

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：

```
root@a9bc9a2f69f0:/# 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=0.090 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.067 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.048 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.067 ms
64 bytes from 10.9.0.11: icmp_seq=5 ttl=64 time=0.057 ms
64 bytes from 10.9.0.11: icmp_seq=6 ttl=64 time=0.062 ms
```

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：

```
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:fea9:6e35 > ff02::2: ICMP6, router solicitation
, length 16
16:49:07.225767 IP6 fe80::42:bff:fea9:6e35.5353 > 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 (QM)?
_ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
16:49:13.548818 IP6 fe80::6448:2ff:fe79:db30 > ff02::2: ICMP6, router solicitation
, 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 64
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 64
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 64
[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。



```
1#!/usr/bin/env python3
2
3import fcntl
4import struct
5import os
6import time
7from scapy.all import *
8
9TUNSETIFF = 0x400454ca
10IFF_TUN = 0x0001
11IFF_TAP = 0x0002
12IFF_NO_PI = 0x1000
13
14# Create the tun interface
15tun = os.open("/dev/net/tun", os.O_RDWR)
16ifr = struct.pack('16sH', b'Peng\0', IFF_TUN | IFF_NO_PI)
17ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
18
19# Get the interface name
20ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
21print("Interface Name: {}".format(ifname))
22
23while True:
24    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 default 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 default qlen 500
    link/none
8: eth0@if9: <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
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。

```
1#!/usr/bin/env python3
2
3 import fcntl
4 import struct
5 import os
6 import time
7 from scapy.all import *
8
9 TUNSETIFF = 0x400454ca
10 IFF_TUN = 0x0001
11 IFF_TAP = 0x0002
12 IFF_NO_PI = 0x1000
13
14 # Create the tun interface
15 tun = os.open("/dev/net/tun", os.O_RDWR)
16 ifr = struct.pack('16sH', b'Peng%d', IFF_TUN | IFF_NO_PI)
17 fname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
18
19 # Get the interface name
20 fname = fname_bytes.decode('UTF-8')[:16].strip("\x00")
21 print("Interface Name: {}".format(fname))
22 os.system("ip addr add 192.168.53.99/24 dev {}".format(fname))
23 os.system("ip link set dev {} up".format(fname))
24
25 while True:
26     time.sleep(10)
27
```

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 default
    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 state 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 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
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。

```
Open Save <|> X
tun.py
~/Desktop/Labs_20.04/Network Security/VPN Tunneling Lab/Labsetup/volumes

1#!/usr/bin/env python3
2
3 import fcntl
4 import struct
5 import os
6 import time
7 from scapy.all import *
8
9 TUNSETIFF = 0x400454ca
10 IFF_TUN = 0x0001
11 IFF_TAP = 0x0002
12 IFF_NO_PI = 0x1000
13
14 # Create the tun interface
15 tun = os.open("/dev/net/tun", os.O_RDWR)
16 ifr = struct.pack('16s', b'Peng%d', IFF_TUN | IFF_NO_PI)
17 ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
18
19 # Get the interface name
20 ifname = ifname_bytes.decode('UTF-8')[16:].strip("\x00")
21 print("Interface Name: {}".format(ifname))
22 os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
23 os.system("ip link set dev {} up".format(ifname))
24
25 while True:
26     # Get a packet from the tun interface
27     packet = os.read(tun, 2048)
28     if packet:
29         ip = IP(packet)
30         print(ip.summary())
31
```

从主机 U 上 ping 192.168.53.11 发现有 ICMP 请求报文，没有响应。

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 data.
```

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。

```
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。

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

```
14
15 while True:
16     # Get a packet from the tun interface
17     packet = os.read(tun, 2048)
18     if packet:
19         ip = IP(packet)
20         print(ip.summary())
21     if 'echo-request' in str(ip.summary()):
22         os.write(tun, bytes('nishino-nanase'.encode('utf-8')))
23         packet = os.read(tun,2048)
24         ip = IP(packet)
```

发现只有请求报文，没有响应报文。

```
root@d4232af94ee:/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
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
root@d4232af94ee:/# 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。

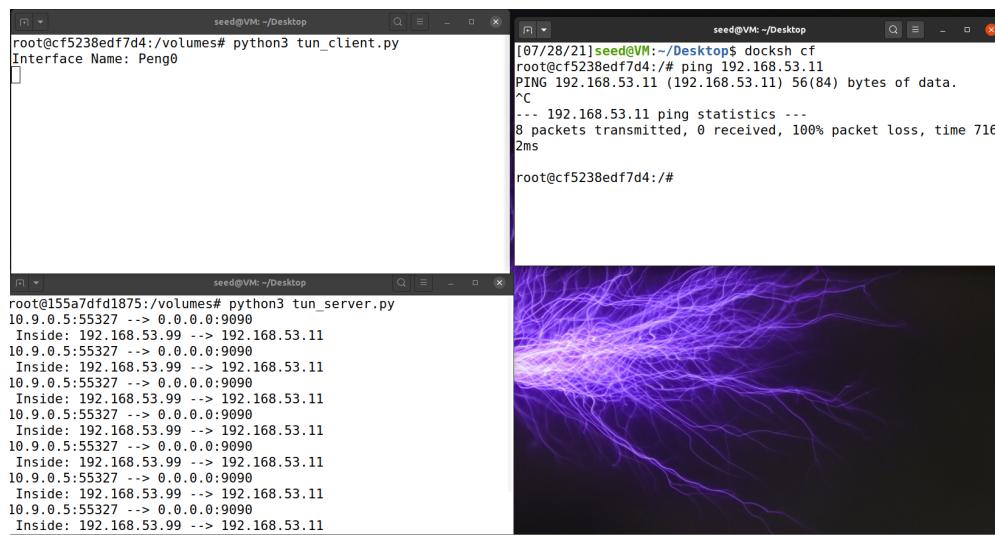
给出的 tun_client.py 代码如下：

```
6 import time
7 from scapy.all import *
8 import socket
9
10 TUNSETIFF = 0x400454ca
11 IFF_TUN   = 0x0001
12 IFF_TAP   = 0x0002
13 IFF_NO_PI = 0x1000
14
15 # Create the tun interface
16 tun = os.open("/dev/net/tun", os.O_RDWR)
17 ifr = struct.pack('16sH', b'Peng%d', IFF_TUN | IFF_NO_PI)
18 ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
19
20 # Get the interface name
21 ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
22 print("Interface Name: {}".format(ifname))
23 os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
24 os.system("ip link set dev {} up".format(ifname))
25 # Create UDP socket
26 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
27 while True:
28     # Get a packet from the tun interface
29     packet = os.read(tun, 2048)
30     if packet:
31         # Send the packet via the tunnel
32         sock.sendto(packet, ('10.9.0.11', 9090))
33
```

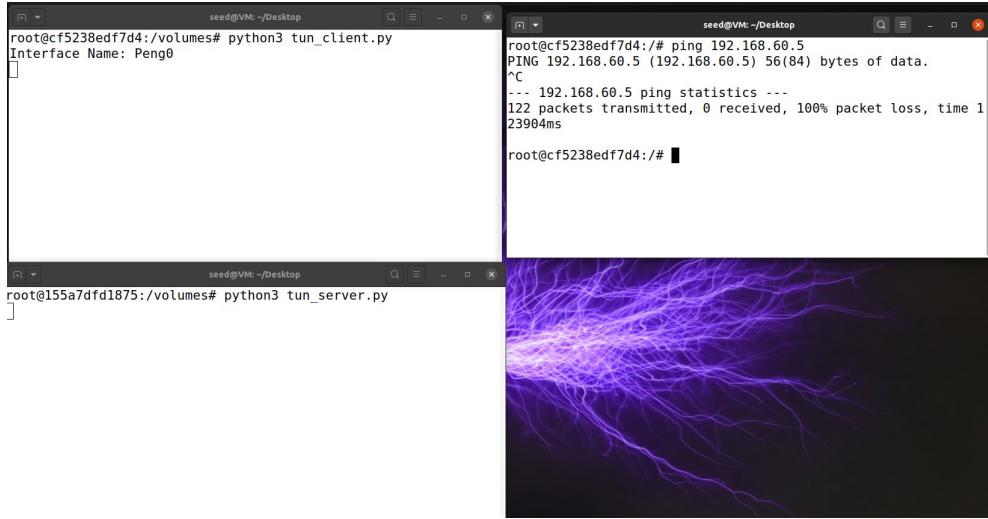
给出的 tun_server.py 代码如下：

```
1#!/usr/bin/env python3
2from scapy.all import *
3IP_A = "0.0.0.0"
4PORT = 9090
5sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
6sock.bind((IP_A, PORT))
7while True:
8    data, (ip, port) = sock.recvfrom(2048)
9    print("{}:{} --> {}:{}".format(ip, port, IP_A, PORT))
10   pkt = IP(data)
11   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。



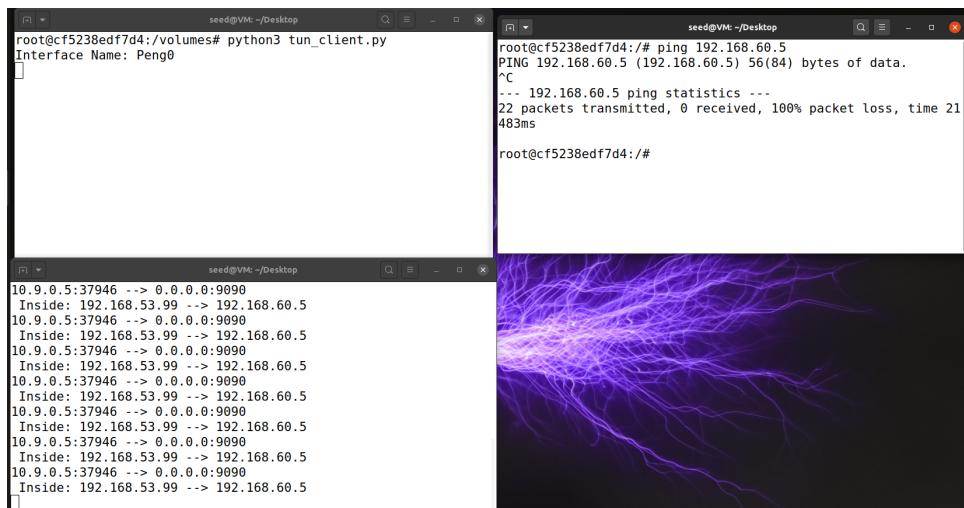
在主机 U 上 ping 主机 V，发现 server 上没有任何信息输出。



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 ▾ Save ▾ tun_server.py
1 import os
2 import time
3 from scapy.all import *
4 import socket
5 IP_A = "0.0.0.0"
6 PORT = 9090
7
8 TUNSETIFF = 0x400454ca
9 IFF_TUN = 0x0001
10 IFF_TAP = 0x0002
11 IFF_NO_PI = 0x1000
12
13 # Create the tun interface
14 tun = os.open("/dev/net/tun", os.O_RDWR)
15 ifr = struct.pack('16sH', b'Peng0\0', IFF_TUN | IFF_NO_PI)
16 ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
17
18 # Get the interface name
19 ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
20 print("Interface Name: {}".format(ifname))
21 os.system("ip addr add 192.168.53.5/24 dev {}".format(ifname))
22 os.system("ip link set dev {} up".format(ifname))
23
24 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
25 sock.bind((IP_A, PORT))
26 while True:
27     data, (ip, port) = sock.recvfrom(2048)
28     print("{}:{} --> {}:{}".format(ip, port, IP_A, PORT))
29     pkt = IP(data)
30     print(" Inside: {} --> {}".format(pkt.src, pkt.dst))
31     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。

The screenshot shows three terminal windows on a Linux desktop. The top-left window shows the command `python3 tun_client.py` being run, with output indicating the interface name is Peng0. The top-right window shows the command `ip route add 192.168.60.0/24 dev Peng0 via 192.168.53.99` followed by a ping command to 192.168.60.5. The bottom-left window shows the command `python3 tun_server.py` being run, with multiple log entries showing traffic between 192.168.53.99 and 192.168.60.5.

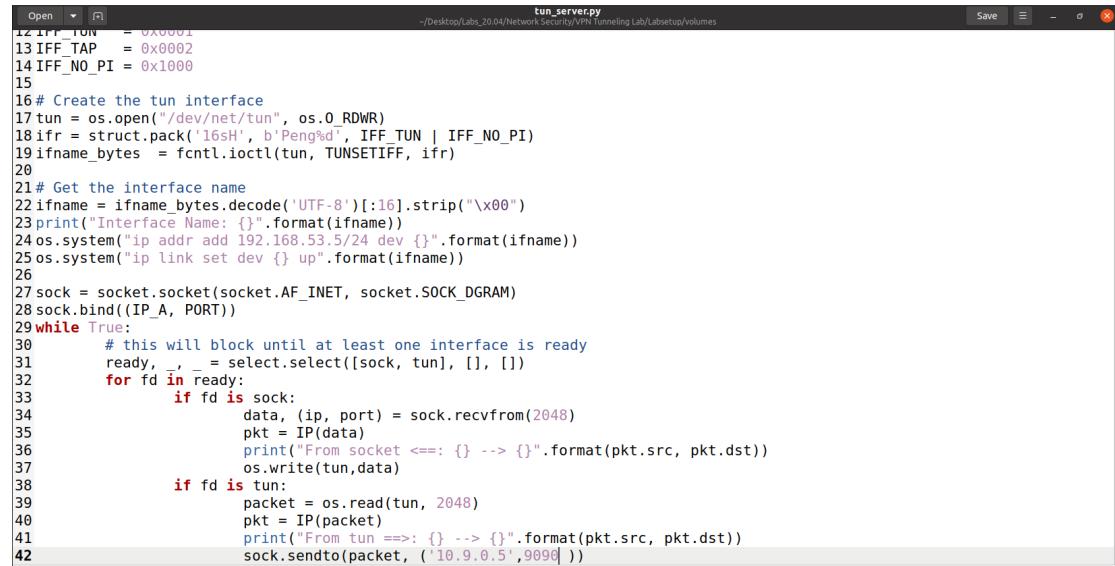
同时通过 wireshark 抓包，可以发现是从 192.168.53.99 发往 192.168.60.5 的，隧道搭建成功。

Time	Source	Destination	Type	Description
2 2021-07-25 06:3...	192.168.60.5	192.168.53.99	ICMP	98 Echo (ping) reply id=0x00c3, seq=1/256, ttl=64 (request in..)
3 2021-07-25 06:3...	192.168.53.99	192.168.60.5	ICMP	98 Echo (ping) request id=0x00c3, seq=2/512, ttl=63 (reply in 4)
4 2021-07-25 06:3...	192.168.60.5	192.168.53.99	ICMP	98 Echo (ping) reply id=0x00c3, seq=2/512, ttl=64 (request in..)
5 2021-07-25 06:3...	192.168.53.99	192.168.60.5	ICMP	98 Echo (ping) request id=0x00c3, seq=3/768, ttl=63 (reply in 6)
6 2021-07-25 06:3...	192.168.60.5	192.168.53.99	ICMP	98 Echo (ping) reply id=0x00c3, seq=3/768, ttl=64 (request in..)
7 2021-07-25 06:3...	192.168.53.99	192.168.60.5	ICMP	98 Echo (ping) request id=0x00c3, seq=4/1024, ttl=63 (reply in ..)
8 2021-07-25 06:3...	192.168.60.5	192.168.53.99	ICMP	98 Echo (ping) reply id=0x00c3, seq=4/1024, ttl=64 (request in..)
9 2021-07-25 06:3...	192.168.53.99	192.168.60.5	ICMP	98 Echo (ping) request id=0x00c3, seq=5/1280, ttl=63 (reply in ..)
10 2021-07-25 06:3...	192.168.60.5	192.168.53.99	ICMP	98 Echo (ping) reply id=0x00c3, seq=5/1280, ttl=64 (request in ..)

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：



```
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     # this will block until at least one interface is ready
31     ready, _, _ = select.select([sock, tun], [], [])
32     for fd in ready:
33         if fd is sock:
34             data, (ip, port) = sock.recvfrom(2048)
35             pkt = IP(data)
36             print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))
37             os.write(tun,data)
38         if fd is tun:
39             packet = os.read(tun, 2048)
40             pkt = IP(packet)
41             print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))
42             sock.sendto(packet, ('10.9.0.5', 9090))
```

然后是 tun_client.py，代码如下：



```
14 IFF_TAP = 0x0002
15 IFF_NO_PI = 0x1000
16
17 # Create the tun interface
18 tun = os.open("/dev/net/tun", os.O_RDWR)
19 ifr = struct.pack('16sH', b'Peng%d', IFF_TUN | IFF_NO_PI)
20 ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
21
22 # Get the interface name
23 ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
24 print("Interface Name: {}".format(ifname))
25 os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
26 os.system("ip link set dev {} up".format(ifname))
27 # Create UDP socket
28 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
29 sock.bind((IP_A, PORT))
30 while True:
31     # this will block until at least one interface is ready
32     ready, _, _ = select.select([sock, tun], [], [])
33     for fd in ready:
34         if fd is sock:
35             data, (ip, port) = sock.recvfrom(2048)
36             pkt = IP(data)
37             print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))
38             os.write(tun,data)
39         if fd is tun:
40             packet = os.read(tun, 2048)
41             pkt = IP(packet)
42             print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))
43             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。

[SEED Labs] Capturing from br-cbc0e2786dd0

No.	Time	Source	Destination	Protocol	Length	Info
73	2021-07-28 19:07:32.105670356	02:42:c0:a8:3c:0b	02:42:c0:a8:3c:05	ARP	42	Who has 192.168.60.5? Tell 192.168.60.11
74	2021-07-28 19:07:32.105727759	02:42:c0:a8:3c:05	02:42:c0:a8:3c:0b	ARP	42	192.168.60.5 is at 02:42:c0:a8:3c:05
75	2021-07-28 19:09:11.436528308	192.168.53.99	192.168.60.5	TELNET	74	Telnet Data ...
76	2021-07-28 19:09:11.436602010	192.168.60.5	192.168.53.99	TCP	66	23 → 35944 [ACK] Seq=1278319524 Ack=2063
77	2021-07-28 19:09:11.438924049	192.168.60.5	192.168.53.99	TELNET	71	Telnet Data ...
78	2021-07-28 19:09:11.445431842	192.168.53.99	192.168.60.5	TELNET	307	Telnet Data ...
79	2021-07-28 19:09:11.445459553	192.168.60.5	192.168.53.99	TCP	66	23 → 35944 [ACK] Seq=1278319529 Ack=2063
80	2021-07-28 19:09:11.451564383	192.168.53.99	192.168.60.5	TCP	66	35944 → 23 [ACK] Seq=2063811531 Ack=1278:
81	2021-07-28 19:09:11.451605539	192.168.60.5	192.168.53.99	TELNET	297	Telnet Data ...
82	2021-07-28 19:09:11.460324284	192.168.53.99	192.168.60.5	TCP	66	35944 → 23 [ACK] Seq=2063811531 Ack=1278:
83	2021-07-28 19:09:16.552743417	02:42:c0:a8:3c:05	02:42:c0:a8:3c:0b	ARP	42	Who has 192.168.60.11? Tell 192.168.60.5
84	2021-07-28 19:09:16.552748567	02:42:c0:a8:3c:0b	02:42:c0:a8:3c:05	ARP	42	Who has 192.168.60.5? Tell 192.168.60.11
85	2021-07-28 19:09:16.552805276	02:42:c0:a8:3c:0b	02:42:c0:a8:3c:05	ARP	42	192.168.60.11 is at 02:42:c0:a8:3c:0b
86	2021-07-28 19:09:16.552808800	02:42:c0:a8:3c:05	02:42:c0:a8:3c:0b	ARP	42	192.168.60.5 is at 02:42:c0:a8:3c:05

```

> Frame 78: 307 bytes on wire (2456 bits), 307 bytes captured (2456 bits) on interface br-cbc0e2786dd0, id 0
> Ethernet II, Src: 02:42:c0:a8:3c:0b (02:42:c0:a8:3c:0b), Dst: 02:42:c0:a8:3c:05 (02:42:c0:a8:3c:05)
> Internet Protocol Version 4, Src: 192.168.53.99, Dst: 192.168.60.5
> Transmission Control Protocol, Src Port: 35944, Dst Port: 23, Seq: 2063811290, Ack: 1278319524, Len: 241
> Telnet

```

00c0	61	73	64	61	73	64	73	64	61	73	73	61	64	asdadasd asdassad
00d0	61	73	64	61	73	64	73	61	64	73	64	61	73	asdadsa dsddasds
00e0	64	73	64	61	73	64	73	61	64	73	61	73	73	sdasdads dasdasas
00f0	64	61	73	64	73	61	64	61	73	64	61	73	64	dasdadsa sdasdad
0100	73	64	61	73	64	73	64	73	61	64	73	61	64	73
0110	64	73	64	61	64	73	64	61	73	64	61	73	61	64
0120	73	64	73	64	61	64	73	73	64	61	73	64	61	73
0130	61	73	64											asd

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。