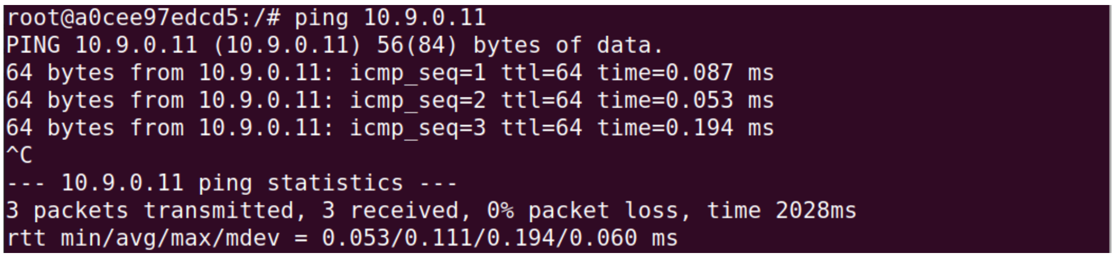
The report of lab 7

57118117 谌雨阳

Task 1: Network Setup

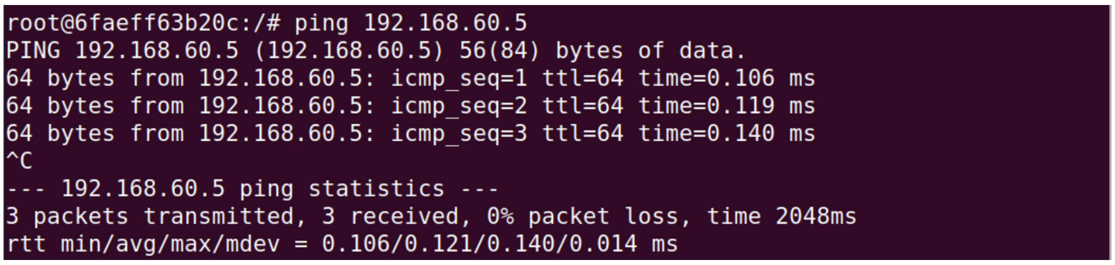
Testing Result：

（1）Host U can communicate with VPN Server.



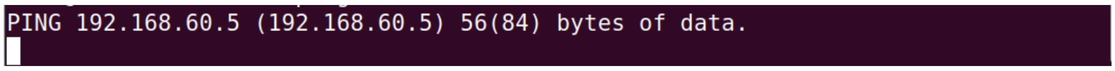
```
root@a0cee97edcd5:/# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.087 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.053 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.194 ms
^C
--- 10.9.0.11 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2028ms
rtt min/avg/max/mdev = 0.053/0.111/0.194/0.060 ms
```

（2）VPN Server can communicate with Host V.



```
root@6faeff63b20c:/# 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=0.106 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=64 time=0.119 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=64 time=0.140 ms
^C
--- 192.168.60.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2048ms
rtt min/avg/max/mdev = 0.106/0.121/0.140/0.014 ms
```

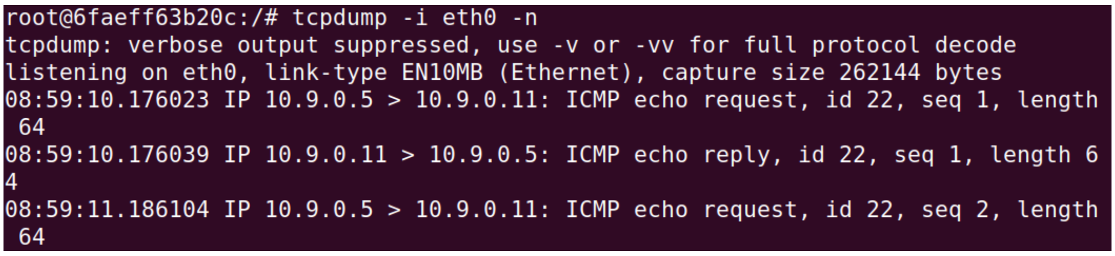
（3）Host U should not be able to communicate with Host V.



```
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
```

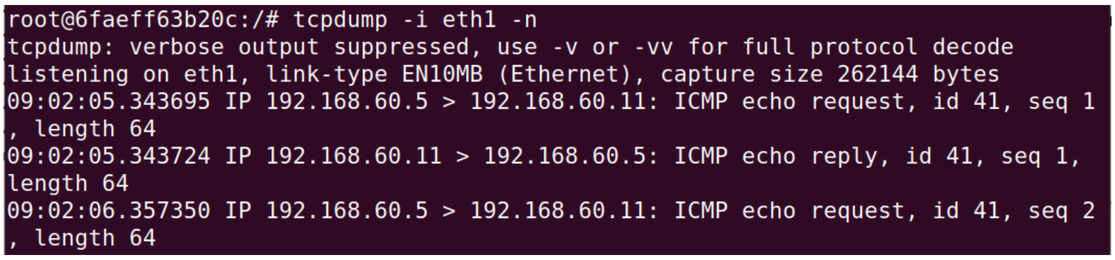
（4）Run tcpdump on the router, and sniff the traffic on each of the network. Show that you can capture packets.

eth0：



```
root@6faeff63b20c:/# 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
08:59:10.176023 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 22, seq 1, length
 64
08:59:10.176039 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 22, seq 1, length 6
4
08:59:11.186104 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 22, seq 2, length
 64
```

eth1：



```
root@6faeff63b20c:/# 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
09:02:05.343695 IP 192.168.60.5 > 192.168.60.11: ICMP echo request, id 41, seq 1
, length 64
09:02:05.343724 IP 192.168.60.11 > 192.168.60.5: ICMP echo reply, id 41, seq 1,
length 64
09:02:06.357350 IP 192.168.60.5 > 192.168.60.11: ICMP echo request, id 41, seq 2
, length 64
```

测试表明 lab 环境配置没有问题。

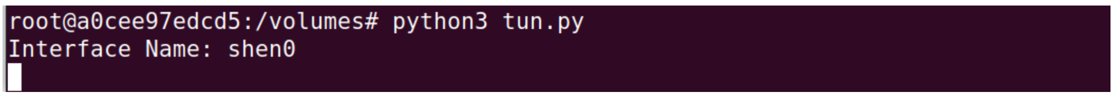
Task 2: Create and Configure TUN Interface

Task 2.a: Name of the Interface

更改代码如下，将名字前缀更改为 shen：

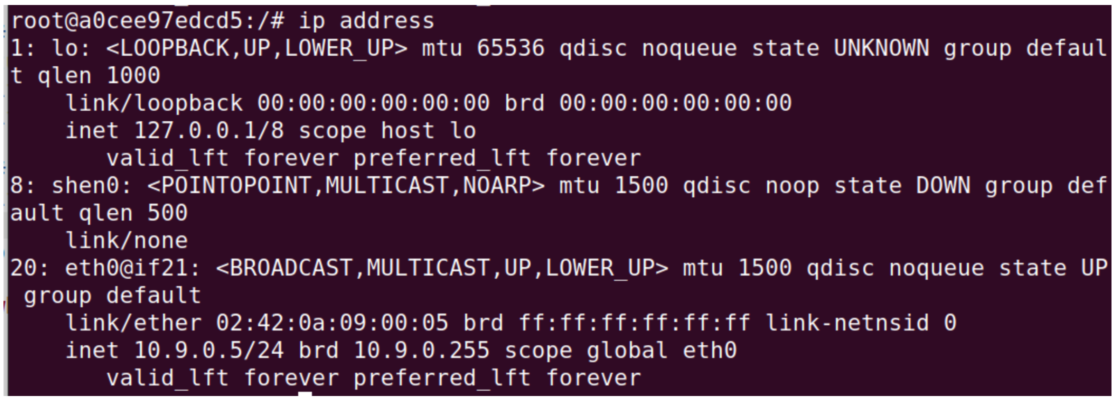
```
# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'shen%d', IFF_TUN | IFF_NO_PI)
ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)
```

在 Host U（10.9.0.5）上运行已有的 tun.py 程序：



```
root@a0cee97edcd5:/volumes# python3 tun.py
Interface Name: shen0
```

新开一个 Host U 的 shell，并通过 ip address 查看网络接口：



```
root@a0cee97edcd5:/# 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
8: shen0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group def
ault qlen 500
    link/none
20: eth0@if21: <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
```

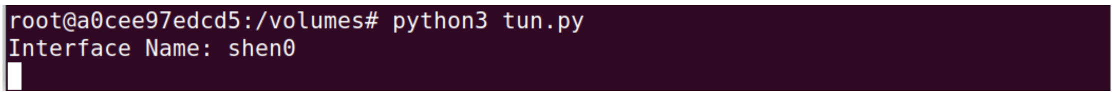
能找到我们通过 tun.py 程序创建的 tun 接口，但是目前还没有任何内容，不可使用。

Task 2.b: Set up the TUN Interface

在 tun.py 中添加如下两行代码：

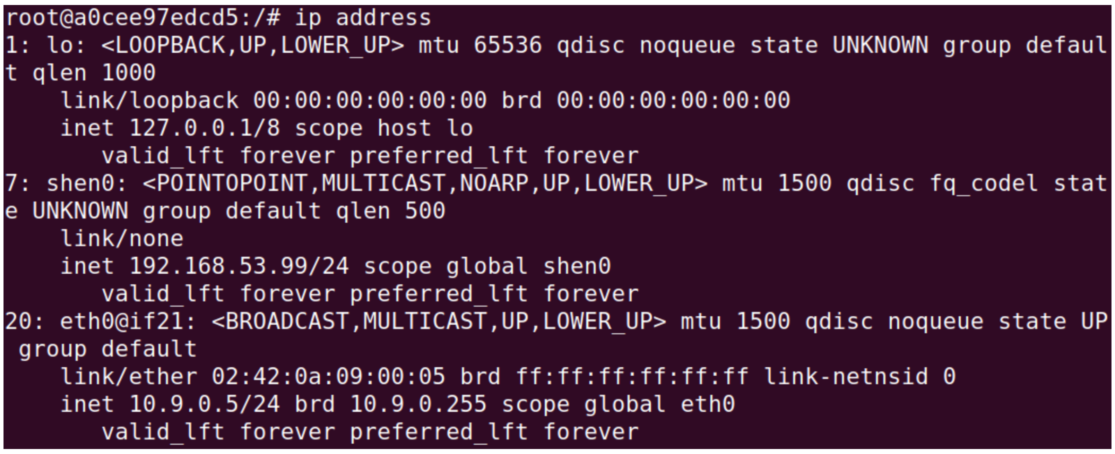
```
os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
```

再在 Host U（10.9.0.5）上运行修改后的 tun.py 程序：



```
root@a0cee97edcd5:/volumes# python3 tun.py
Interface Name: shen0
```

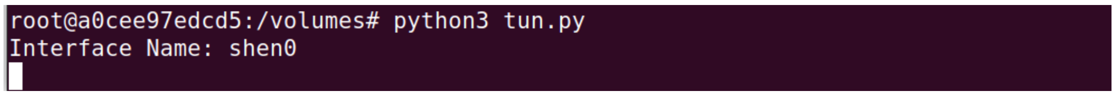
新开一个 Host U 的 shell，并通过 ip address 查看网络接口：



```
root@a0cee97edcd5:/# 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
7: shen0: <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 shen0
       valid_lft forever preferred_lft forever
20: eth0@if21: <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
```

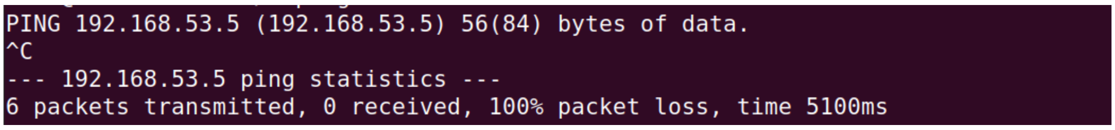
Task 2.c: Read from the TUN Interface

先在 Host U（10.9.0.5）上运行修改后的 tun.py 程序：



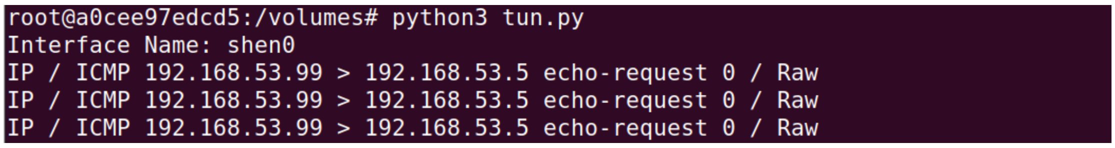
```
root@a0cee97edcd5:/volumes# python3 tun.py
Interface Name: shen0
```

在 Host U 上尝试 ping 192.168.53.0/24 网段上的主机，无法 ping 通：



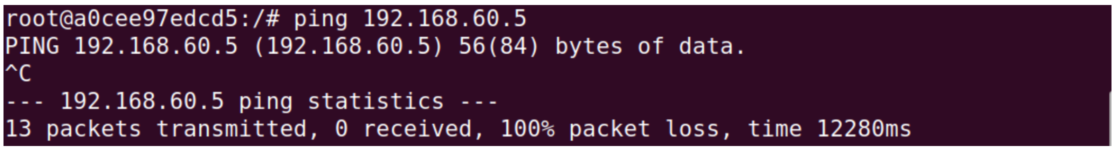
```
PING 192.168.53.5 (192.168.53.5) 56(84) bytes of data.
^C
--- 192.168.53.5 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5100ms
```

tun.py 的打印内容如下，表明程序读取到了 IP 数据包，并打印出了 summary，因为前往该网段的数据包会到达 tun 接口：



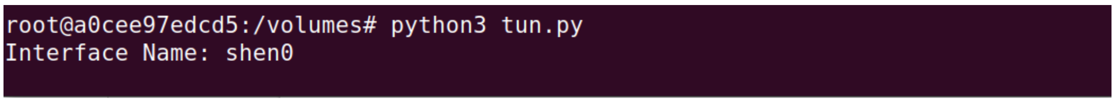
```
root@a0cee97edcd5:/volumes# python3 tun.py
Interface Name: shen0
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
```

再在 Host U 上尝试 ping 192.168.60.0/24 网段上的主机，无法 ping 通，因为前往该网段的数据包不到达 tun 接口：



```
root@a0cee97edcd5:/# 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 ---
13 packets transmitted, 0 received, 100% packet loss, time 12280ms
```

tun.py 的打印内容如下，表明程序没有读取到 IP 数据包：



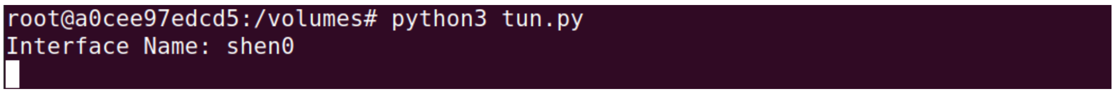
```
root@a0cee97edcd5:/volumes# python3 tun.py
Interface Name: shen0
```

Task 2.d: Write to the TUN Interface

修改 while 循环部分代码如下：

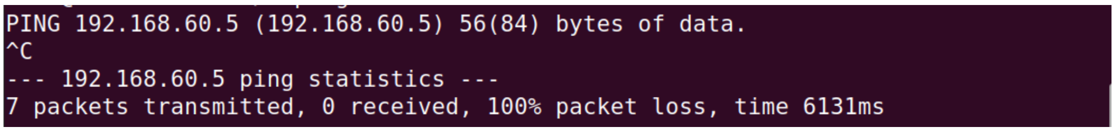
```python
while True:
    # Get a packet from the tun interface
    packet = os.read(tun, 2048)
    if packet:
        pkt = IP(packet)
        print(pkt.summary())

        if ICMP in pkt:
            # Send out a spoof packet using the tun interface
            newip = IP(src=pkt[IP].dst, dst=pkt[IP].src,ihl=pkt[IP].ihl,ttl=99)
            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))
```

在 Host U（10.9.0.5）上运行修改后的 tun.py 程序：



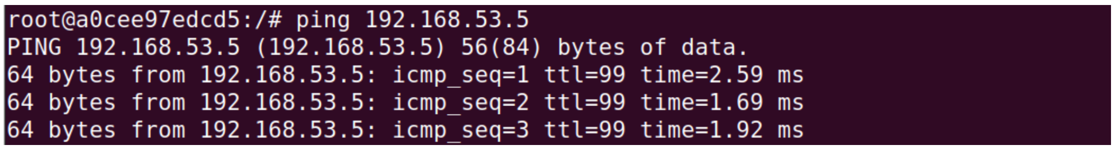
```
root@a0cee97edcd5:/volumes# python3 tun.py
Interface Name: shen0
```

在 Host U 上尝试 ping 192.168.60.0/24 网段上的主机，无法 ping 通：



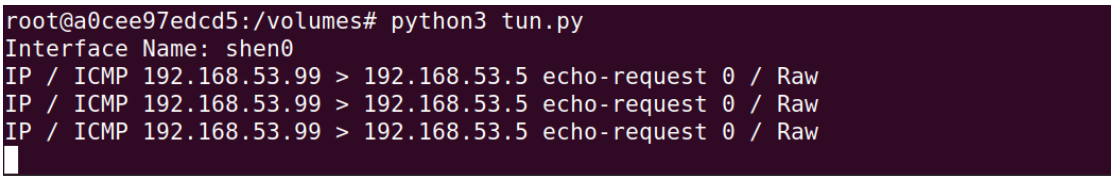
```
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
7 packets transmitted, 0 received, 100% packet loss, time 6131ms
```

在 Host U 上尝试 ping 192.168.53.0/24 网段上的主机，可以 ping 通，可以从 ttl=99 看出返回的是程序伪造的数据包：



```
root@a0cee97edcd5:/# ping 192.168.53.5
PING 192.168.53.5 (192.168.53.5) 56(84) bytes of data.
64 bytes from 192.168.53.5: icmp_seq=1 ttl=99 time=2.59 ms
64 bytes from 192.168.53.5: icmp_seq=2 ttl=99 time=1.69 ms
64 bytes from 192.168.53.5: icmp_seq=3 ttl=99 time=1.92 ms
```

tun.py 的打印内容如下

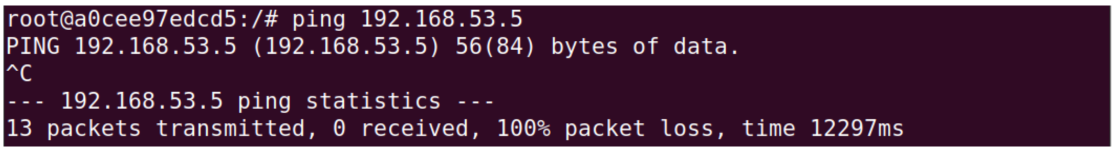


```
root@a0cee97edcd5:/volumes# python3 tun.py
Interface Name: shen0
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
```

将上面代码中的 os.write()函数参数做如下修改，改为写入人为制造的字符串：

```python
a = "Hello World!"
os.write(tun, bytes(a,'UTF-8'))
```

在 Host U 上尝试 ping 192.168.53.0/24 网段上的主机，无法 ping 通了：



```
root@a0cee97edcd5:/# 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 ---
13 packets transmitted, 0 received, 100% packet loss, time 12297ms
```

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

Testing Result:

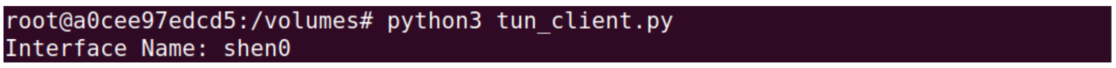
tun_client.py 代码如下：

```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'shen%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)
SERVER_IP = "10.9.0.11"
SERVER_PORT = 9090

while True:
    # Get a packet from the tun interface
    packet = os.read(tun, 2048)
    if packet:
        pkt = IP(packet)
        print(pkt.summary())
        # Send the packet via the tunnel
        sock.sendto(packet, (SERVER_IP, SERVER_PORT))

```

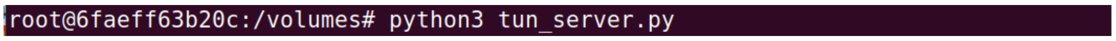
tun_server.py 代码如下：

```python
#!/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 Host U 上运行 tun_client.py，VPN Server 上运行 tun_server.py：

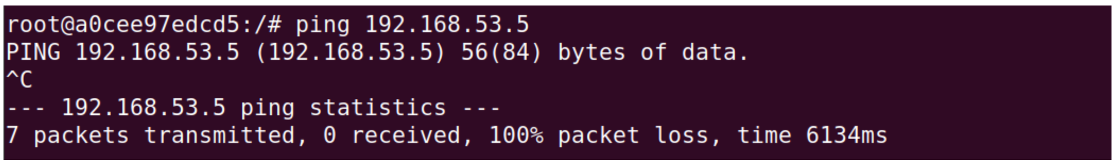


```
root@a0cee97edcd5:/volumes# python3 tun_client.py
Interface Name: shen0
```



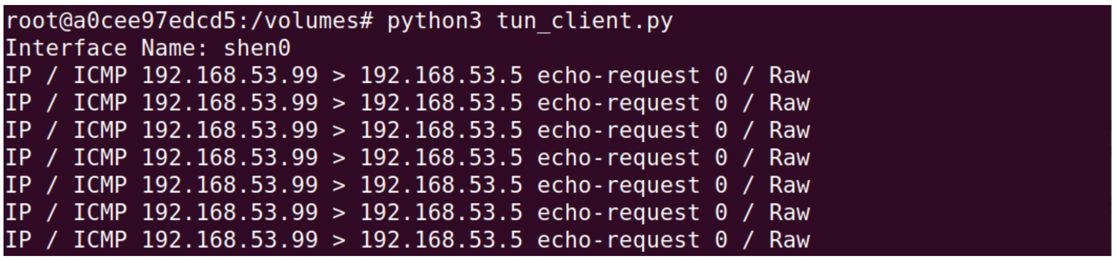
```
root@6faeff63b20c:/volumes# python3 tun_server.py
```

在 Host U 上尝试 ping 192.168.53.0/24 网段上的主机，ping 不通：



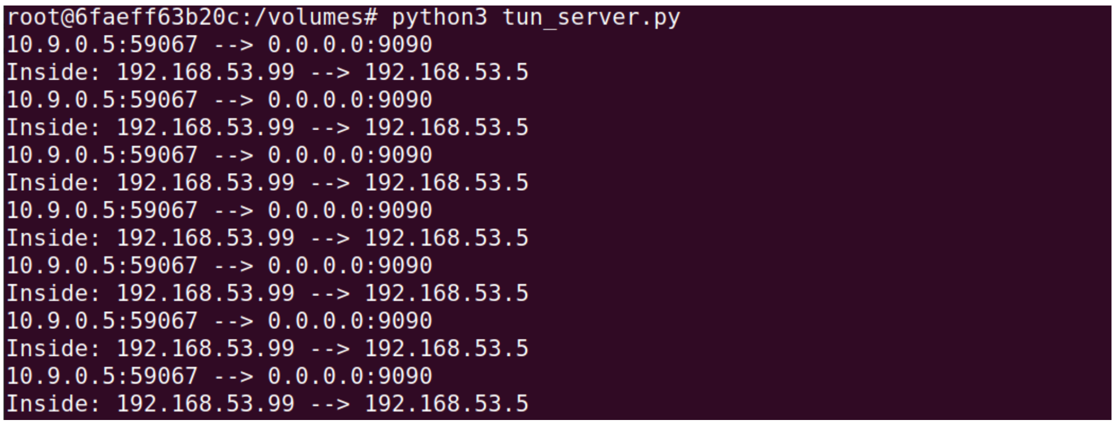
```
root@a0cee97edcd5:/# ping 192.168.53.5
PING 192.168.53.5 (192.168.53.5) 56(84) bytes of data.
^C
--- 192.168.53.5 ping statistics ---
7 packets transmitted, 0 received, 100% packet loss, time 6134ms
```

tun_client.py 打印内容如下：



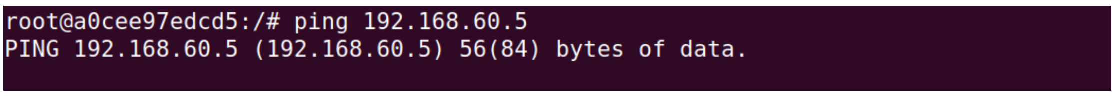
```
root@a0cee97edcd5:/volumes# python3 tun_client.py
Interface Name: shen0
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 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
```

tun_server.py 打印内容如下：

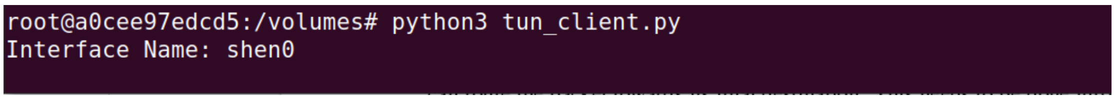


```
root@6faeff63b20c:/volumes# python3 tun_server.py
10.9.0.5:59067 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:59067 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:59067 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:59067 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:59067 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:59067 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:59067 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.53.5
```

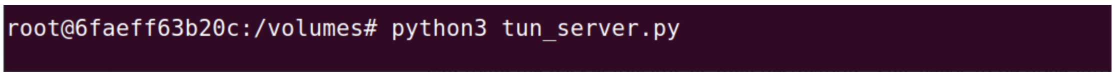
若 ping 192.168.60.0/24 网段，则 ping 不通且两个程序都没有打印出任何内容：



```
root@a0cee97edcd5:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
```



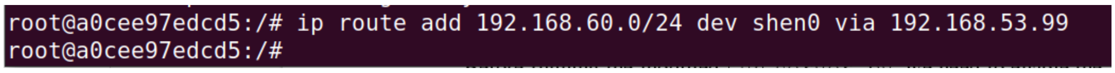
```
root@a0cee97edcd5:/volumes# python3 tun_client.py
Interface Name: shen0
```



```
root@6faeff63b20c:/volumes# python3 tun_server.py
```

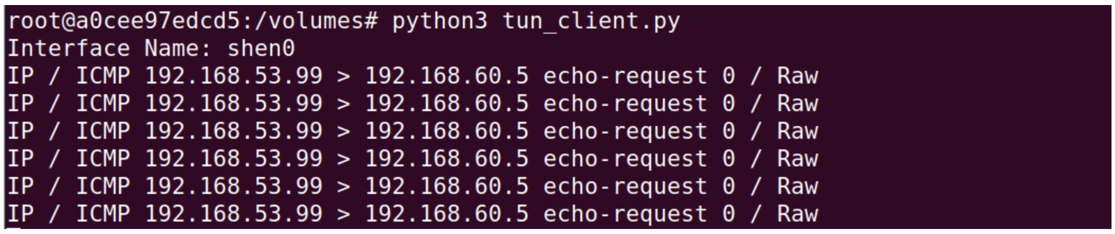
这是因为通往 60 网段的包并不会通过 tunnel，需要修改路由来解决这个问题。

如图，添加了一条去往 60 网段的路由：



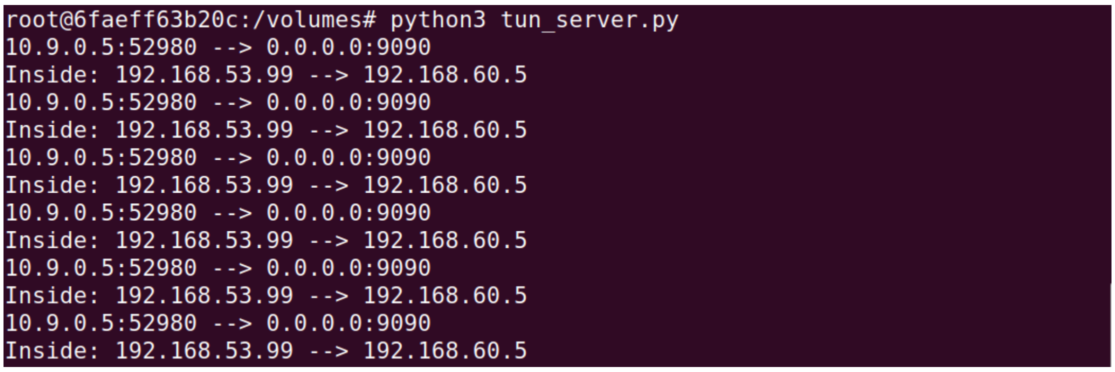
```
root@a0cee97edcd5:/# ip route add 192.168.60.0/24 dev shen0 via 192.168.53.99
root@a0cee97edcd5:/#
```

tun_client.py 打印内容如下：



```
root@a0cee97edcd5:/volumes# python3 tun_client.py
Interface Name: shen0
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
```

tun_server.py 打印内容如下：



```
root@6faeff63b20c:/volumes# python3 tun_server.py
10.9.0.5:52980 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:52980 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:52980 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:52980 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:52980 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:52980 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
```

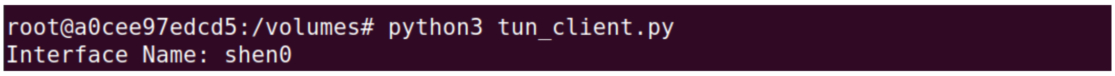
Task 4: Set Up the VPN Server

Testing Result:

修改 tun_server.py 代码如下：

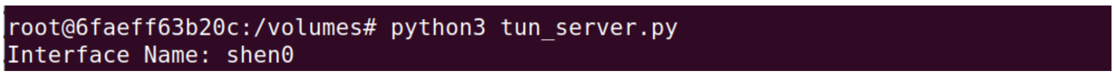
```python
#!/usr/bin/env python3
from scapy.all import *
import fcntl
import struct
import os
import time

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'shen%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 192.168.53.98/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
print("Interface Name: {}".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)
    print("write")
```

在 Host U 上运行 tun_client.py：

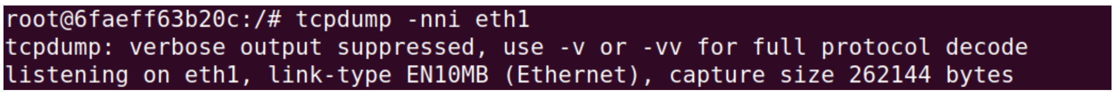


```
root@a0cee97edcd5:/volumes# python3 tun_client.py
Interface Name: shen0
```

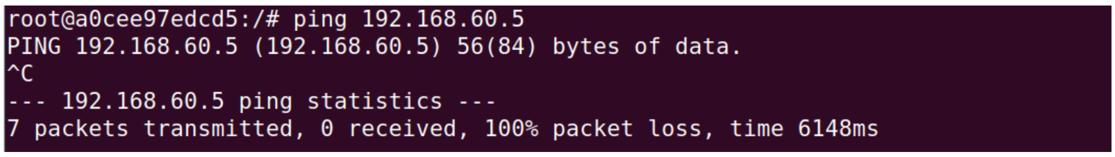
在 VPN Server 上运行 tun_server.py 和 tcpdump：


```
root@6faeff63b20c:/volumes# python3 tun_server.py
Interface Name: shen0
```

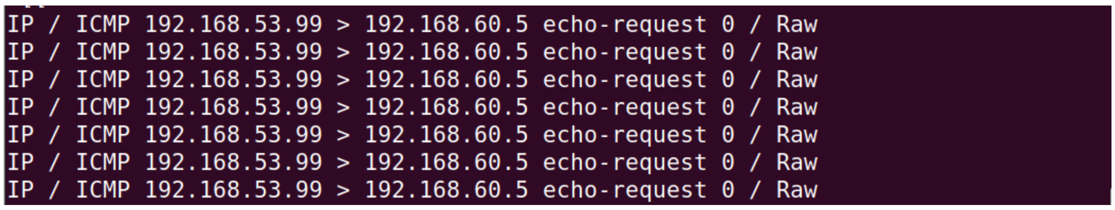
tcpdump 跟踪 eth1 端口：


```
root@6faeff63b20c:/# tcpdump -nni eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
```

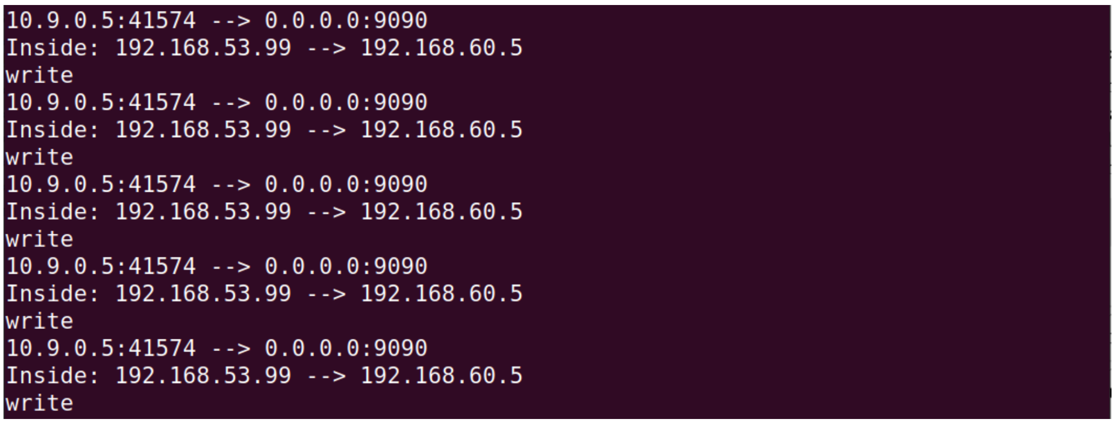
在 Host U 上尝试 ping 192.168.53.0/24 网段上的主机，ping 不通：


```
root@a0cee97edcd5:/# 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 ---
7 packets transmitted, 0 received, 100% packet loss, time 6148ms
```

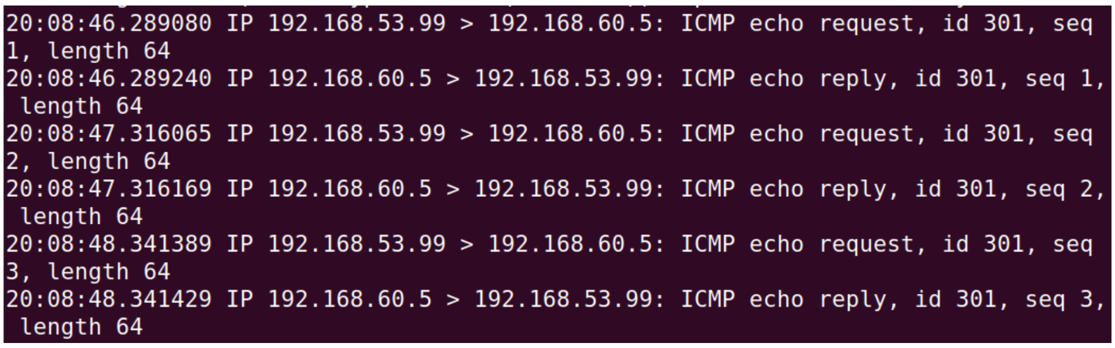
tun_client.py 打印内容：


```
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
```

tun_server.py 打印内容：


```
10.9.0.5:41574 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
write
10.9.0.5:41574 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
write
10.9.0.5:41574 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
write
10.9.0.5:41574 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
write
10.9.0.5:41574 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
write
```

tcpdump 打印内容，说明 request 包被送至 192.168.60.5 了，ping 不通说明 reply 包没有送回 10.9.0.5：


```
20:08:46.289080 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 301, seq
1, length 64
20:08:46.289240 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 301, seq 1,
 length 64
20:08:47.316065 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 301, seq
2, length 64
20:08:47.316169 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 301, seq 2,
 length 64
20:08:48.341389 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 301, seq
3, length 64
20:08:48.341429 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 301, seq 3,
 length 64
```

Task 5: Handling Traffic in Both Directions

Testing Result：

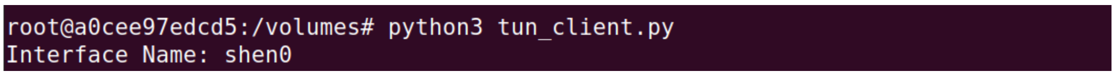
修改 tun_client.py 代码如下：

```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'shen%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))
os.system("ip route add 192.168.60.0/24 dev {}".format(ifname))
print("Interface Name: {}".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:
    # this will block until at least one interface is ready
    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))
```

修改 tun_server.py 代码如下：

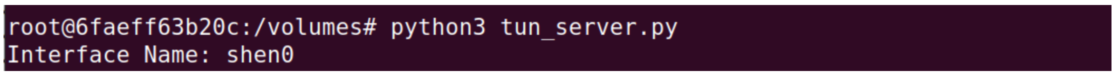
```python
#!/usr/bin/env python3
from scapy.all import *
import fcntl
import struct
import os
import time

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'shen%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 192.168.53.98/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
print("Interface Name: {}".format(ifname))

IP_A = "0.0.0.0"
PORT = 9090
ip = '10.9.0.5'
port = 10000
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))
fds = [sock,tun]
while True:
    # this will block until at least one interface is ready
    ready, _, _ = select.select(fds, [], [])
    for fd in ready:
        if fd is sock:
            print("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:
            print("tun...")
            packet = os.read(tun, 2048)
            pkt = IP(packet)
            print("Return:{} --- {}".format(pkt.src,pkt.dst))
            sock.sendto(packet,(ip,port))
```

在 Host U 上运行 tun_client.py：



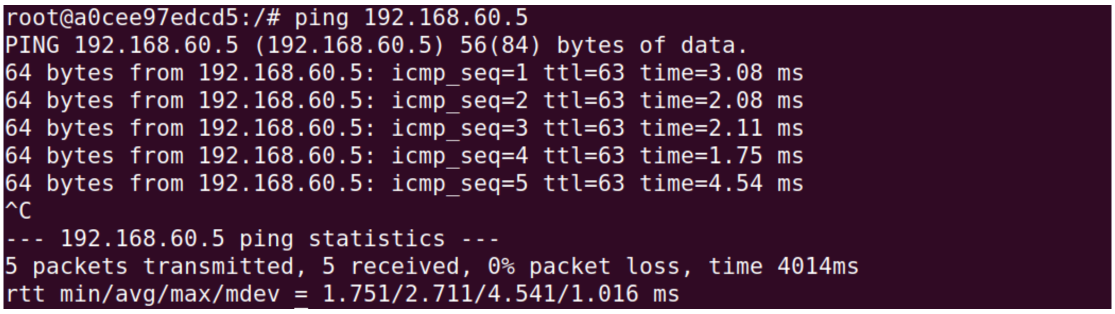
```
root@a0cee97edcd5:/volumes# python3 tun_client.py
Interface Name: shen0
```

在 VPN Server 上运行 tun_server.py：



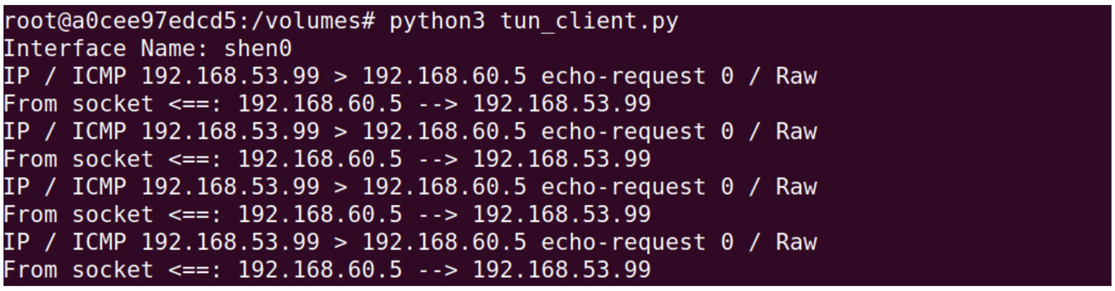
```
root@6faeff63b20c:/volumes# python3 tun_server.py
Interface Name: shen0
```

在 Host U 上尝试 ping 192.168.60.5，可以 ping 通了：



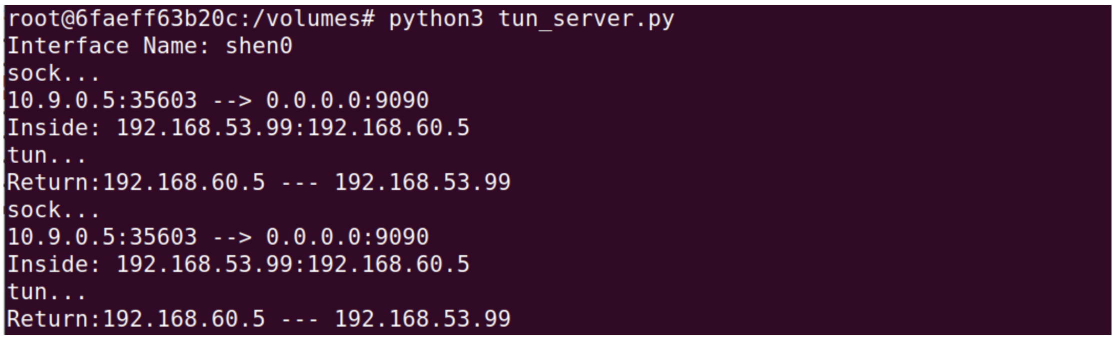
```
root@a0cee97edcd5:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=3.08 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=2.08 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=2.11 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=1.75 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=4.54 ms
^C
--- 192.168.60.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4014ms
rtt min/avg/max/mdev = 1.751/2.711/4.541/1.016 ms
```

tun_client.py 打印内容：



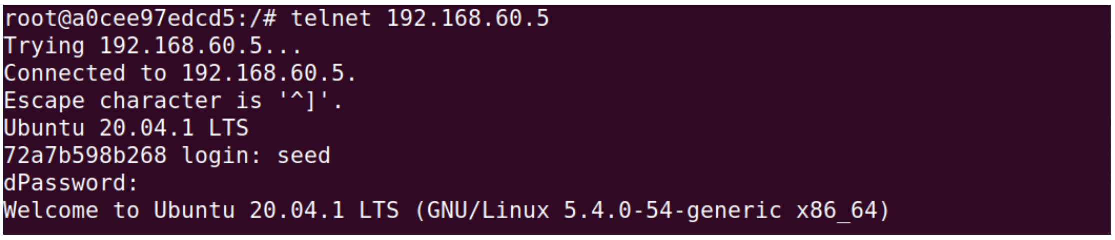
```
root@a0cee97edcd5:/volumes# python3 tun_client.py
Interface Name: shen0
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
From socket <==: 192.168.60.5 --> 192.168.53.99
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
From socket <==: 192.168.60.5 --> 192.168.53.99
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
From socket <==: 192.168.60.5 --> 192.168.53.99
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
From socket <==: 192.168.60.5 --> 192.168.53.99
```

tun_server.py 打印内容：



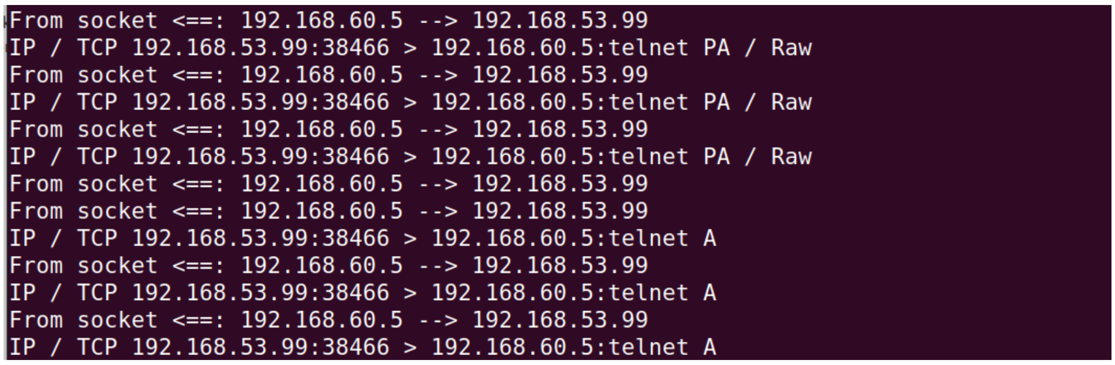
```
root@6faeff63b20c:/volumes# python3 tun_server.py
Interface Name: shen0
sock...
10.9.0.5:35603 --> 0.0.0.0:9090
Inside: 192.168.53.99:192.168.60.5
tun...
Return:192.168.60.5 --- 192.168.53.99
sock...
10.9.0.5:35603 --> 0.0.0.0:9090
Inside: 192.168.53.99:192.168.60.5
tun...
Return:192.168.60.5 --- 192.168.53.99
```

再尝试 telnet，也能够成功连接：



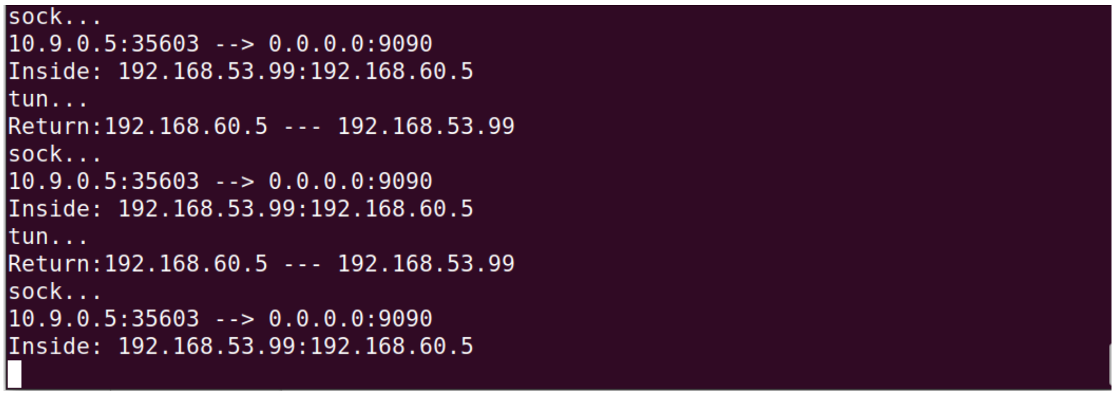
```
root@a0cee97edcd5:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
72a7b598b268 login: seed
dPassword:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

tun_client.py 打印内容：



```
From socket <==: 192.168.60.5 --> 192.168.53.99
IP / TCP 192.168.53.99:38466 > 192.168.60.5:telnet PA / Raw
From socket <==: 192.168.60.5 --> 192.168.53.99
IP / TCP 192.168.53.99:38466 > 192.168.60.5:telnet PA / Raw
From socket <==: 192.168.60.5 --> 192.168.53.99
IP / TCP 192.168.53.99:38466 > 192.168.60.5:telnet PA / Raw
From socket <==: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.60.5 --> 192.168.53.99
IP / TCP 192.168.53.99:38466 > 192.168.60.5:telnet A
From socket <==: 192.168.60.5 --> 192.168.53.99
IP / TCP 192.168.53.99:38466 > 192.168.60.5:telnet A
From socket <==: 192.168.60.5 --> 192.168.53.99
IP / TCP 192.168.53.99:38466 > 192.168.60.5:telnet A
```

tun_server.py 打印内容：

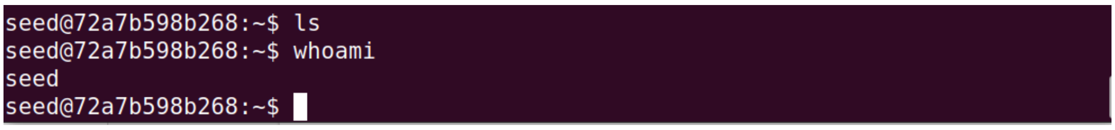


```
sock...
10.9.0.5:35603 --> 0.0.0.0:9090
Inside: 192.168.53.99:192.168.60.5
tun...
Return:192.168.60.5 --- 192.168.53.99
sock...
10.9.0.5:35603 --> 0.0.0.0:9090
Inside: 192.168.53.99:192.168.60.5
tun...
Return:192.168.60.5 --- 192.168.53.99
sock...
10.9.0.5:35603 --> 0.0.0.0:9090
Inside: 192.168.53.99:192.168.60.5
```

Task 6: Tunnel-Breaking Experiment

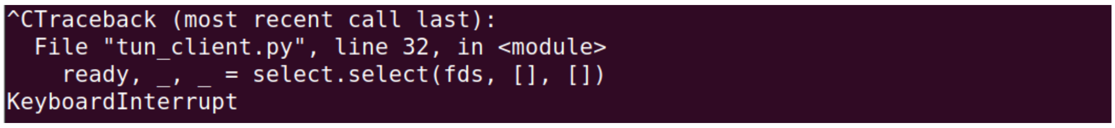
Testing Result：

续接 task5，telnet 处在正常连接状态：



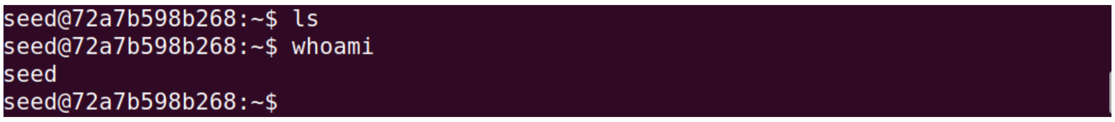
```
seed@72a7b598b268:~$ ls
seed@72a7b598b268:~$ whoami
seed
seed@72a7b598b268:~$ 
```

此时中断 tun_client.py 程序：



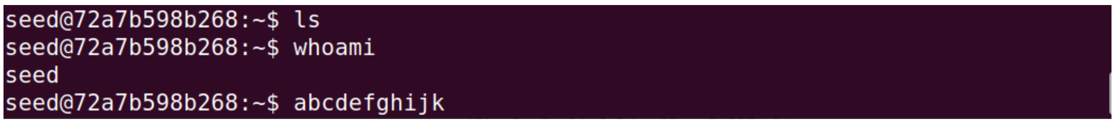
```
^CTraceback (most recent call last):
  File "tun_client.py", line 32, in <module>
    ready, _, _ = select.select(fds, [], [])
KeyboardInterrupt
```

再在 telnet 中输入内容不会有任何反应：



```
seed@72a7b598b268:~$ ls
seed@72a7b598b268:~$ whoami
seed
seed@72a7b598b268:~$
```

重新启动 tun_client.py，则之前输入的内容一次性显示了出来：



```
seed@72a7b598b268:~$ ls
seed@72a7b598b268:~$ whoami
seed
seed@72a7b598b268:~$ abcdefghijk
```

由此可见当 tun.client.py 程序终止后，待发送的报文会存储在 tun 接口的 buffer 中，待连接再次建立时一并发送。