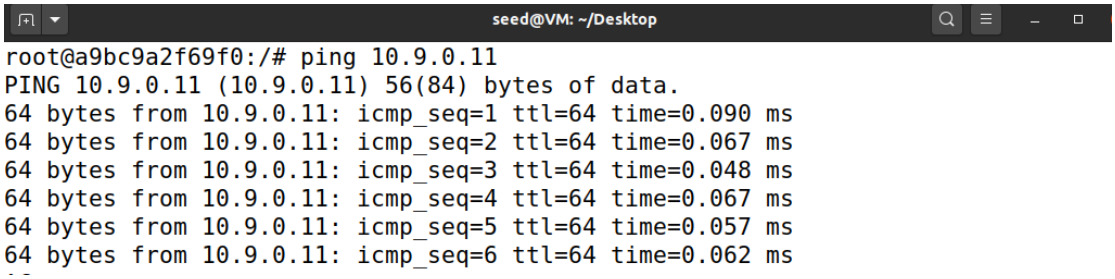
Lab7-report

57118239 张鹏

Task1:

后续的实验中，我们将在客户端和网关之间创建一个 VPN 隧道，用于计算机通过网关访问私有网络。这个实验的目的是测试各个主机、服务器之间的通信状况以确保网络环境的正确。

主机 U 可以和 VPN 服务器通信:



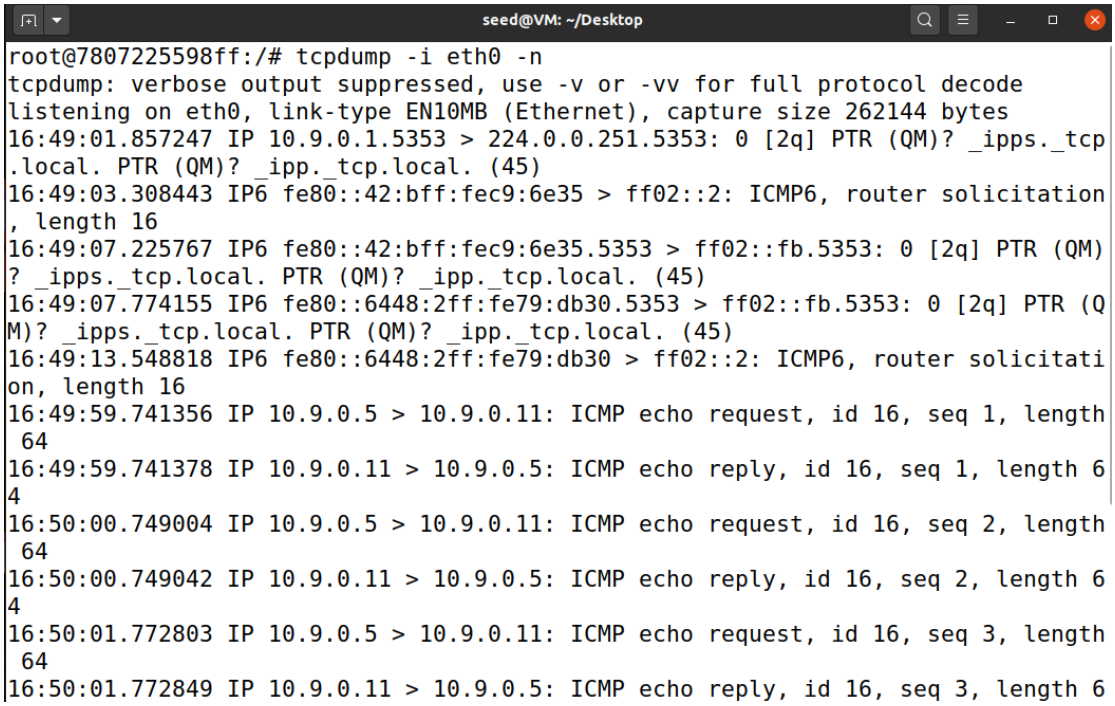
VPN 服务器可以和主机 V 通信:

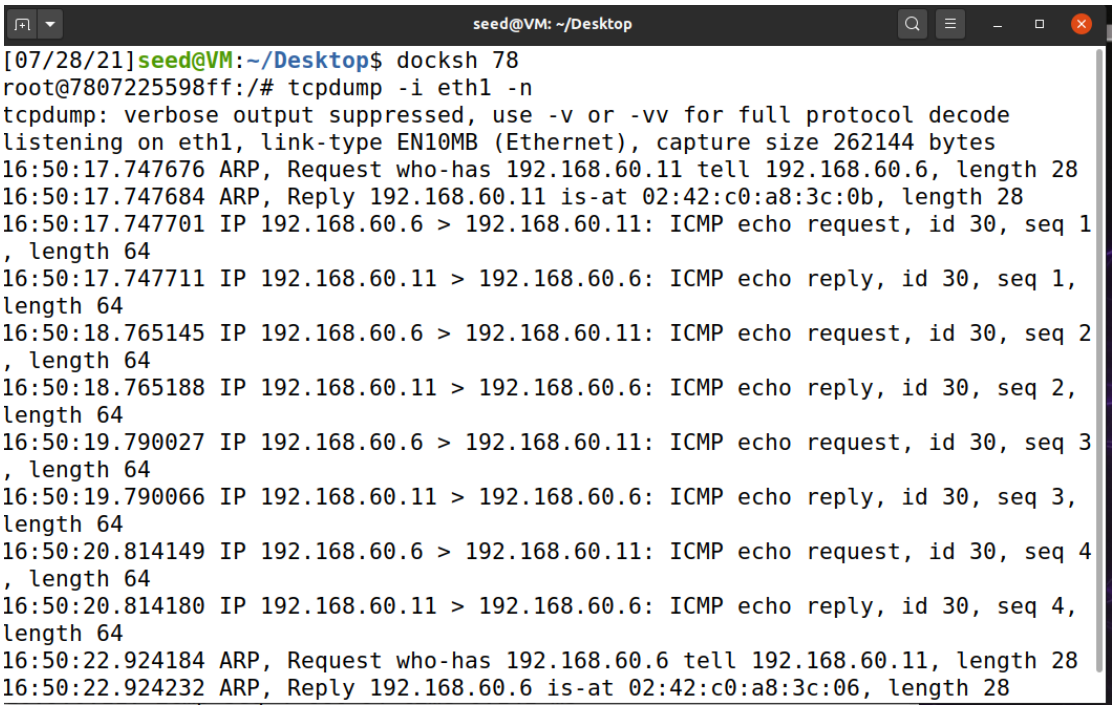
```
root@155a7dfd1875:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=64 time=0.381 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=64 time=0.120 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=64 time=0.177 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=64 time=0.121 ms
```

主机 U 不能和主机 V 通信:

```
root@a9bc9a2f69f0:/# 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 ---
21 packets transmitted, 0 received, 100% packet loss, time 20456ms

root@a9bc9a2f69f0:/#
```

在路由器上运行 tcpdump，嗅探 eth0 和 eth1 上的流量:


seed@VM: ~/Desktop
root@7807225598ff:/# 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
16:49:01.857247 IP 10.9.0.1.5353 > 224.0.0.251.5353: 0 [2q] PTR (QM)? _ipps._tcp
.local. PTR (QM)? _ipp._tcp.local. (45)
16:49:03.308443 IP6 fe80::42:bff:fec9:6e35 > ff02::2: ICMP6, router solicitation
, length 16
16:49:07.225767 IP6 fe80::42:bff:fec9:6e35.5353 > ff02::fb.5353: 0 [2q] PTR (QM)
? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
16:49:07.774155 IP6 fe80::6448:2ff:fe79:db30.5353 > ff02::fb.5353: 0 [2q] PTR (Q
M)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
16:49:13.548818 IP6 fe80::6448:2ff:fe79:db30 > ff02::2: ICMP6, router solicitati
on, length 16
16:49:59.741356 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 16, seq 1, length
64
16:49:59.741378 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 16, seq 1, length 6
4
16:50:00.749004 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 16, seq 2, length
64
16:50:00.749042 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 16, seq 2, length 6
4
16:50:01.772803 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 16, seq 3, length
64
16:50:01.772849 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 16, seq 3, length 6


seed@VM: ~/Desktop
[07/28/21]seed@VM:~/Desktop$ docksh 78
root@7807225598ff:/# 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
16:50:17.747676 ARP, Request who-has 192.168.60.11 tell 192.168.60.6, length 28
16:50:17.747684 ARP, Reply 192.168.60.11 is-at 02:42:c0:a8:3c:0b, length 28
16:50:17.747701 IP 192.168.60.6 > 192.168.60.11: ICMP echo request, id 30, seq 1
, length 64
16:50:17.747711 IP 192.168.60.11 > 192.168.60.6: ICMP echo reply, id 30, seq 1,
length 64
16:50:18.765145 IP 192.168.60.6 > 192.168.60.11: ICMP echo request, id 30, seq 2
, length 64
16:50:18.765188 IP 192.168.60.11 > 192.168.60.6: ICMP echo reply, id 30, seq 2,
length 64
16:50:19.790027 IP 192.168.60.6 > 192.168.60.11: ICMP echo request, id 30, seq 3
, length 64
16:50:19.790066 IP 192.168.60.11 > 192.168.60.6: ICMP echo reply, id 30, seq 3,
length 64
16:50:20.814149 IP 192.168.60.6 > 192.168.60.11: ICMP echo request, id 30, seq 4
, length 64
16:50:20.814180 IP 192.168.60.11 > 192.168.60.6: ICMP echo reply, id 30, seq 4,
length 64
16:50:22.924184 ARP, Request who-has 192.168.60.6 tell 192.168.60.11, length 28
16:50:22.924232 ARP, Reply 192.168.60.6 is-at 02:42:c0:a8:3c:06, length 28

Task2：

该任务的目标是熟悉 TUN/TAP 技术。我们将进行几次实验来了解 TUN/TAP 接口的技术细节。我们将使用下面的 Python 程序作为实验的基础，我们将在整个实验中修改这个基础代码。该代码已经包含在 zip 文件中的 volumes 文件夹中。

Task2.a：

将文档中给的 tun.py 代码中接口名称 tun 改为 Peng。

Open　tun.py　~/Desktop/Labs_20.04/Network Security/VPN Tunneling Lab/Labsetup/volumes

```
#!/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'Peng%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)
```

运行后会创建接口“Peng0”。

```
root@a9bc9a2f69f0:/volumes# chmod a+x tun.py
root@a9bc9a2f69f0:/volumes# tun.py
Interface Name: Peng0
```

可以发现接口信息多了个“Peng0”。

```
root@a9bc9a2f69f0:/# 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: Peng0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group def
ault qlen 500
    link/none
8: eth0@if9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP g
roup 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
root@a9bc9a2f69f0:/#
```

Task2b：

此时的 TUN 接口是不可用的，因为它还没有配置。在使用接口之前，我们需要做两件事。首先，我们需要给它分配一个 IP 地址。其次，我们需要启动接口，因为接口仍然

处于 down 状态。

在程序中为其分配 IP 地址 192.168.53.99，并且设置启动接口。

```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'Peng%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))

while True:
   time.sleep(10)
```

再次运行该程序，发现 Peng0 的接口被分配了 192.168.53.99 的 IP 地址。

```
root@a9bc9a2f69f0:/volumes# tun.py
Interface Name: Peng0
```

```
root@a9bc9a2f69f0:/# 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
4: Peng0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel stat
e UNKNOWN group default qlen 500
    link/none
    inet 192.168.53.99/24 scope global Peng0
       valid_lft forever preferred_lft forever
8: eth0@if9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP g
roup 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
root@a9bc9a2f69f0:/#
```

Task2c:

在这个任务中，我们将从 TUN 接口读取数据。从 TUN 接口输出的是一个 IP 包。我们可以将从接口接收到的数据转换为 Scapy IP 对象，这样就可以打印出 IP 包的每个字段。

该程序输出了定义的 Peng0 接口的报文信息。

```
#!/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'Peng%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))

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

从主机 U 上 ping192.168.53.11 发现有 ICMP 请求报文，没有响应。

```
root@a9bc9a2f69f0:/volumes# tun.py
Interface Name: Peng0
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
```

```
data.
^C
--- 192.168.53.11 ping statistics ---
21 packets transmitted, 0 received, 100% packet los
s, time 20473ms

root@a9bc9a2f69f0:/#
```

在主机 U 上 ping192.168.60.5，程序没有任何输出，因为 192.168.60.0/24 和 192.168.53.0/24 不在同一网段。

```
root@a9bc9a2f69f0:/volumes# tun.py
Interface Name: Peng0
```

```
root@a9bc9a2f69f0:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of da
ta.
```

Task2d：

在这个任务中，我们将写入 TUN 接口。由于这是一个虚拟网络接口，因此应用程序写入接口的内容都将作为 IP 包出现在内核中。我们将修改 tun.py 程序，因此在从 TUN 接口获得一个数据包后，我们根据收到的数据包构造一个新的数据包。然后我们将新包写入 TUN 接口。

修改 tun.py 程序如下，构造 ICMP 回复报文。

```python
while True:
        # Get a packet from the tun interface
        packet = os.read(tun, 2048)
        if packet:
                ip = IP(packet)
                print(ip.summary())
        if 'echo-request' in str(ip.summary()):
                newip = IP(src=ip.dst, dst=ip.src)
                icmp = ICMP(type='echo-reply')
                newpkt = newip/icmp
                os.write(tun, bytes(newpkt))
                print(newpkt.summary())
```

在主机 U 上 ping192.168.53.5，发现有回复报文。

```
seed@VM: ~/Desktop
root@d4232afd94ee:/volumes# python3 tun.py
Interface Name: Peng0
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.5 > 192.168.53.99 echo-reply 0
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.5 > 192.168.53.99 echo-reply 0
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.5 > 192.168.53.99 echo-reply 0
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.5 > 192.168.53.99 echo-reply 0
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.5 > 192.168.53.99 echo-reply 0
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.5 > 192.168.53.99 echo-reply 0
```

```
seed@VM: ~/Desktop
root@d4232afd94ee:/# ping 192.168.53.5
PING 192.168.53.5 (192.168.53.5) 56(84) bytes of data.
^C
--- 192.168.53.5 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5097ms

root@d4232afd94ee:/#
```

在 TUN 里将报文 reply 改写为本人偏爱的字符串‘nishino-nanase‘。

```python

while True:
        # Get a packet from the tun interface
        packet = os.read(tun, 2048)
        if packet:
                ip = IP(packet)
                print(ip.summary())
        if 'echo-request' in str(ip.summary()):
                os.write(tun, bytes('nishino-nanase'.encode('utf-8')))
                packet = os.read(tun,2048)
                ip = IP(packet)
```

发现只有请求报文，没有响应报文。

```
seed@VM: ~/Desktop
root@d4232afd94ee:/volumes# python3 tun.py
Interface Name: Peng0
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
```

```
seed@VM: ~/Desktop
root@d4232afd94ee:/# ping 192.168.53.5
PING 192.168.53.5 (192.168.53.5) 56(84) bytes of data.
```

Task3：

在本任务中，我们将从 TUN 接口接收到的 IP 数据包放入一个新的 IP 数据包的 UDP 有效载荷字段中，发送给另一台计算机，这被称为 IP 隧道。隧道实现只是标准的客户端/服务器编程。它可以建立在 TCP 或 UDP 之上。在本任务中，我们将使用 UDP。即，我们把一个 IP 数据包放在一个 UDP 数据包的有效负载字段中。

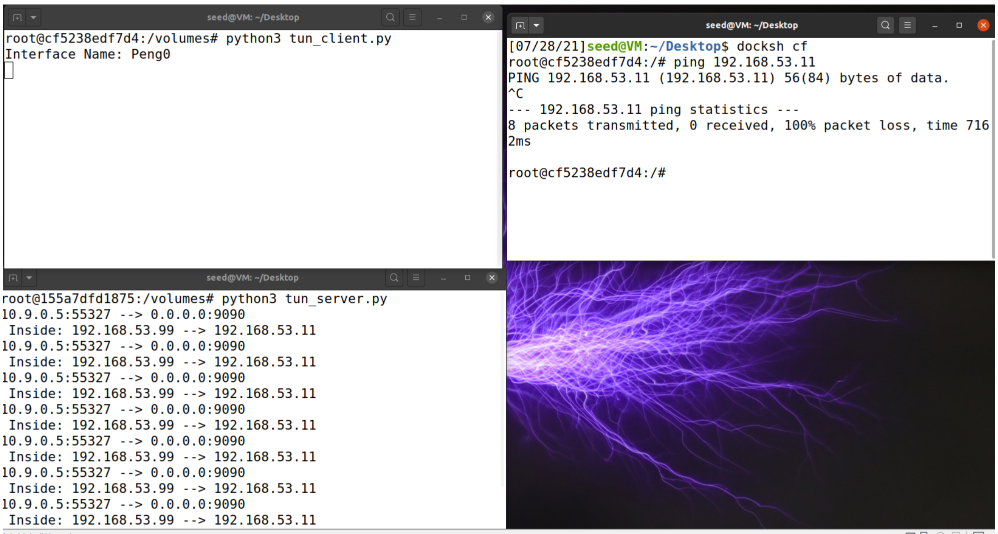
给出的 tun_client.py 代码如下：

```
import time
from scapy.all import *
import socket

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'Peng%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))
# 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 packet:
        # Send the packet via the tunnel
                sock.sendto(packet, ('10.9.0.11', 9090))
```

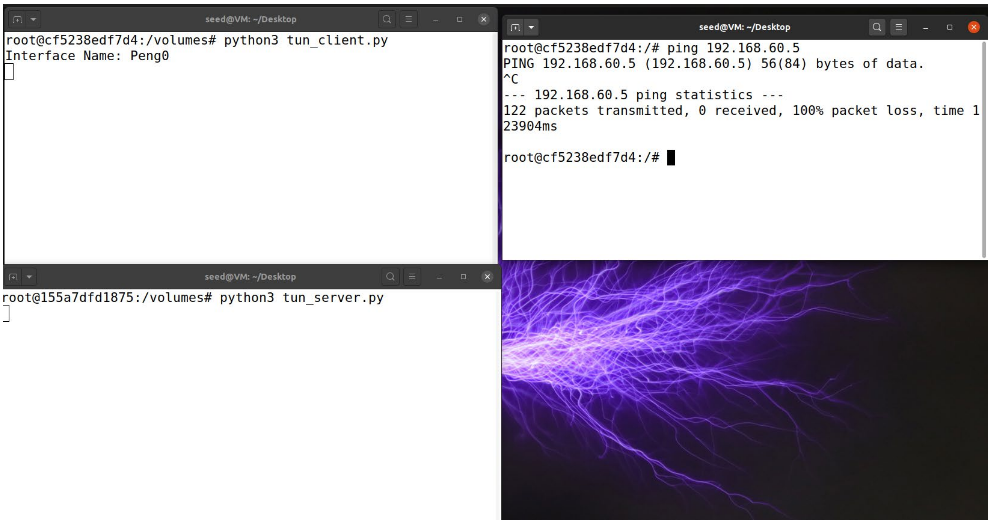
给出的 tun_server.py 代码如下：

```
#!/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))
```

在主机 U 上 ping 路由器的 192.168.53.11，可以发现 server 收到了请求报文。



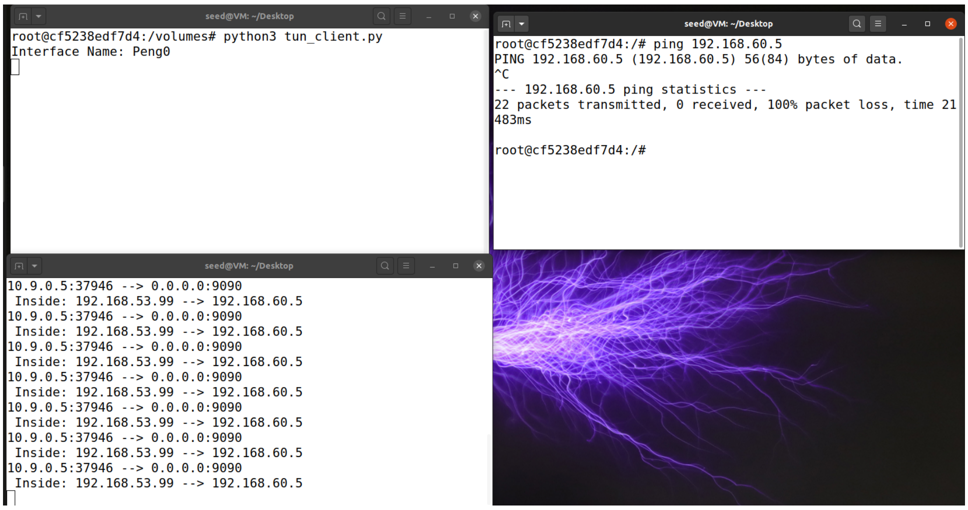
在主机 U 上 ping 主机 V，发现 server 上没有任何信息输出。



在客户端上添加去往 192.168.60.0/24 网段的路由。

```
root@cf5238edf7d4:/# ip route add 192.168.60.0/24 dev Peng0 via 192.168.53.99
root@cf5238edf7d4:/#
```

添加路由条目后再次测试，发现 router 上有信息输出，可以发现，此时却是从 ICMP 报文从隧道中经过。

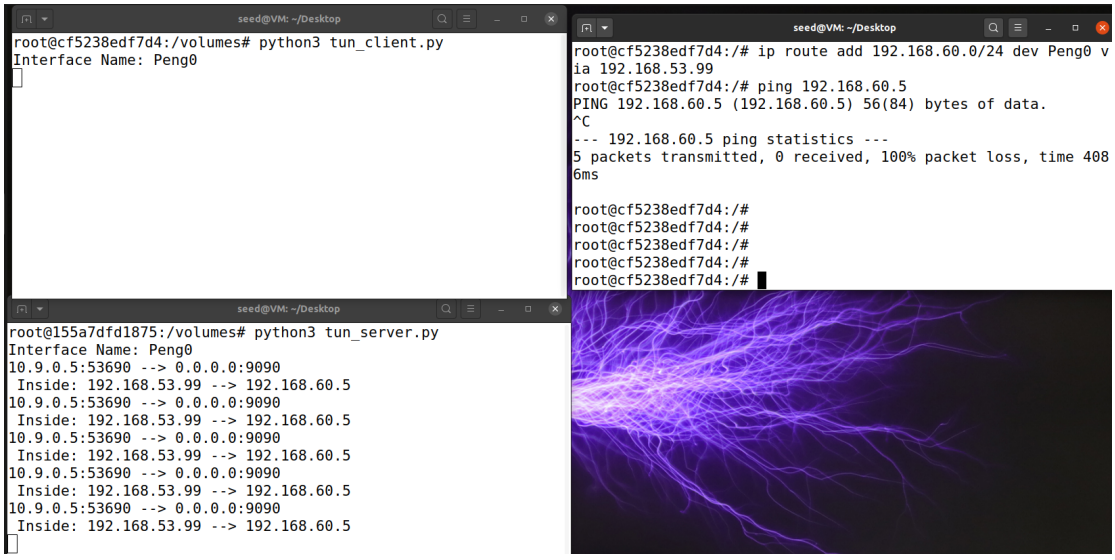


Task4：

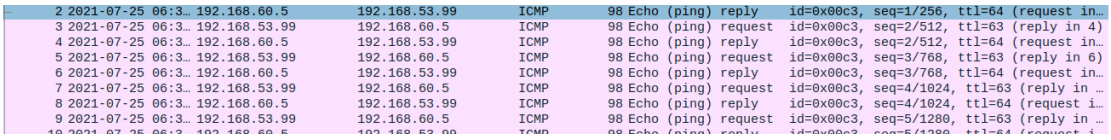
首先修改 tun_server.py 代码如下，将接口 Peng0 处分配 IP：192.168.53.5/24

```
Open   tun_server.py   ~/Desktop/Labs_20.04/Network Security/VPN Tunneling Lab/Labsetup/volumes   Save
 4 import os
 5 import time
 6 from scapy.all import *
 7 import socket
 8 IP_A = "0.0.0.0"
 9 PORT = 9090
10
11 TUNSETIFF = 0x400454ca
12 IFF_TUN   = 0x0001
13 IFF_TAP   = 0x0002
14 IFF_NO_PI = 0x1000
15
16 # Create the tun interface
17 tun = os.open("/dev/net/tun", os.O_RDWR)
18 ifr = struct.pack('16sH', b'Peng%d', IFF_TUN | IFF_NO_PI)
19 ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)
20
21 # Get the interface name
22 ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
23 print("Interface Name: {}".format(ifname))
24 os.system("ip addr add 192.168.53.5/24 dev {}".format(ifname))
25 os.system("ip link set dev {} up".format(ifname))
26
27 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
28 sock.bind((IP_A, PORT))
29 while True:
30         data, (ip, port) = sock.recvfrom(2048)
31         print("{}:{} --> {}:{}".format(ip, port, IP_A, PORT))
32         pkt = IP(data)
33         print(" Inside: {} --> {}".format(pkt.src, pkt.dst))
34         os.write(tun,bytes(pkt))
```

运行客户端和服务器程序，在主机 U 上 ping 主机 V，在 route 上有信息显示发往了 192.168.60.5，但是没有任何回应。



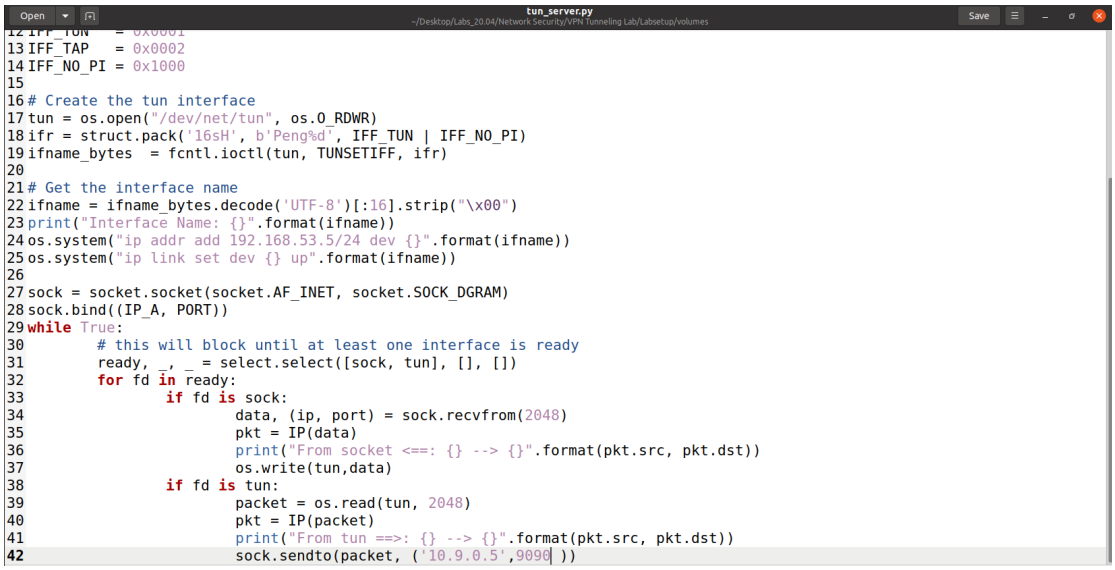
同时通过 wireshark 抓包，可以发现是从 192.168.53.99 发往 192.168.60.5 的，隧道搭建成功。



Task5：

到达这一点后，隧道的一个方向就完成了，也就是说，我们可以通过隧道将数据包从主机 U 发送到主机 V。如果我们查看 Host V 上的 Wireshark 跟踪，我们可以看到 Host V 已经发送了响应，但是数据包被丢到了某个地方。这是因为我们的隧道只有一个方向；我们需要设置它的另一个方向，这样返回的流量可以通过隧道回到主机 U。

首先是 tun_server.py，对 while 部分进行修改，代码如下：

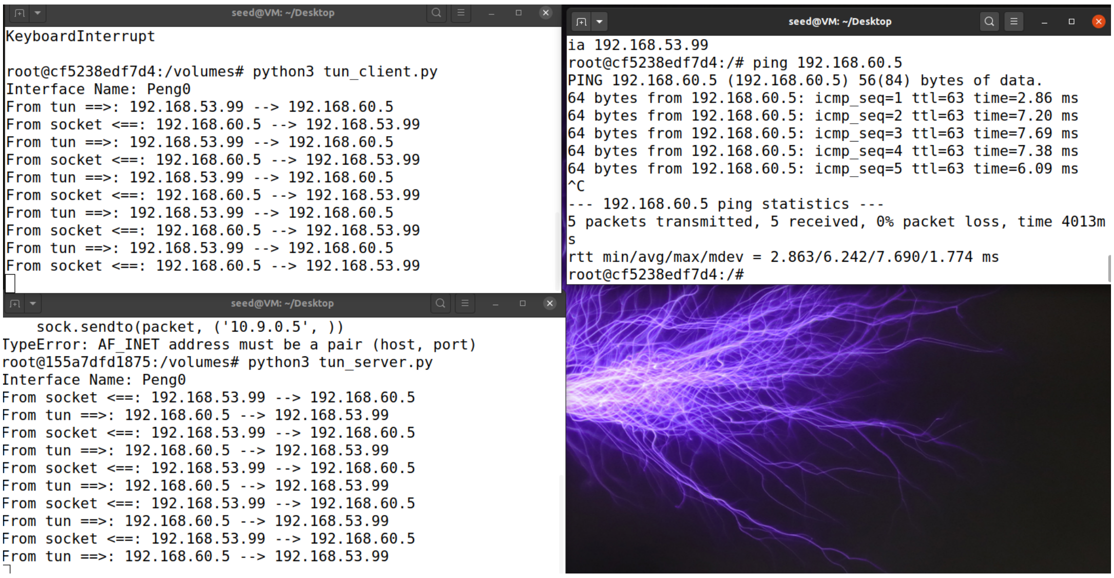



```
IFF_TUN   = 0x0001
IFF_TAP   = 0x0002
IFF_NO_PI = 0x1000

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'Peng%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 192.168.53.5/24 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:
        # this will block until at least one interface is ready
        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,data)
                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 ))
```

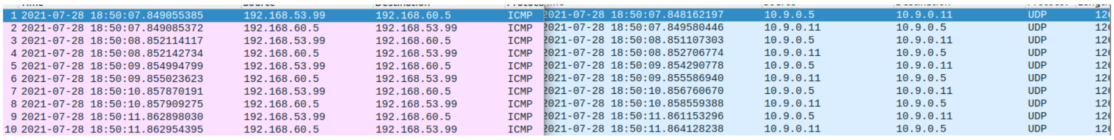
然后是 tun_client.py，代码如下：


```
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'Peng%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))
# Create UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))
while True:
        # this will block until at least one interface is ready
        ready, _, _ = select.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, ('10.9.0.11', 9090))
```

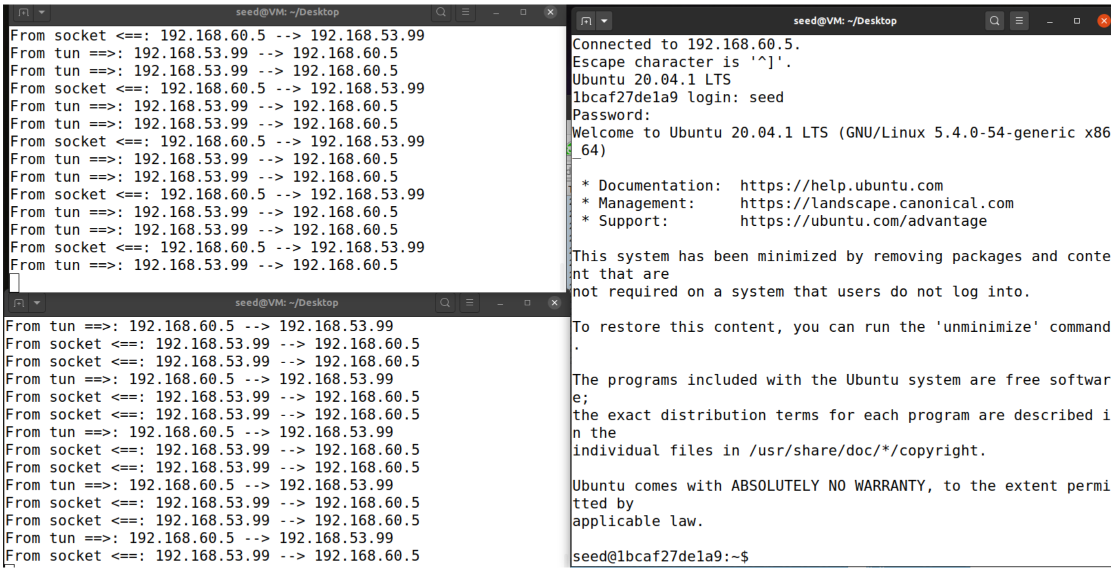
在主机 U 上 ping 主机 V，可以 ping 通，并且在 server 和 client 程序上都有接口信息的输出。



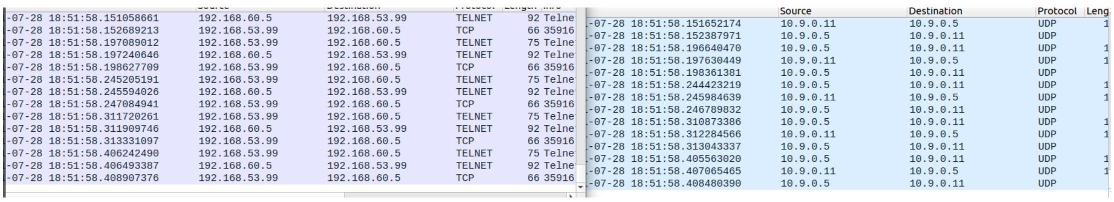
通过 wireshark 抓包，发现收到并发送了 ICMP 报文，且其被封装在 UDP 报文中。



在主机 U 上 telnet 到主机 V，发现可以成功 telnet。



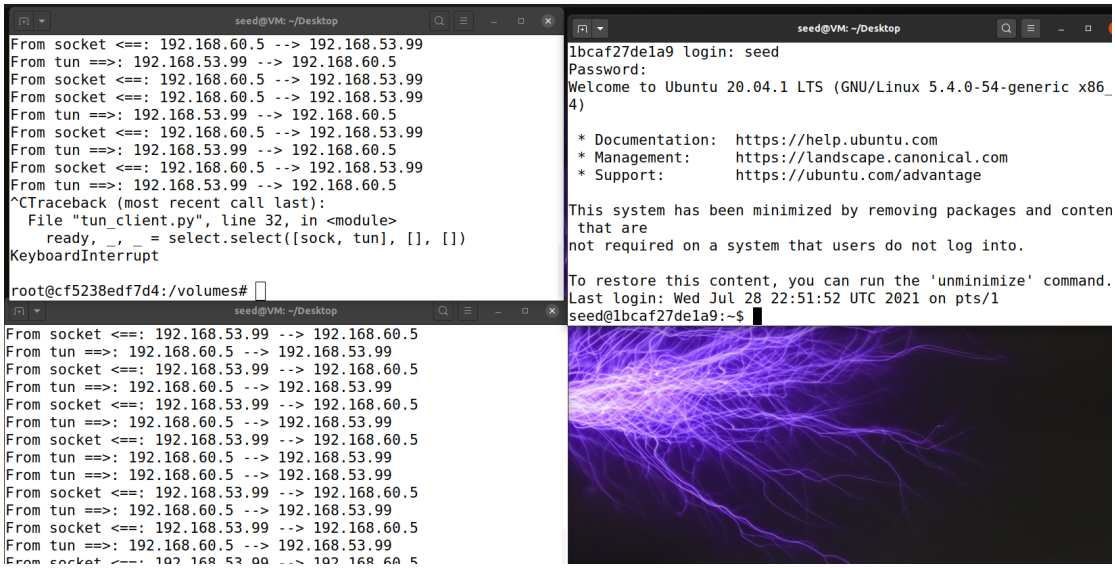
通过抓包可以观察到一样是被封装在 UDP 报文中。



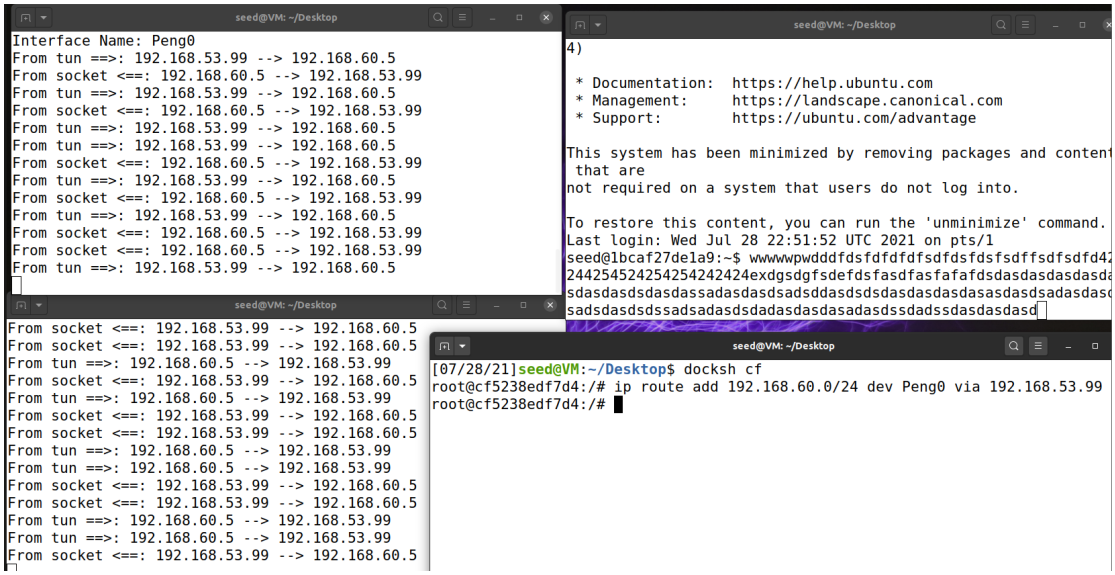
Task6：

这个实验的目的是测试当主机 U 通过隧道 telnet 到主机 V，停止客户端或者服务器程序来中断 VPN 隧道后，会发生什么，重启程序后会发生什么。

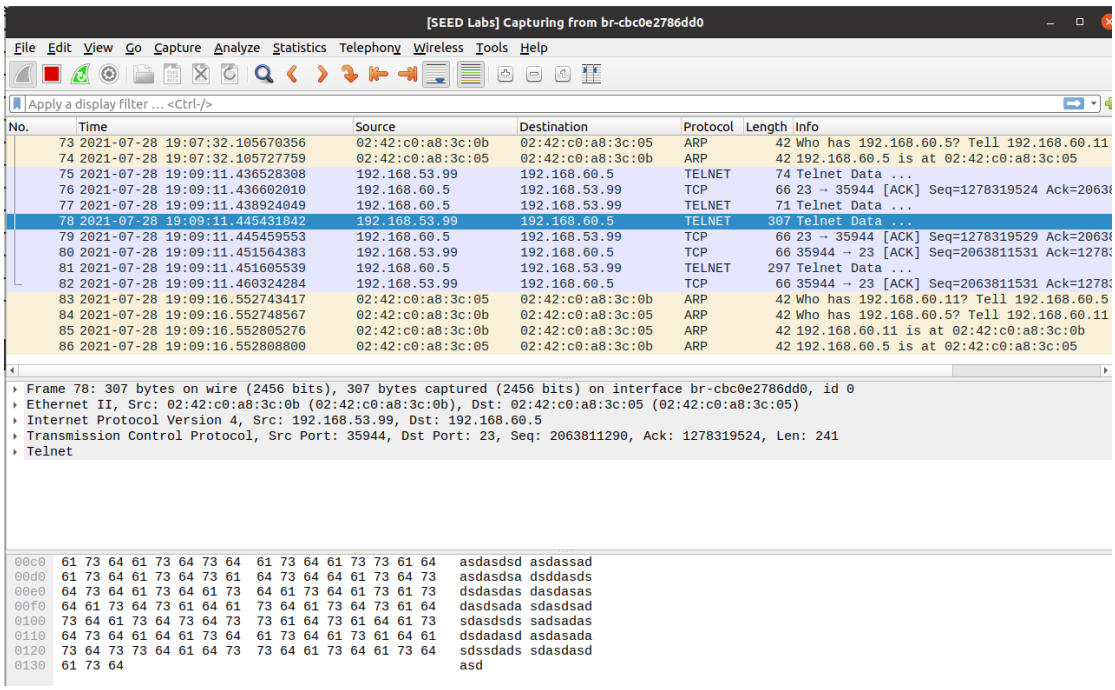
首先打开隧道，可以成功 telnet，之后关闭 tun_client.py，发现在 shell 输入命令什么也输入不了，连接断掉了。



重新运行 tun_client.py 程序，发现 telnet 的连接恢复，刚刚输入的东西全部显示出来了。



抓包结果也证明了其恢复了连接。



总结:

在本次实验中，我们学习了有关 VPN 隧道搭建的基本知识，理解了通过将报文封装在 UDP 或 TCP 报文下，通过隧道来实现对私有网络的访问。在实验中，学习了接口的创建和配置，对于 VPN 的工作原理有了更深刻的认识，客户端服务器结构的通信方式也有了更深的认识。