# VPN Tunneling Lab

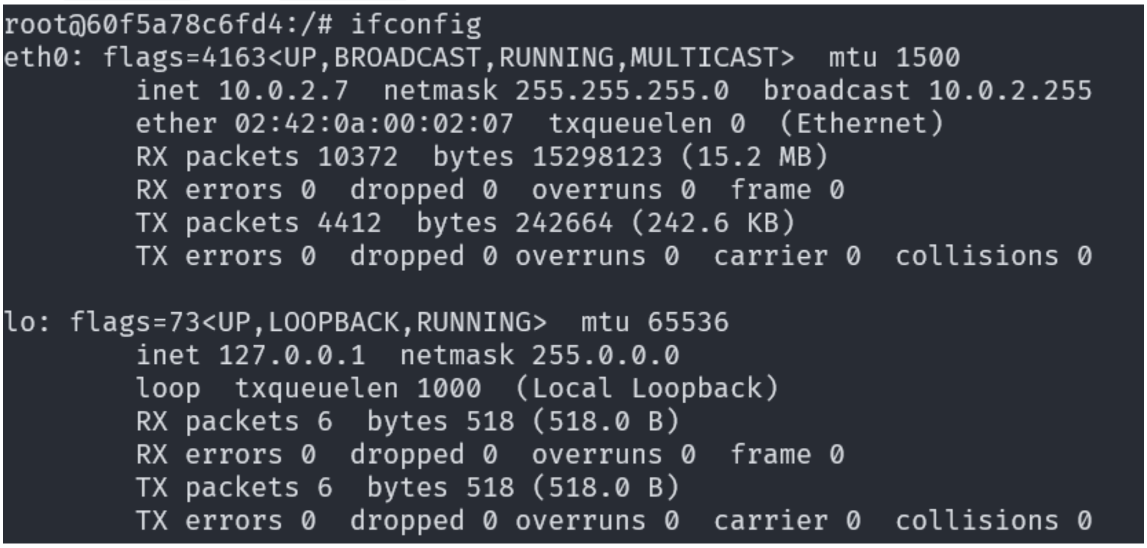
57117213 张曙

## Task1: Network Setup

首先，创建两个Docker的`bridge`类型的网络`internet`与`intranet`：

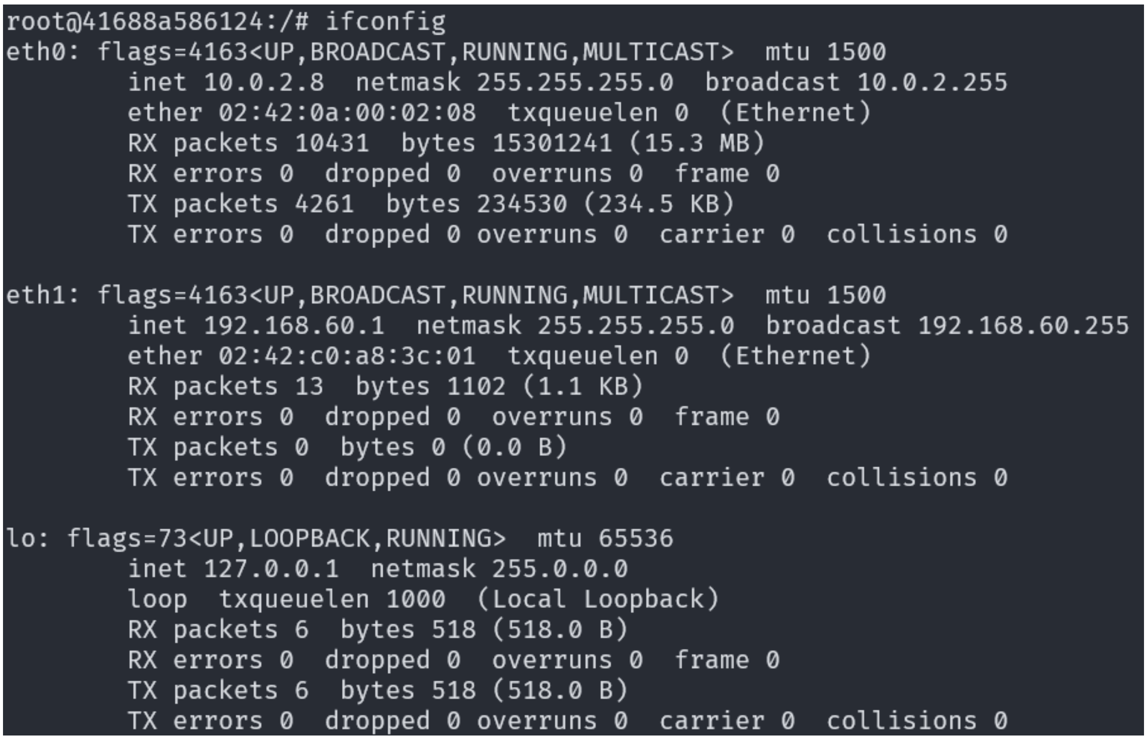
```
docker network create --subnet=10.0.2.0/24 internet
docker network create --subnet=192.168.60.0/24 intranet
```

然后分别创建主机U容器、VPN服务器容器、主机V容器，其中：

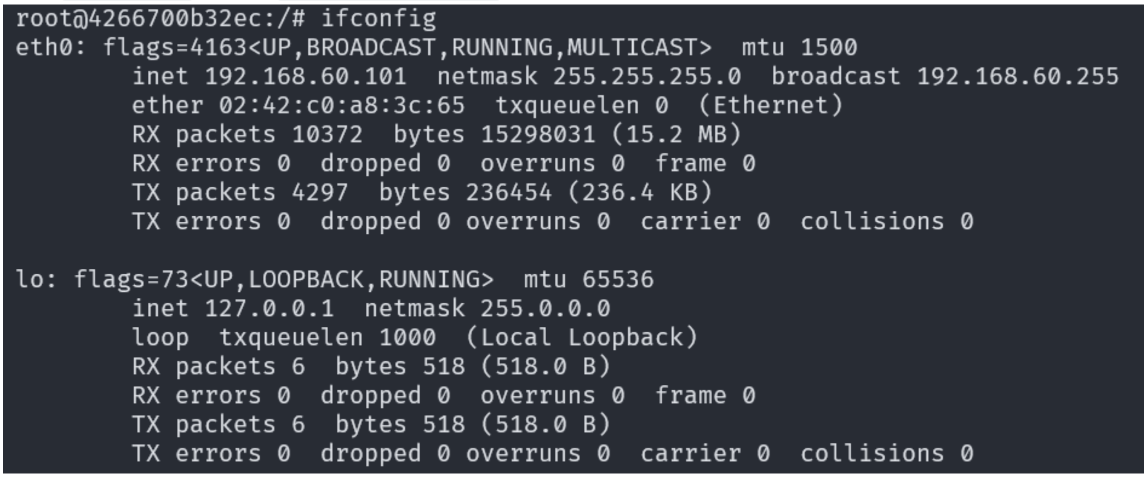
- 主机U容器
  - 容器名为`60f5a78c6fd4`
  - 连接`internet`，IP为`10.0.2.7`
  - 

```
root@60f5a78c6fd4:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.7  netmask 255.255.255.0  broadcast 10.0.2.255
        ether 02:42:0a:00:02:07  txqueuelen 0  (Ethernet)
        RX packets 10372  bytes 15298123 (15.2 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4412  bytes 242664 (242.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 6  bytes 518 (518.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6  bytes 518 (518.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

- VPN服务器容器
  - 容器名为`41688a586124`
  - 连接`internet`，IP为`10.0.2.8`
  - 连接`intranet`，IP为`192.168.60.1`

- 

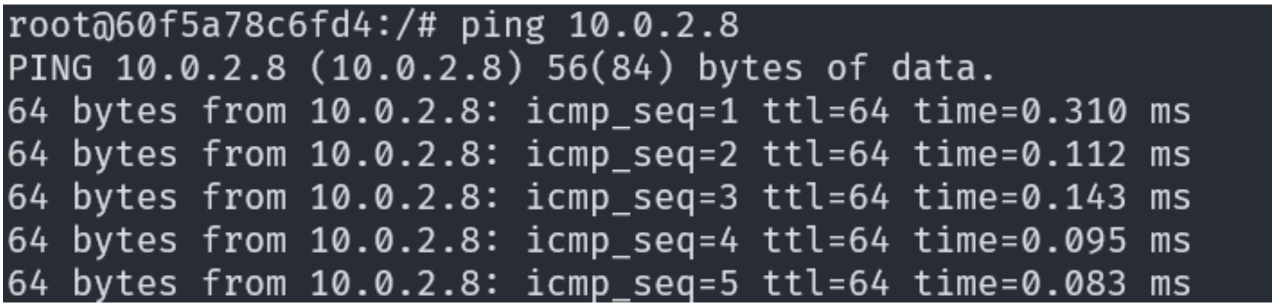
```
root@41688a586124:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.8  netmask 255.255.255.0  broadcast 10.0.2.255
        ether 02:42:0a:00:02:08  txqueuelen 0  (Ethernet)
        RX packets 10431  bytes 15301241 (15.3 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4261  bytes 234530 (234.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.60.1  netmask 255.255.255.0  broadcast 192.168.60.255
        ether 02:42:c0:a8:3c:01  txqueuelen 0  (Ethernet)
        RX packets 13  bytes 1102 (1.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 6  bytes 518 (518.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6  bytes 518 (518.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

- 主机V容器
  - 容器名为`4266700b32ec`
  - 连接`intranet`，IP为`192.168.60.101`
  - 

```
root@4266700b32ec:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.60.101  netmask 255.255.255.0  broadcast 192.168.60.255
        ether 02:42:c0:a8:3c:65  txqueuelen 0  (Ethernet)
        RX packets 10372  bytes 15298031 (15.2 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4297  bytes 236454 (236.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 6  bytes 518 (518.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6  bytes 518 (518.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

# Testing

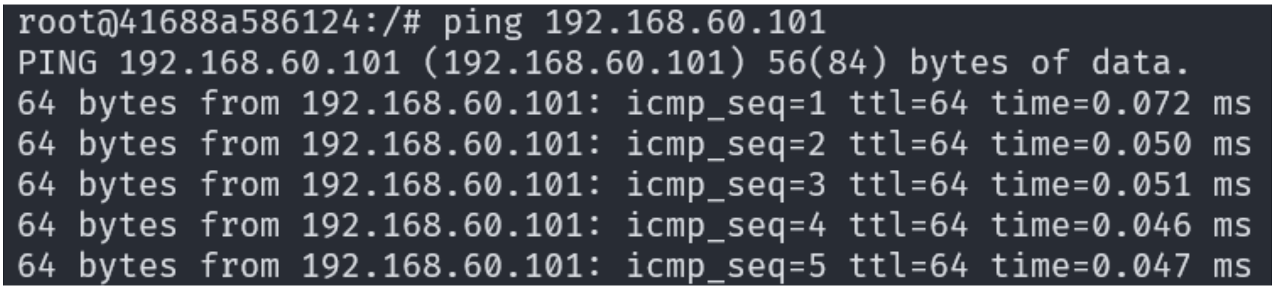
## 主机U连接VPN服务器



```
root@60f5a78c6fd4:/# ping 10.0.2.8
PING 10.0.2.8 (10.0.2.8) 56(84) bytes of data.
64 bytes from 10.0.2.8: icmp_seq=1 ttl=64 time=0.310 ms
64 bytes from 10.0.2.8: icmp_seq=2 ttl=64 time=0.112 ms
64 bytes from 10.0.2.8: icmp_seq=3 ttl=64 time=0.143 ms
64 bytes from 10.0.2.8: icmp_seq=4 ttl=64 time=0.095 ms
64 bytes from 10.0.2.8: icmp_seq=5 ttl=64 time=0.083 ms
```

成功连接。

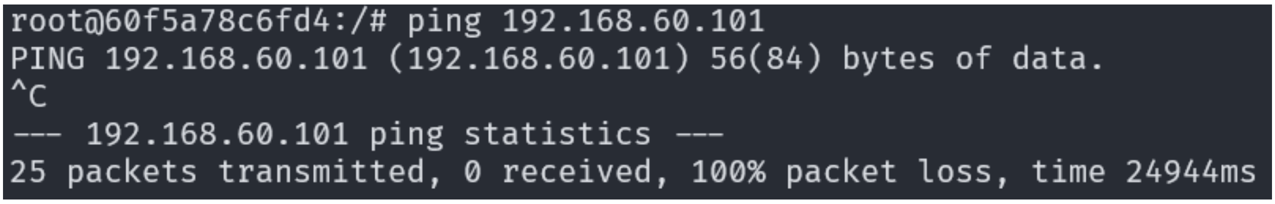
### VPN服务器连接主机V



```
root@41688a586124:/# ping 192.168.60.101
PING 192.168.60.101 (192.168.60.101) 56(84) bytes of data.
64 bytes from 192.168.60.101: icmp_seq=1 ttl=64 time=0.072 ms
64 bytes from 192.168.60.101: icmp_seq=2 ttl=64 time=0.050 ms
64 bytes from 192.168.60.101: icmp_seq=3 ttl=64 time=0.051 ms
64 bytes from 192.168.60.101: icmp_seq=4 ttl=64 time=0.046 ms
64 bytes from 192.168.60.101: icmp_seq=5 ttl=64 time=0.047 ms
```

成功连接。

### 主机U连接主机V



```
root@60f5a78c6fd4:/# ping 192.168.60.101
PING 192.168.60.101 (192.168.60.101) 56(84) bytes of data.
^C
--- 192.168.60.101 ping statistics ---
25 packets transmitted, 0 received, 100% packet loss, time 24944ms
```

无法连接。

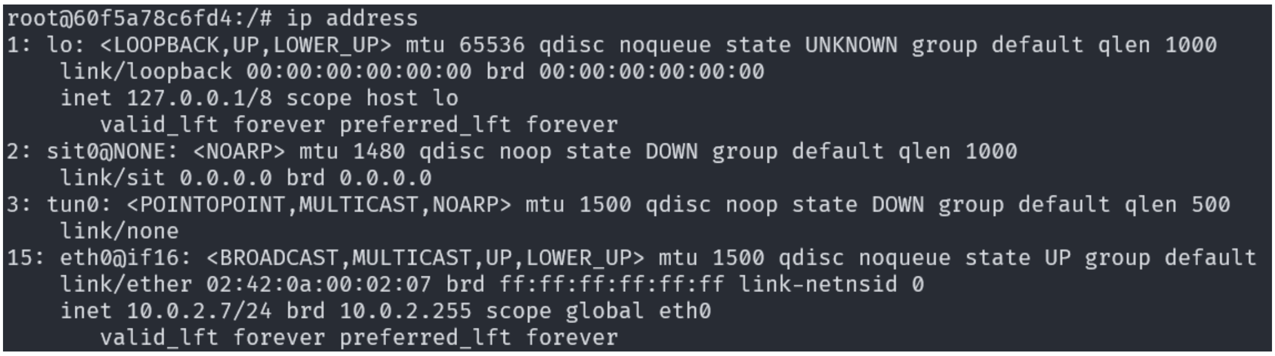
# Task 2: Cerate and Configure TUN Interface

## Task 2.a: Name of the Interface

使用题目给出的代码运行后，使用

```
ip address
```

查看所有网口信息：



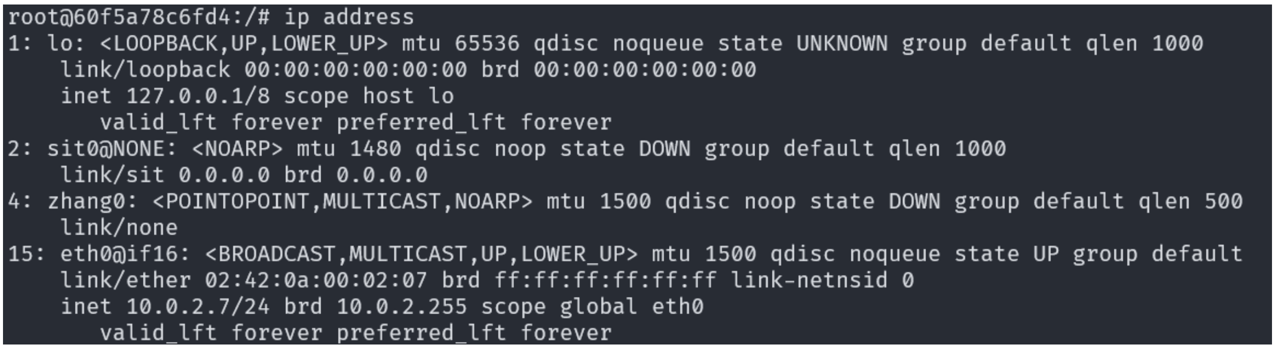
```
root@60f5a78c6fd4:/# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/sit 0.0.0.0 brd 0.0.0.0
3: tun0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 500
    link/none
15: eth0@if16: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:0a:00:02:07 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.2.7/24 brd 10.0.2.255 scope global eth0
       valid_lft forever preferred_lft forever
```

成功注册`tun0`网口。

为了将`tun0`网口名改为`zhang0`，我们只需要将代码中初始化`ifr`的代码改为：

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

即可。重新运行后查看：

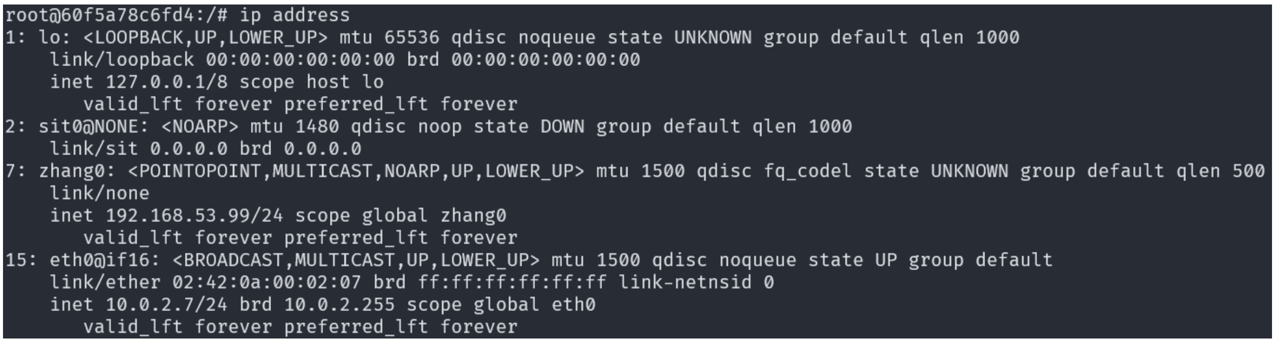


```
root@60f5a78c6fd4:/# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/sit 0.0.0.0 brd 0.0.0.0
4: zhang0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 500
    link/none
15: eth0@if16: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:0a:00:02:07 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.2.7/24 brd 10.0.2.255 scope global eth0
       valid_lft forever preferred_lft forever
```

成功注册`zhang0`。

## Task 2.b: Set up the TUN Interface

按照题目中代码编写后，运行结果为：



```
root@60f5a78c6fd4:/# 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
2: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/sit 0.0.0.0 brd 0.0.0.0
7: zhang0: <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 zhang0
       valid_lft forever preferred_lft forever
15: eth0@if16: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:0a:00:02:07 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.2.7/24 brd 10.0.2.255 scope global eth0
       valid_lft forever preferred_lft forever
```

大的区别主要有几点：

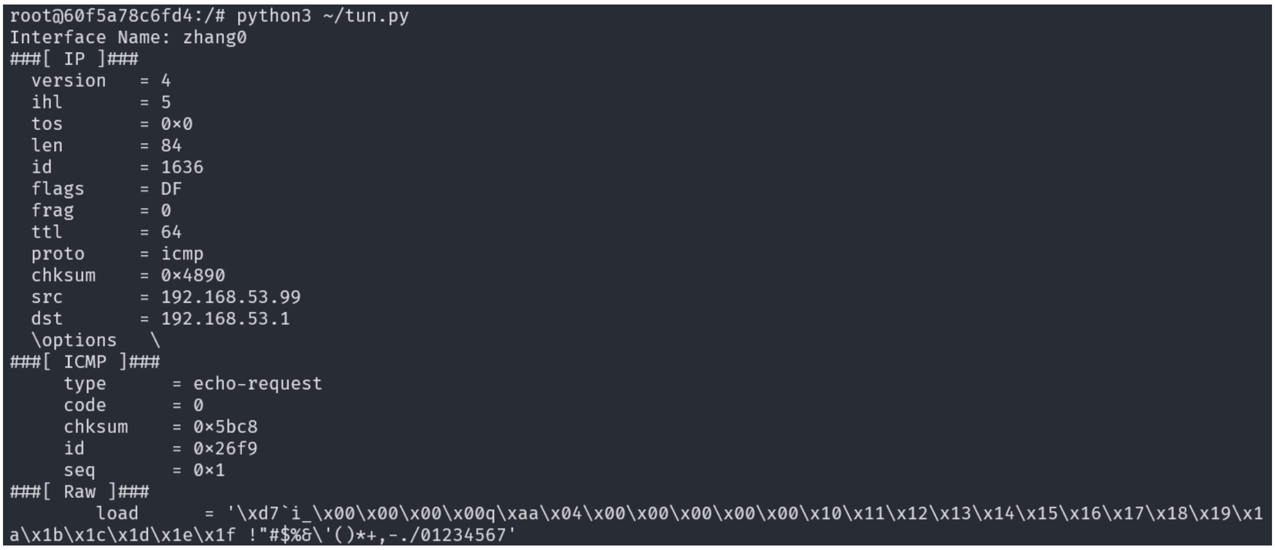
- `zhang0`前的数字由`4`变为`7`
- `zhang0`的状态Flag多出了`UP`和`LOWER_UP`
- `zhang0`有了`inet`的IP

## Task 2.c: Read from the TUN interface

按照题目要求修改代码后运行，然后在主机U的另一个shell中：

```
ping 192.168.53.1
```

输出为



```
root@60f5a78c6fd4:/# python3 ~/tun.py
Interface Name: zhang0
###[ IP ]###
  version   = 4
  ihl       = 5
  tos       = 0x0
  len       = 84
  id        = 1636
  flags     = DF
  frag      = 0
  ttl       = 64
  proto     = icmp
  chksum    = 0x4890
  src       = 192.168.53.99
  dst       = 192.168.53.1
  \options   \
###[ ICMP ]###
     type      = echo-request
     code      = 0
     chksum    = 0x5bc8
     id        = 0x26f9
     seq       = 0x1
###[ Raw ]###
        load      = '\xd7`i_\x00\x00\x00\x00q\xaa\x04\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1
a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

确实查看到了ICMP报文。

这是因为`zhang0`就在这个子网中，所以`ping`命令就是用了`zhang0`这个网口发出命令。

但如果是

```
ping 192.168.60.0
```

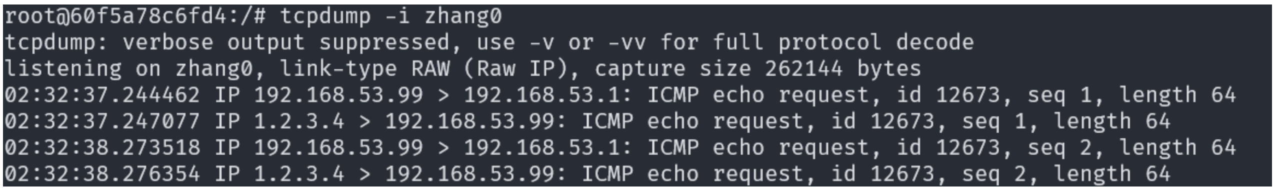
则看不到任何输出，这是因为`ping`命令如果找不到就在该子网中的网口，那么就会找第一个网口，而不是`zhang0`来发命令。

## Task 2.d: Write to the TUN Interface

按照题目要求修改代码后运行，然后

```
ping 192.168.53.1
```

并使用`tcpdump`查看：



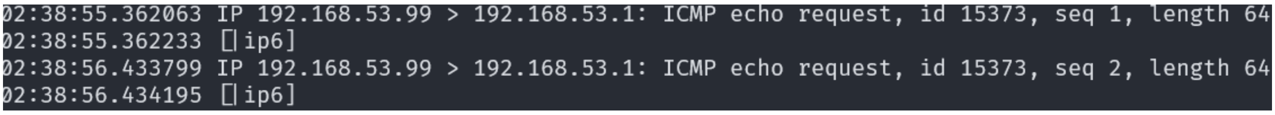
```
root@60f5a78c6fd4:/# tcpdump -i zhang0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on zhang0, link-type RAW (Raw IP), capture size 262144 bytes
02:32:37.244462 IP 192.168.53.99 > 192.168.53.1: ICMP echo request, id 12673, seq 1, length 64
02:32:37.247077 IP 1.2.3.4 > 192.168.53.99: ICMP echo request, id 12673, seq 1, length 64
02:32:38.273518 IP 192.168.53.99 > 192.168.53.1: ICMP echo request, id 12673, seq 2, length 64
02:32:38.276354 IP 1.2.3.4 > 192.168.53.99: ICMP echo request, id 12673, seq 2, length 64
```

确实看到了伪造的报文写入了TUN网口。

但如果不写IP报文，比如说把代码修改成

```
os.write(tun, b'evian')
```

那么，运行同样的命令，在`tcpdump`中只能看到



```
02:38:55.362063 IP 192.168.53.99 > 192.168.53.1: ICMP echo request, id 15373, seq 1, length 64
02:38:55.362233 [|ip6]
02:38:56.433799 IP 192.168.53.99 > 192.168.53.1: ICMP echo request, id 15373, seq 2, length 64
02:38:56.434195 [|ip6]
```

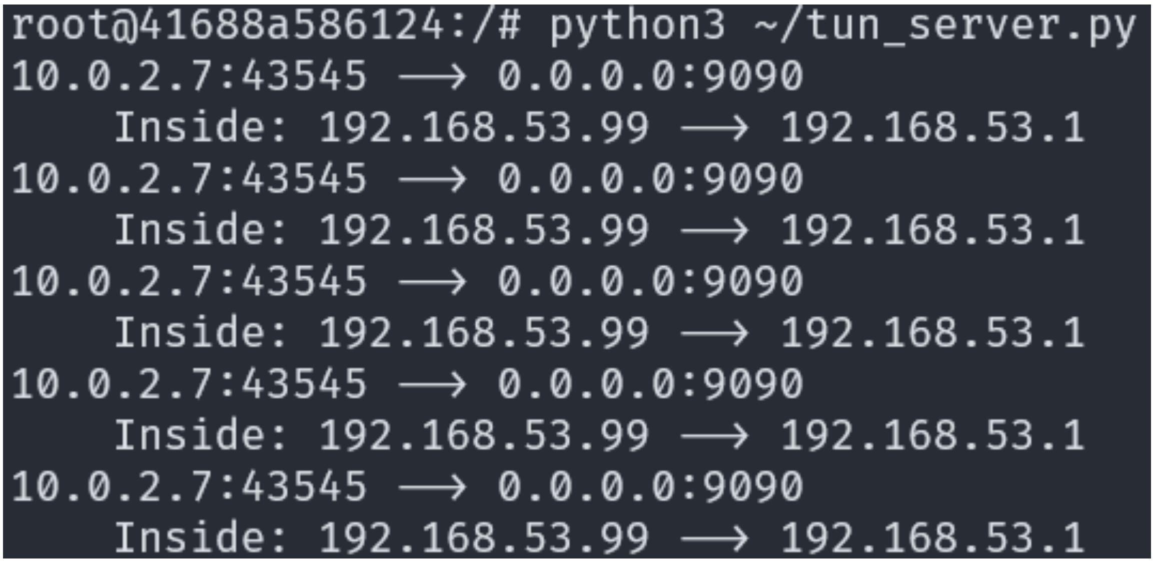
被错误识别成了奇怪的报文。

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

根据题目要求编写代码并运行后，在主机U中：

```
ping 192.168.53.1
```

在VPN服务器中可以查看到



```
root@41688a586124:/# python3 ~/tun_server.py
10.0.2.7:43545 --> 0.0.0.0:9090
    Inside: 192.168.53.99 --> 192.168.53.1
10.0.2.7:43545 --> 0.0.0.0:9090
    Inside: 192.168.53.99 --> 192.168.53.1
10.0.2.7:43545 --> 0.0.0.0:9090
    Inside: 192.168.53.99 --> 192.168.53.1
10.0.2.7:43545 --> 0.0.0.0:9090
    Inside: 192.168.53.99 --> 192.168.53.1
10.0.2.7:43545 --> 0.0.0.0:9090
    Inside: 192.168.53.99 --> 192.168.53.1
```

这是因为，根据之前的Task，当主机U中对`192.168.53.1`发出`ping`请求时，会使用TUN网口。而`tun_client`则把TUN网口收到的报文封装成UDP报文，通过与`internet`相连的`eth0`网口发送给VPN服务器，所以VPN服务器能成功收到，并且收到的TCP报文是主机U的`eth0`网口发来，其负载中包装的IP报文则是由TUN网口发来。

如果直接在主机U中`ping`目的IP`192.168.60.101`，VPN服务器是不会收到报文的，原因在Task2中已陈述。为了解决这个问题，需要在主机U中添加静态路由

```
route add -net 192.168.60.0/24 zhang0
```

也就是把所有发向`192.168.60.0/24`子网的报文由TUN网口`zhang0`发出。

通过这样的修改，VPN服务器成功接收到ICMP请求。

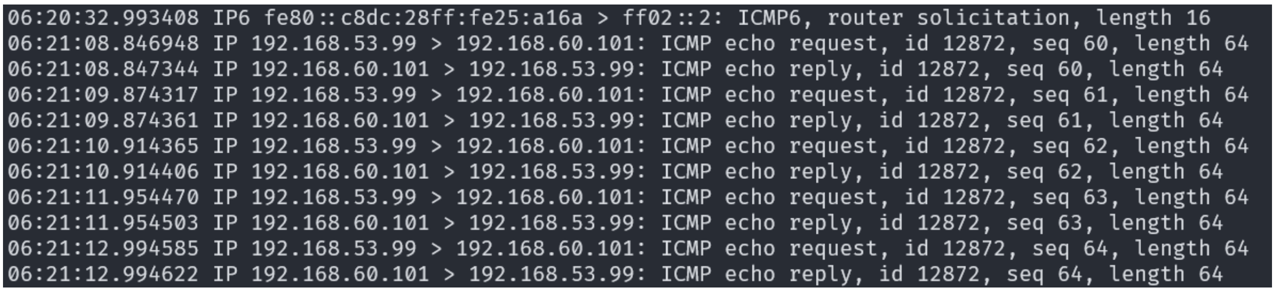
# Task 4: Set Up the VPN Server

```
import fcntl
import struct
import os
import time
from scapy.all import *

TUNSETIFF = 0x400454ca
```

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

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

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

os.system("ip addr add 192.168.53.98/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

IP_A = "0.0.0.0"
PORT = 9090

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))

while True:
    data, (ip, port) = sock.recvfrom(2048)
    print("{}:{} --> {}:{}".format(ip, port, IP_A, PORT))
    pkt = IP(data)
    print("    Inside: {} --> {}".format(pkt.src, pkt.dst))
    print("Sending raw: {}".format(data))
    os.write(tun, data)
```

经过修改后的`tun_server.py`如上。

运行后，在主机U中`ping`主机V：`192.168.60.101`，在主机V中可以通过`tcpdump`查看到



```
06:20:32.993408 IP6 fe80::c8dc:28ff:fe25:a16a > ff02::2: ICMP6, router solicitation, length 16
06:21:08.846948 IP 192.168.53.99 > 192.168.60.101: ICMP echo request, id 12872, seq 60, length 64
06:21:08.847344 IP 192.168.60.101 > 192.168.53.99: ICMP echo reply, id 12872, seq 60, length 64
06:21:09.874317 IP 192.168.53.99 > 192.168.60.101: ICMP echo request, id 12872, seq 61, length 64
06:21:09.874361 IP 192.168.60.101 > 192.168.53.99: ICMP echo reply, id 12872, seq 61, length 64
06:21:10.914365 IP 192.168.53.99 > 192.168.60.101: ICMP echo request, id 12872, seq 62, length 64
06:21:10.914406 IP 192.168.60.101 > 192.168.53.99: ICMP echo reply, id 12872, seq 62, length 64
06:21:11.954470 IP 192.168.53.99 > 192.168.60.101: ICMP echo request, id 12872, seq 63, length 64
06:21:11.954503 IP 192.168.60.101 > 192.168.53.99: ICMP echo reply, id 12872, seq 63, length 64
06:21:12.994585 IP 192.168.53.99 > 192.168.60.101: ICMP echo request, id 12872, seq 64, length 64
06:21:12.994622 IP 192.168.60.101 > 192.168.53.99: ICMP echo reply, id 12872, seq 64, length 64
```

成功收到ICMP请求。

# Task 5: Handling Traffic in Both Directions

修改代码：

`tun_client.py`：

```
import fcntl
import struct
import os
import time
import select
from scapy.all import *

TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
SERVER_IP = "10.0.2.8"
SERVER_PORT = 9090

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

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("route add -net 192.168.60.0/24 {}".format(ifname))

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

while True:
    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, (SERVER_IP, SERVER_PORT))
```

`tun_server.py`:

```
import fcntl
import struct
import os
import time
import select
from scapy.all import *
```

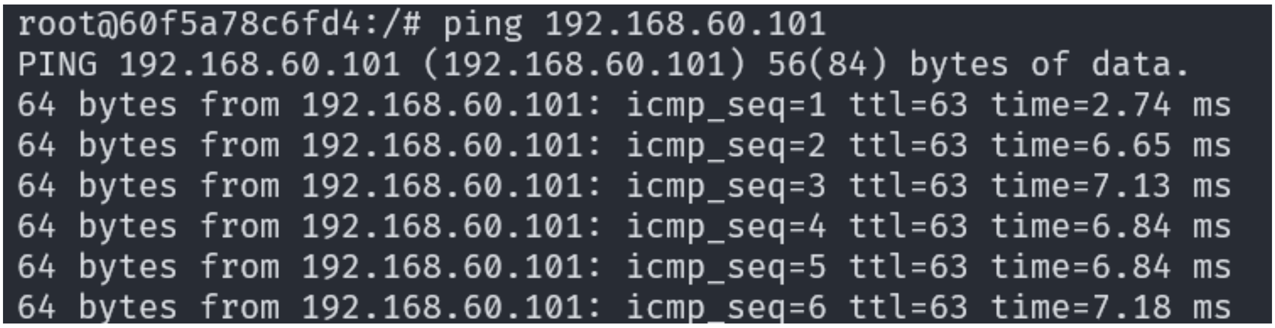
```
TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000

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

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

os.system("ip addr add 192.168.53.98/24 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))

port

while True:
    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, ("10.0.2.7", port))
```

然后在主机V中添加路由表项：

```
ip route add 192.168.53.0/24 gw 192.168.60.1 eth0
```

使ICMP响应报文都前往VPN服务器。

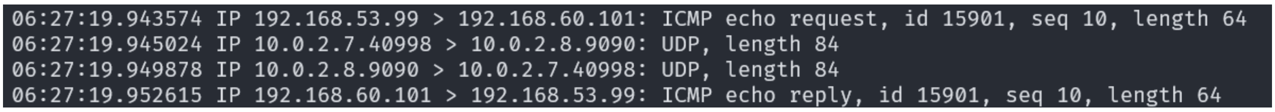
设置完成后，在主机U上再次`ping`主机V：



```
root@60f5a78c6fd4:/# ping 192.168.60.101
PING 192.168.60.101 (192.168.60.101) 56(84) bytes of data.
64 bytes from 192.168.60.101: icmp_seq=1 ttl=63 time=2.74 ms
64 bytes from 192.168.60.101: icmp_seq=2 ttl=63 time=6.65 ms
64 bytes from 192.168.60.101: icmp_seq=3 ttl=63 time=7.13 ms
64 bytes from 192.168.60.101: icmp_seq=4 ttl=63 time=6.84 ms
64 bytes from 192.168.60.101: icmp_seq=5 ttl=63 time=6.84 ms
64 bytes from 192.168.60.101: icmp_seq=6 ttl=63 time=7.18 ms
```

成功`ping`通。

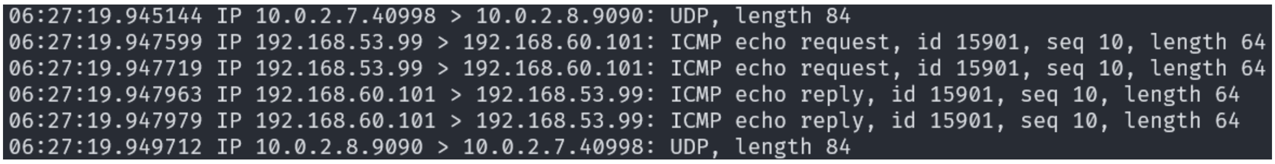
此时，主机U上的`tcpdump`为：



```
06:27:19.943574 IP 192.168.53.99 > 192.168.60.101: ICMP echo request, id 15901, seq 10, length 64
06:27:19.945024 IP 10.0.2.7.40998 > 10.0.2.8.9090: UDP, length 84
06:27:19.949878 IP 10.0.2.8.9090 > 10.0.2.7.40998: UDP, length 84
06:27:19.952615 IP 192.168.60.101 > 192.168.53.99: ICMP echo reply, id 15901, seq 10, length 64
```

说明由TUN网口向主机V发送ICMP请求，`tun_client.py`将这个请求封装成UDP报文后，转发给位于`10.0.2.8`的VPN服务器。VPN服务器由返回给`internet`端口一个UDP报文，`tun_client.py`将这个报文解析后，把其中的IP报文写入TUN网口，我们就可以看到最后一行的主机V向TUN网口发送了ICMP响应。

VPN服务器上的`tcpdump`为：



```
06:27:19.945144 IP 10.0.2.7.40998 > 10.0.2.8.9090: UDP, length 84
06:27:19.947599 IP 192.168.53.99 > 192.168.60.101: ICMP echo request, id 15901, seq 10, length 64
06:27:19.947719 IP 192.168.53.99 > 192.168.60.101: ICMP echo request, id 15901, seq 10, length 64
06:27:19.947963 IP 192.168.60.101 > 192.168.53.99: ICMP echo reply, id 15901, seq 10, length 64
06:27:19.947979 IP 192.168.60.101 > 192.168.53.99: ICMP echo reply, id 15901, seq 10, length 64
06:27:19.949712 IP 10.0.2.8.9090 > 10.0.2.7.40998: UDP, length 84
```

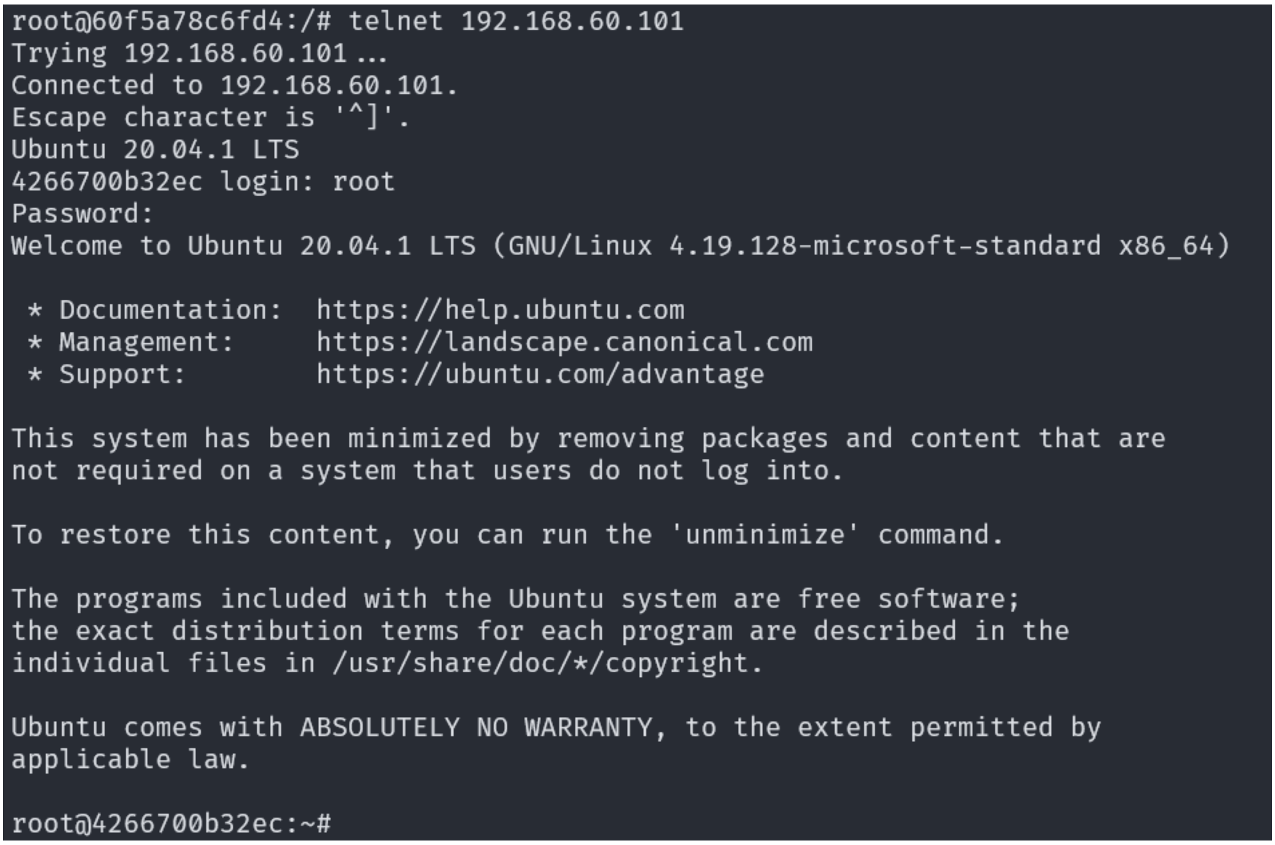
可以看到，在收到了主机U封装成UDP报文的ICMP请求后，`tun_server.py`解析出ICMP请求，发送给主机V，主机V再根据路由表将ICMP响应发送给VPN服务器，VPN服务器将ICMP响应封装在UDP报文中再发回给主机U。

主机V上的`tcpdump`结果与Task 4相同，就是收到ICMP请求，返回ICMP响应。

综上：

1. 主机U的TUN网口向主机V发送ICMP请求
2. 主机U上的`tun_client.py`将TUN网口中的ICMP请求报文封装在UDP报文中，通过`eth0`网口发送给VPN服务器
3. VPN服务器收到UDP报文，`tun_server.py`解析后，将ICMP请求报文通过TUN网口发送给主机V
4. 主机V收到ICMP请求报文，返回ICMP响应报文，并根据路由表，将报文发往VPN服务器
5. VPN服务器的TUN网口收到ICMP响应报文，`tun_server.py`将其封装成UDP报文发送回主机U
6. 主机U的`eth0`网口收到UDP报文，`tun_client.py`将其解析成ICMP响应报文，发送给TUN网口

此外，使用`telnet`也能成功：



```
root@60f5a78c6fd4:/# telnet 192.168.60.101
Trying 192.168.60.101...
Connected to 192.168.60.101.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
4266700b32ec login: root
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 4.19.128-microsoft-standard x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@4266700b32ec:~#
```

# Task 6: Tunnel-Breaking Experiment

在主机U与主机V建立Telnet连接之后，关闭`tun_client.py`，发现无论输入什么，主机U的Telnet界面都没有显示。但此时TCP连接并没有断。再次运行`tun_client.py`，输入几个字符，等待一会儿延迟之后，会发现之前输入的字符重新出现在了Telnet界面中。

这是因为，当关闭`tun_client.py`后，之前建立的TCP连接会将内容缓存进缓冲区进入重连状态。如果我们及时恢复`tun_client.py`，那么缓冲区的字符又会重新通过TCP连接发送出去。

# Task 7: Routing Experiment on Host V

这个任务由于我一开始就是在主机V上直接

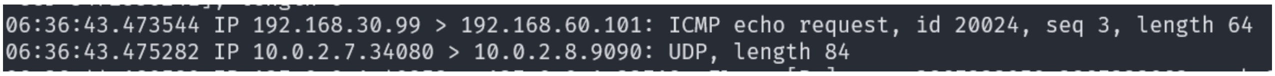
```
ip route add 192.168.53.0/24 gw 192.168.60.1 eth0
```

所以并没有使用默认路由，就相当于一开始就做了这个任务。其结果是正常运行。

# Task 8: Experiment with TUN IP Address

将主机U上的TUN网口IP地址改为`192.168.30.99`后，无法`ping`通。

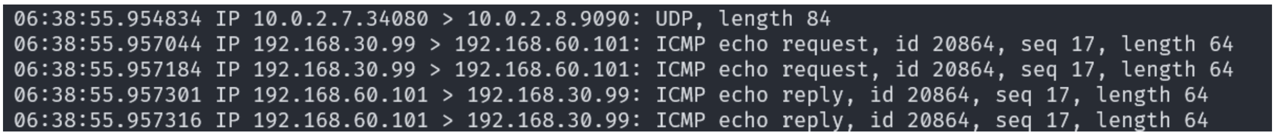
首先，查看主机U上的`tcpdump`结果：



```
06:36:43.473544 IP 192.168.30.99 > 192.168.60.101: ICMP echo request, id 20024, seq 3, length 64
06:36:43.475282 IP 10.0.2.7.34080 > 10.0.2.8.9090: UDP, length 84
```

只将其封装成UDP发送出去了，但并没有收到返回的UDP。

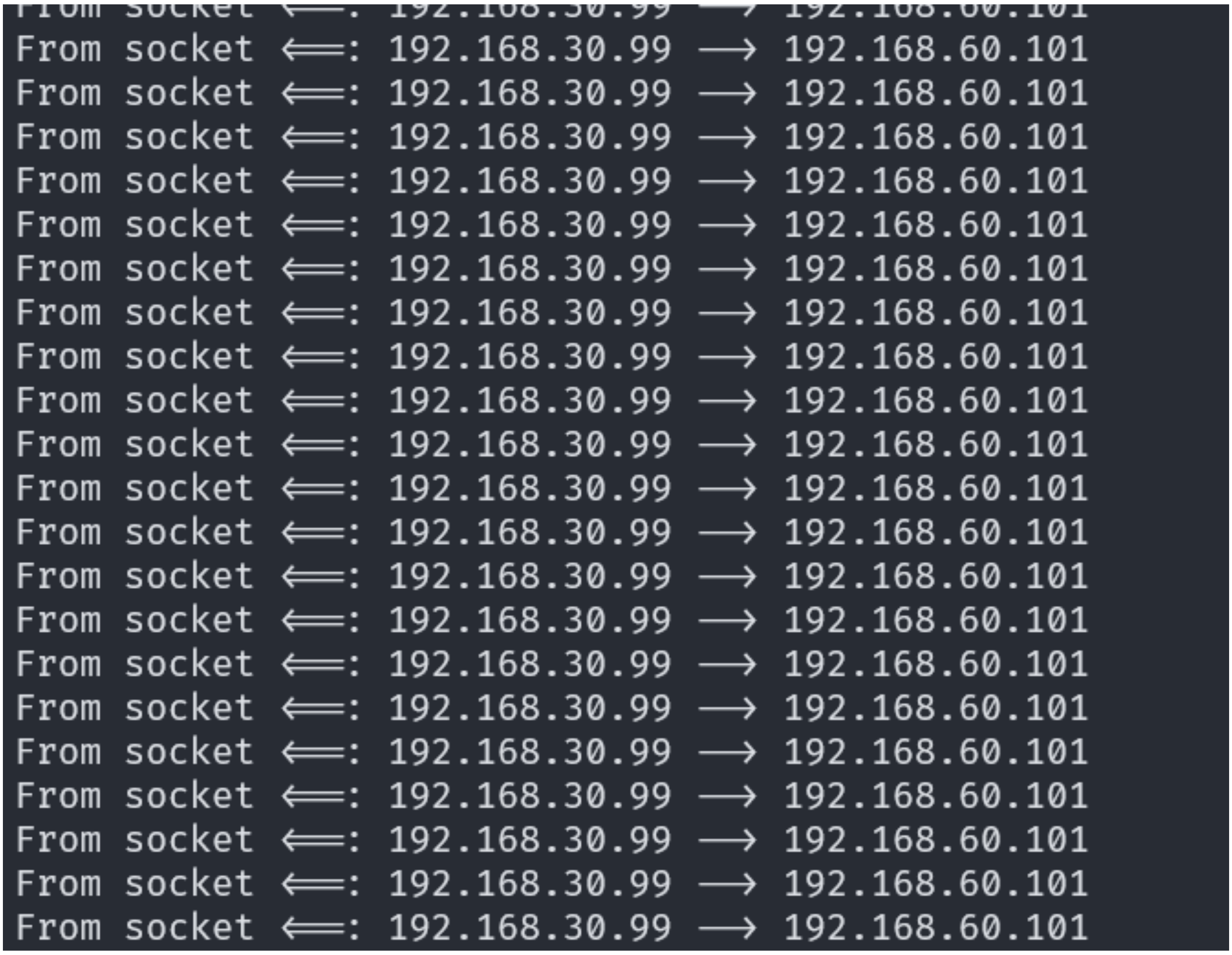
然后，在VPN服务器上查看`tcpdump`结果：



```
06:38:55.954834 IP 10.0.2.7.34080 > 10.0.2.8.9090: UDP, length 84
06:38:55.957044 IP 192.168.30.99 > 192.168.60.101: ICMP echo request, id 20864, seq 17, length 64
06:38:55.957184 IP 192.168.30.99 > 192.168.60.101: ICMP echo request, id 20864, seq 17, length 64
06:38:55.957301 IP 192.168.60.101 > 192.168.30.99: ICMP echo reply, id 20864, seq 17, length 64
06:38:55.957316 IP 192.168.60.101 > 192.168.30.99: ICMP echo reply, id 20864, seq 17, length 64
```

确实收到了UDP报文，并且将其发送给了主机V，也收到了主机V的ICMP响应报文，但并没有发出返回的UDP报文。

查看`tun_server.py`的输出也可以发现



```
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
From socket ⟸: 192.168.30.99 ⟶ 192.168.60.101
```

确实没有从TUN口收到包。

说明在VPN服务器的返回UDP阶段丢包了。

理由是因为Linux中反向路径过滤功能中，会查看(`192.168.60.101`,`192.168.30.99`)在路由表中是否匹配对应的网口，但这里是不匹配的，所以就会将包丢弃。

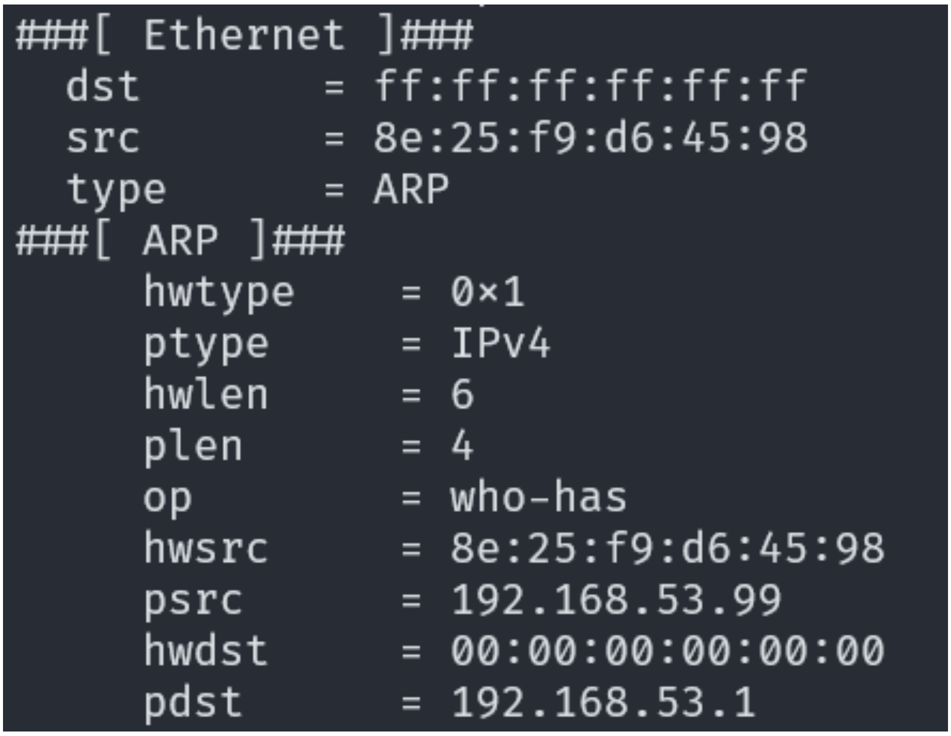
解决方法也许是关闭反向路径过滤功能?

## Task 9: Experiment with the TAP Interface

按照题目要求编写代码运行后，并在主机U上

```
ping 192.168.53.1
```

然后查看Python脚本输出：



```
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 8e:25:f9:d6:45:98
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = 6
     plen      = 4
     op        = who-has
     hwsrc     = 8e:25:f9:d6:45:98
     psrc      = 192.168.53.99
     hwdst     = 00:00:00:00:00:00
     pdst      = 192.168.53.1
```

发现TAP口确实收到了一个ARP请求。

这是因为，`ping`命令首先会根据`192.168.53.1`发出一个ARP请求，查看是否在局域网内就有这个IP的设备。而我们的TAP网口的IP与其在同一子网内，所以命令就转发到了TAP网口中。从而我们就可以查看到，这是一个ARP的`who-has`请求，查看哪个设备拥有IP`192.168.53.1`。