 58048 → 23 [SYN] Seq=1298747501 Win=64240 Len=0 MSS=1460 SACK…
16	2021-07-26 09:4…	192.168.53.99	192.168.60.5	TCP	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 ...

Task 6: Tunnel-Breaking Experiment

在telnet连接过程中，中断tun_client.py程序，在Host V上键入内容将不会有任何显示。



而当我们重新运行tun_client.py程序，重新建立连接的时候，在中断过程中输入的内容会一次性显示出来。



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。相当于一个生产者-消费者问题，中断程序相当于暂时停止了消费者的工作，只要缓存不溢出，就可以继续正常运行。