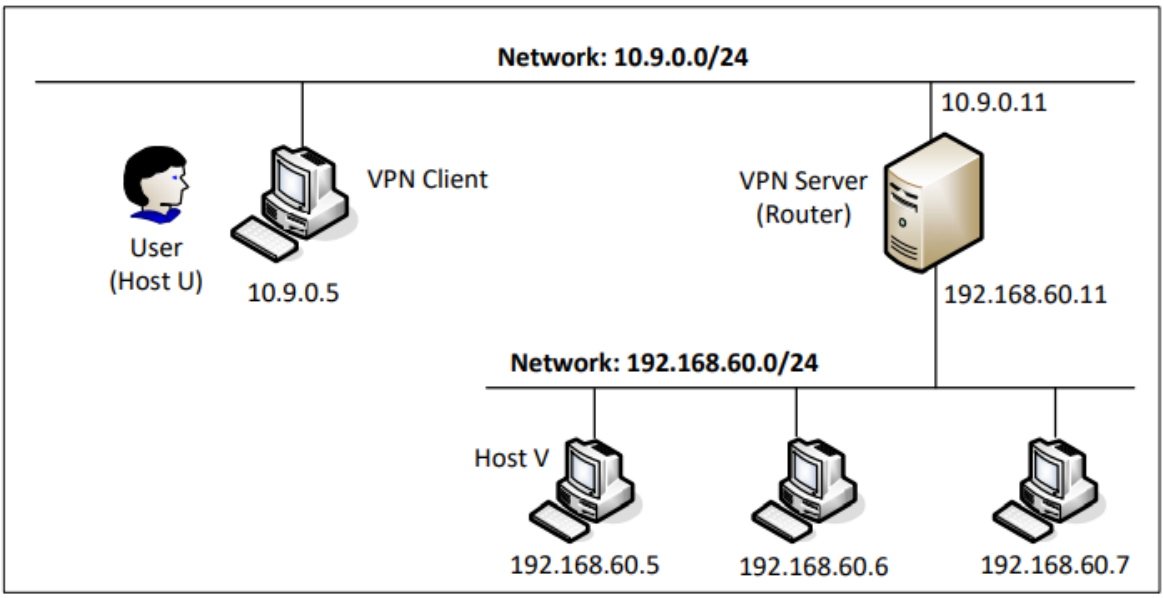
Lab7: VPN Lab

57118228 孙志刚



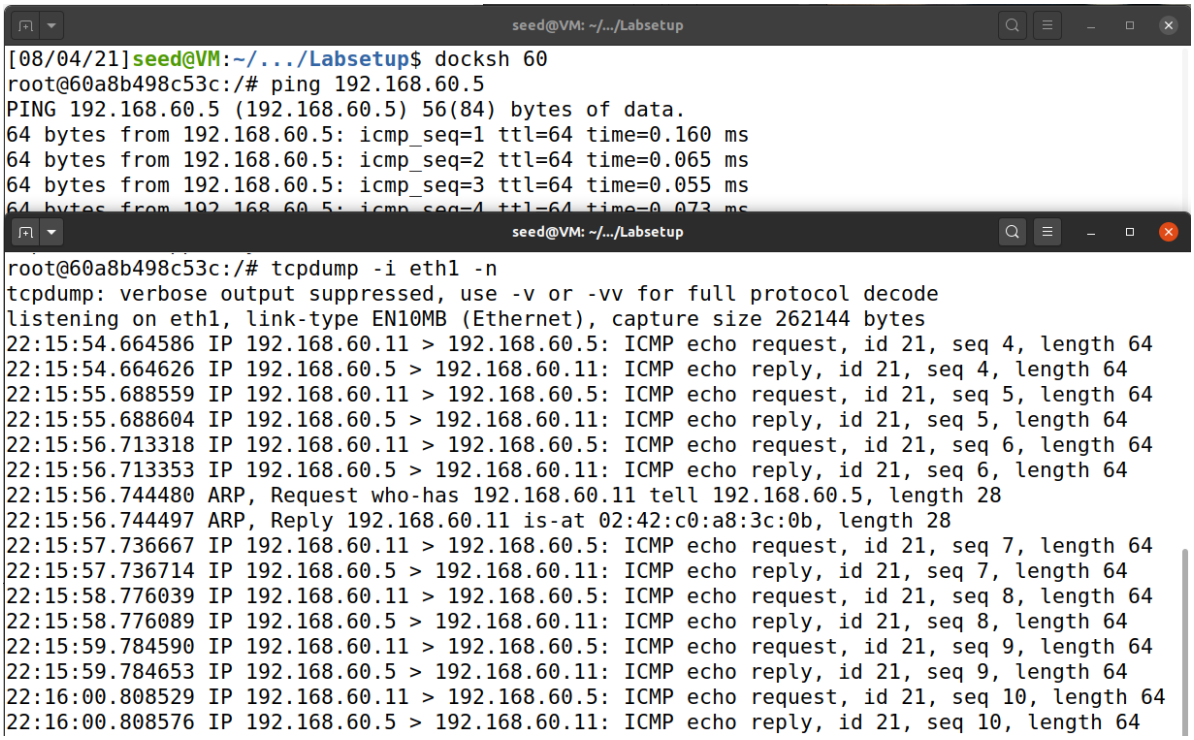
Task 1: Network Setup

验证主机 U 可以与 VPN Server 通信，Host U ping VPN server时，VPN server可从eth 0利用tcpdump命令抓取数据包。

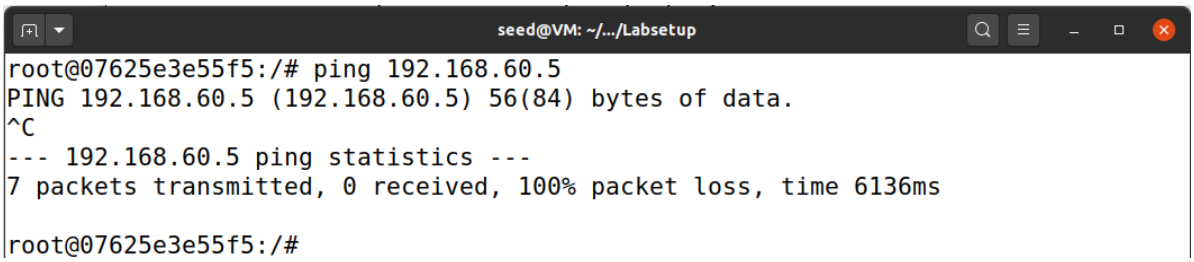
```
seed@VM: ~/.../Labsetup
[08/04/21]seed@VM:~/.../Labsetup$ docksh 07
root@07625e3e55f5:/# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.099 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.064 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.126 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.073 ms
64 bytes from 10.9.0.11: icmp_seq=5 ttl=64 time=0.085 ms
64 bytes from 10.9.0.11: icmp_seq=6 ttl=64 time=0.073 ms
```

```
seed@VM: ~/.../Labsetup
[08/04/21]seed@VM:~/.../Labsetup$ docksh 60
root@60a8b498c53c:/# 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
22:11:05.964533 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 20, seq 3, length 64
22:11:05.964566 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 20, seq 3, length 64
22:11:06.993201 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 20, seq 4, length 64
22:11:06.993223 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 20, seq 4, length 64
22:11:08.009701 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 20, seq 5, length 64
22:11:08.009726 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 20, seq 5, length 64
22:11:09.002548 ARP, Request who-has 10.9.0.5 tell 10.9.0.11, length 28
22:11:09.002669 ARP, Request who-has 10.9.0.11 tell 10.9.0.5, length 28
22:11:09.002682 ARP, Reply 10.9.0.11 is-at 02:42:0a:09:00:0b, length 28
22:11:09.002684 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, length 28
22:11:09.054866 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 20, seq 6, length 64
22:11:09.054889 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 20, seq 6, length 64
22:11:10.117028 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 20, seq 7, length 64
22:11:10.117075 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 20, seq 7, length 64
22:11:11.145621 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 20, seq 8, length 64
```

验证主机 V 192.168.60.5 可以与 VPN Server 通信，同时在VPN服务器上利用tcpdump命令抓取数据包，得到结果如下:



验证主机 U 不可与主机 V 通信。



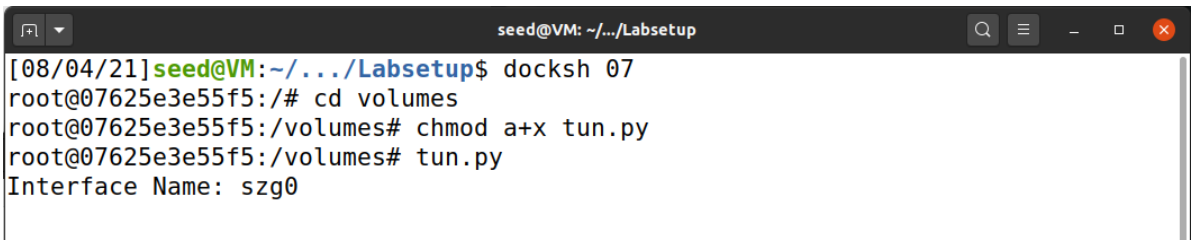
Task 2: Create and Configure TUN Interface

Task 2.a: Name of the Interface

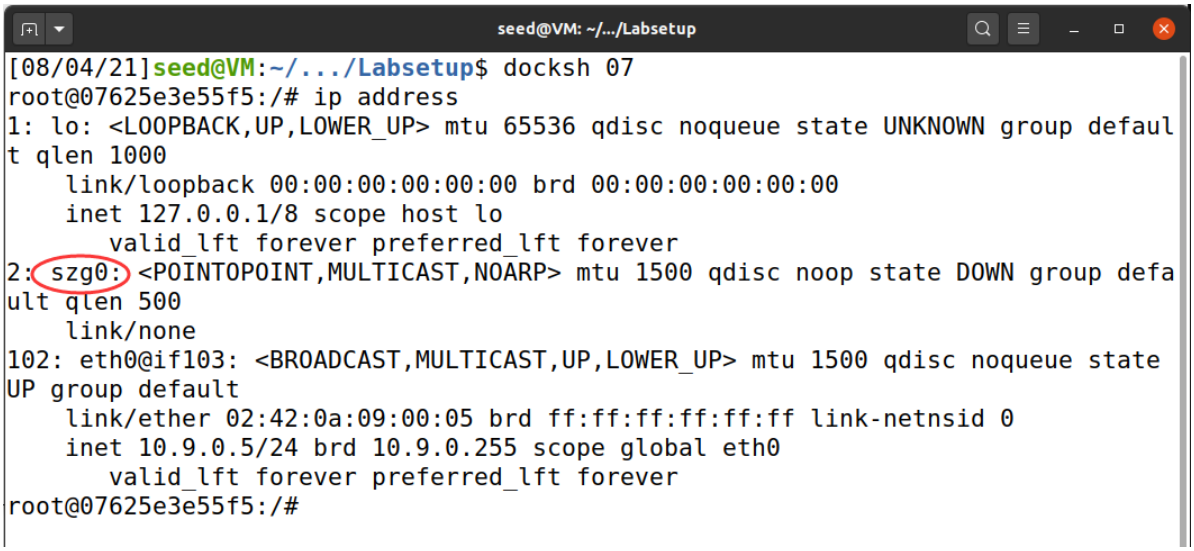
在代码此处将 tun 修改成自己名字简拼 szg 。

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

在主机 U(10.9.0.5) 上运行 chmod a+x tun.py 和 tun.py 可以观察到修改接口成功。



然后在主机 U(10.9.0.5) 上运行 ip address 查看所有接口，可发现我们修改的 tun 接口，命名为 szg0 。

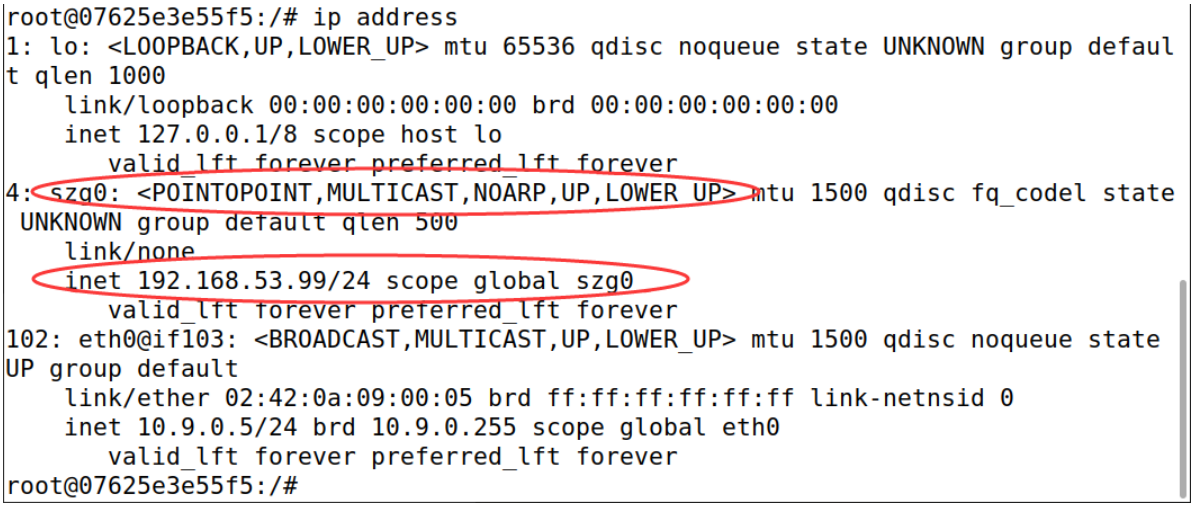


Task 2.b: Set up the TUN Interface

在 tun.py 文件中添加以下两行代码，给端口自动分配地址：

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

再次运行，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

```
#!/usr/bin/env python3
import fcntl
import struct
```

```python
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'szg%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))

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

在主机 U上ping主机192.168.53.1，得到结果如下，可知无法连接:

```
root@07625e3e55f5:/# 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 ---
2 packets transmitted, 0 received, 100% packet loss, time 1012ms

root@07625e3e55f5:/#
```

利用root权限运行该程序后，得到结果如下，可知icmp请求报文成功发送，但IP地址为192.168.53.1的 主机不存在，导致ping无法连接。

```
root@07625e3e55f5:/# ping 192.168.53.1
PING 192.168.53.1 (192.168.53.1) 56(84) bytes of data.
```

```
root@07625e3e55f5:/volumes# tun.py
Interface Name: szg0
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
```

在主机 U上ping主机V，无法连接，这是因为相应报文的目的IP不在TUN接口的网段内。

```
seed@VM: ~/.../volumes
[08/04/21]seed@VM:~/.../volumes$ docksh 07
root@07625e3e55f5:/# cd volumes
root@07625e3e55f5:/volumes# tun.py
Interface Name: szg0
```

```
seed@VM: ~/.../volumes
root@07625e3e55f5:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
```

Task 2.d: Write to the TUN Interface

修改while循环的内容，当host U ping 192.168.53.11网段的地址时，可以收到答复

```python
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))
```

```
root@07625e3e55f5:/# ping 192.168.53.11
PING 192.168.53.11 (192.168.53.11) 56(84) bytes of data.
64 bytes from 192.168.53.11: icmp_seq=4 ttl=64 time=2.39 ms
64 bytes from 192.168.53.11: icmp_seq=5 ttl=64 time=2.43 ms
64 bytes from 192.168.53.11: icmp_seq=6 ttl=64 time=4.31 ms
64 bytes from 192.168.53.11: icmp_seq=7 ttl=64 time=2.63 ms
64 bytes from 192.168.53.11: icmp_seq=8 ttl=64 time=3.28 ms
64 bytes from 192.168.53.11: icmp_seq=9 ttl=64 time=2.30 ms
```

```
seed@VM: ~/.../volumes
root@07625e3e55f5:/volumes# tun.py
Interface Name: szg0
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
```

倘若向接口随意写入字符串

```
...
    os.write(tun,bytes("virtualbox1234"))
```

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。

```
root@07625e3e55f5:/volumes# tun.py
Interface Name: szg0
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
Traceback (most recent call last):
  File "./tun.py", line 41, in <module>
    os.write(tun,bytes("virtualbox1234"))
TypeError: string argument without an encoding
root@07625e3e55f5:/volumes#
```

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

tun_client.py的代码如下。

```
#!/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'szg%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))
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)

while True:
   packet = os.read(tun, 2048)
   if packet:
```

```
        sock.sendto(packet, ("10.9.0.11", 9090))
```

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。

```
seed@VM: ~/.../volumes
[08/04/21]seed@VM:~/.../volumes$ docksh 60
root@60a8b498c53c:/# cd volumes
root@60a8b498c53c:/volumes#  chmod a+x tun_server.py
root@60a8b498c53c:/volumes# tun_server.py
10.9.0.5:38380 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.2
10.9.0.5:38380 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.2
10.9.0.5:38380 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.2
10.9.0.5:38380 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.2
10.9.0.5:38380 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.2
10.9.0.5:38380 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.2
```

192.168.53.0/24的路由经过接口 szg0 ，成功通过隧道发送 udp 报文

在主机U上ping主机V，可知无法连接,因为 192.168.60.0/24 的路由不经过接口 szg0 。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 szg0
```

```
root@07625e3e55f5:/# ip route add 192.168.60.0/24 dev szg0
root@07625e3e55f5:/# ip route
default via 10.9.0.1 dev eth0
10.9.0.0/24 dev eth0 proto kernel scope link src 10.9.0.5
192.168.53.0/24 dev szg0 proto kernel scope link src 192.168.53.99
192.168.60.0/24 dev szg0 scope link
root@07625e3e55f5:/#
```

重复上述实验过程，在VPN服务器上利用root权限运行tun_server.py程序，得到

```
root@60a8b498c53c:/volumes# tun_server.py
10.9.0.5:40340 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:40340 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:40340 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:40340 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:40340 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:40340 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:40340 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
```

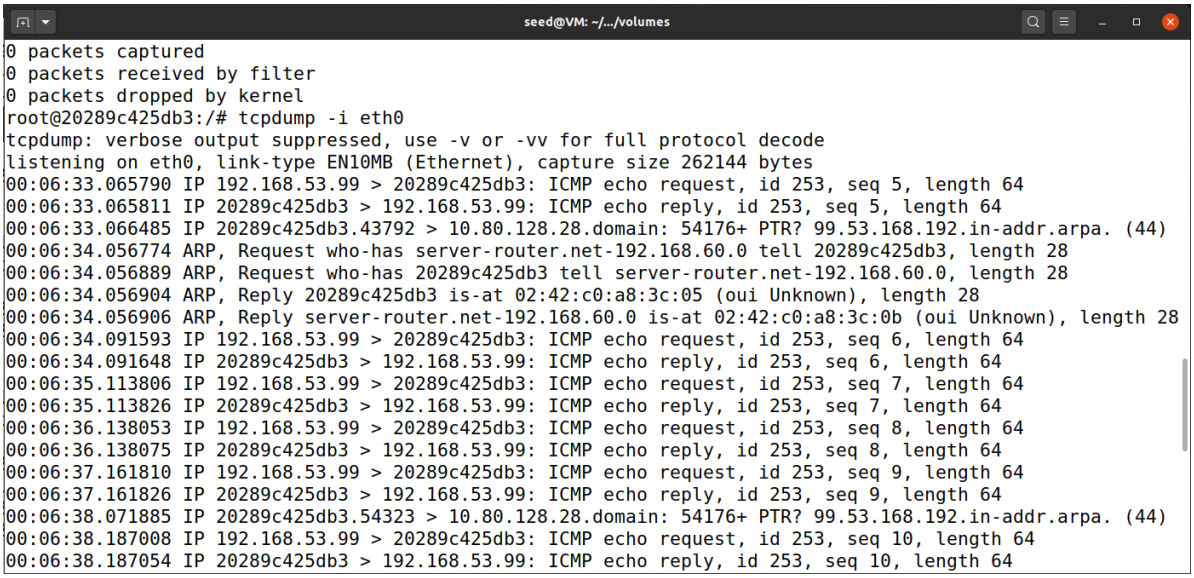
Task 4: Set Up the VPN Server

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'szg%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 192.168.53.11/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

print("Interface Name: {}".format(ifname))
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)

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 Task3 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

修改后的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)
```

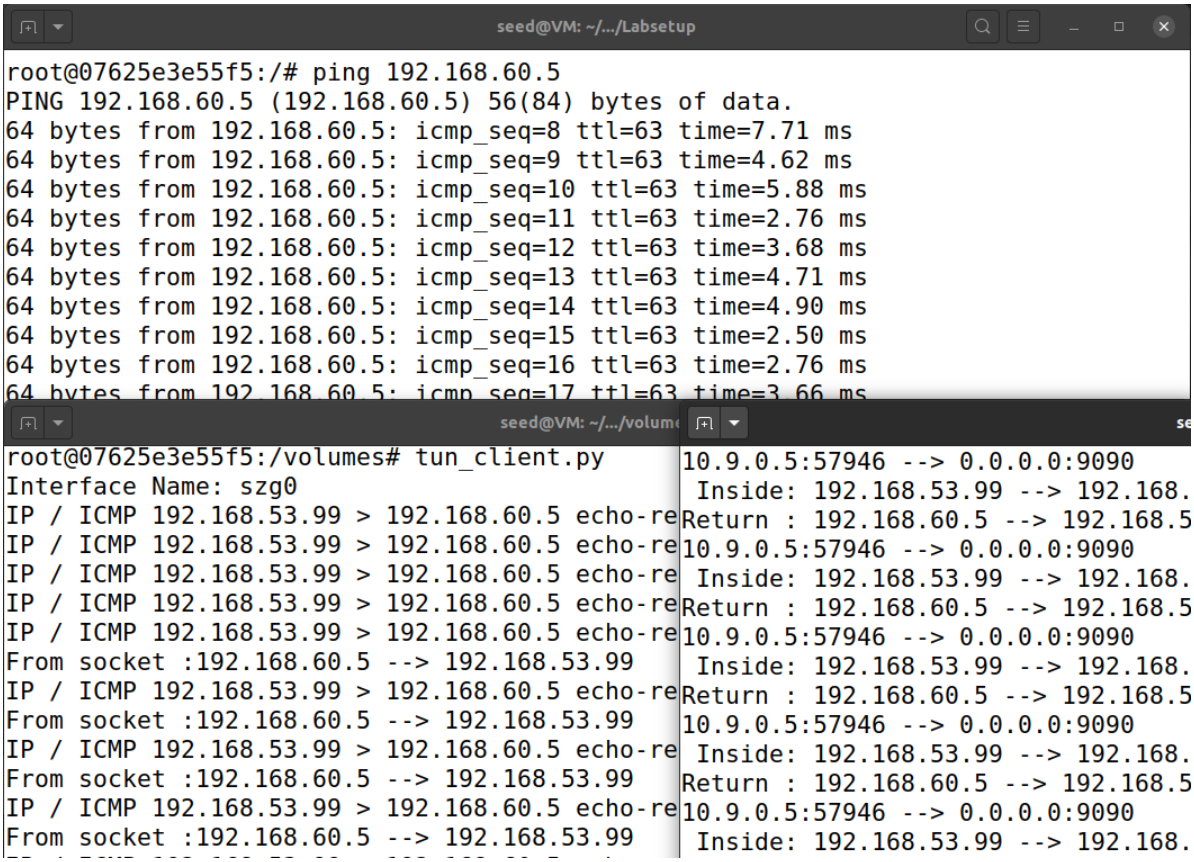
```
ifr = struct.pack('16sH', b'szg%d', IFF_TUN | IFF_NO_PI)
ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)

# 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)

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, ("10.9.0.11", 9090))
```

修改后的tun_server.py如下

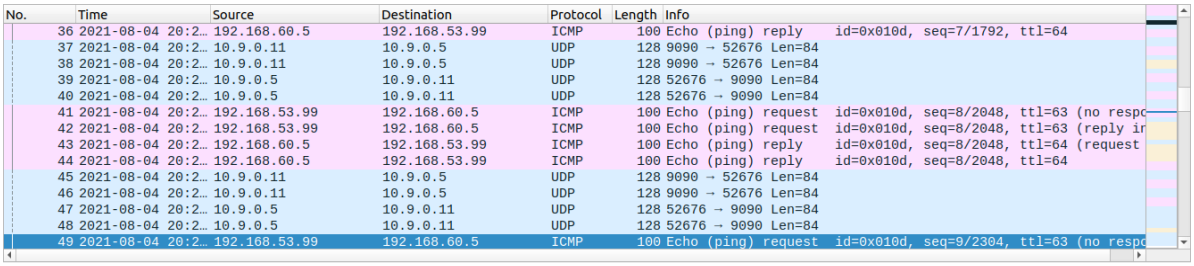
```
#!/usr/bin/env python3

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'szg%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))
```

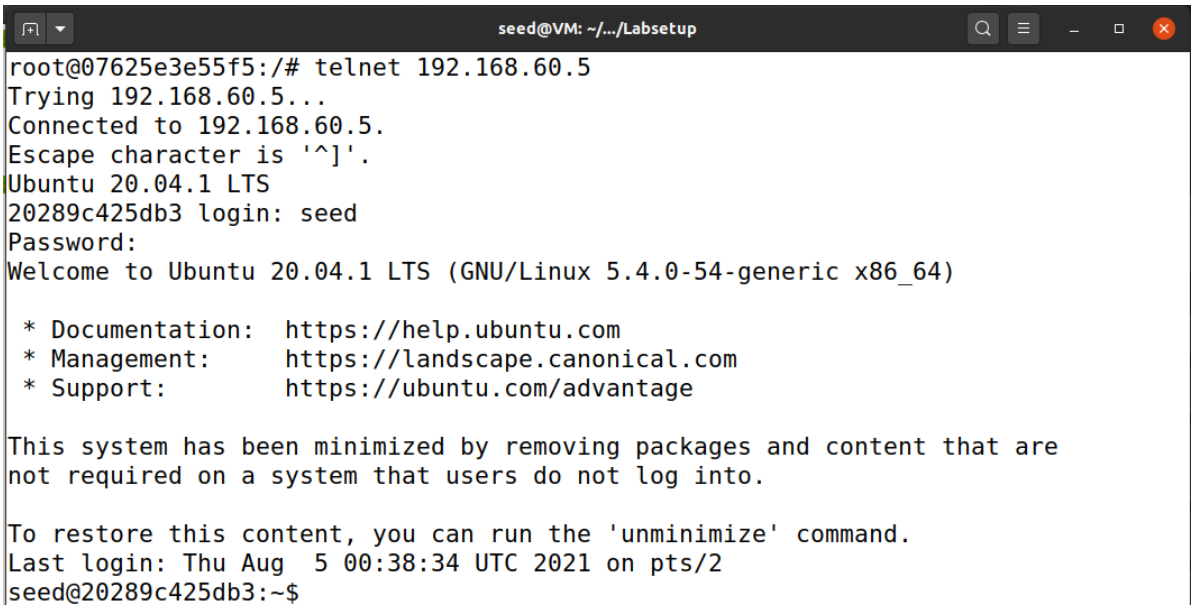
在 Host U上ping Host V，可以看到顺利成功了。



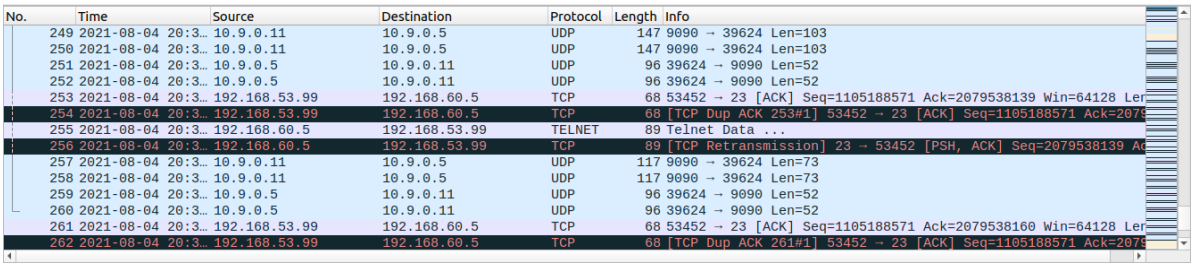
通过Wireshrk可以看见更为清晰的VPN tunneling过程，先是10.9.0.5发送给10.9.0.11，然后VPN变为 192.168.53.99发往192.168.60.5，然后再原路径返回。



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



No.	Time	Source	Destination	Protocol	Length	Info
249	2021-08-04 20:3…	10.9.0.11	10.9.0.5	UDP	147	9090 → 39624 Len=103
250	2021-08-04 20:3…	10.9.0.11	10.9.0.5	UDP	147	9090 → 39624 Len=103
251	2021-08-04 20:3…	10.9.0.5	10.9.0.11	UDP	96	39624 → 9090 Len=52
252	2021-08-04 20:3…	10.9.0.5	10.9.0.11	UDP	96	39624 → 9090 Len=52
253	2021-08-04 20:3…	192.168.53.99	192.168.60.5	TCP	68	53452 → 23 [ACK] Seq=1105188571 Ack=2079538139 Win=64128 Len
254	2021-08-04 20:3…	192.168.53.99	192.168.60.5	TCP	68	[TCP Dup ACK 253#1] 53452 → 23 [ACK] Seq=1105188571 Ack=2079
255	2021-08-04 20:3…	192.168.60.5	192.168.53.99	TELNET	89	Telnet Data ...
256	2021-08-04 20:3…	192.168.60.5	192.168.53.99	TCP	89	[TCP Retransmission] 23 → 53452 [PSH, ACK] Seq=2079538139 Ac
257	2021-08-04 20:3…	10.9.0.11	10.9.0.5	UDP	117	9090 → 39624 Len=73
258	2021-08-04 20:3…	10.9.0.11	10.9.0.5	UDP	117	9090 → 39624 Len=73
259	2021-08-04 20:3…	10.9.0.5	10.9.0.11	UDP	96	39624 → 9090 Len=52
260	2021-08-04 20:3…	10.9.0.5	10.9.0.11	UDP	96	39624 → 9090 Len=52
261	2021-08-04 20:3…	192.168.53.99	192.168.60.5	TCP	68	53452 → 23 [ACK] Seq=1105188571 Ack=2079538160 Win=64128 Len
262	2021-08-04 20:3…	192.168.53.99	192.168.60.5	TCP	68	[TCP Dup ACK 261#1] 53452 → 23 [ACK] Seq=1105188571 Ack=2079

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。

```
seed@20289c425db3:~$ sssss
-bash: sssss: command not found
seed@20289c425db3:~$ whoami
seed
seed@20289c425db3:~$
```

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。

```
seed@20289c425db3:~$ sssss
-bash: sssss: command not found
seed@20289c425db3:~$ whoami
seed
seed@20289c425db3:~$ dfsf sfgsrf
```

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。