VPN Tunneling Lab

57117231 农禄 2020/09/23

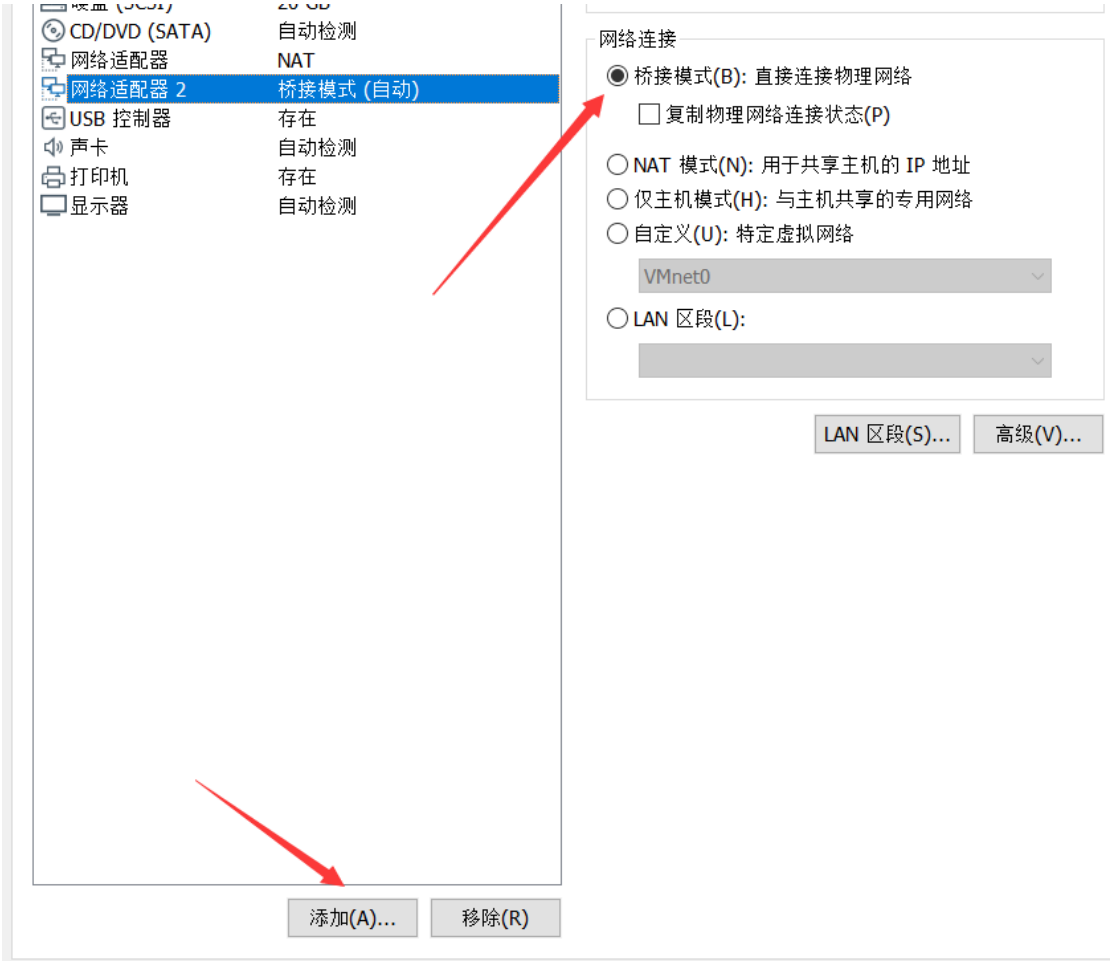
实验环境

Host G：ip 192.168.248.136 10.0.0.1
Host U：ip 192.168.248.135
Host V：ip 10.0.0.2

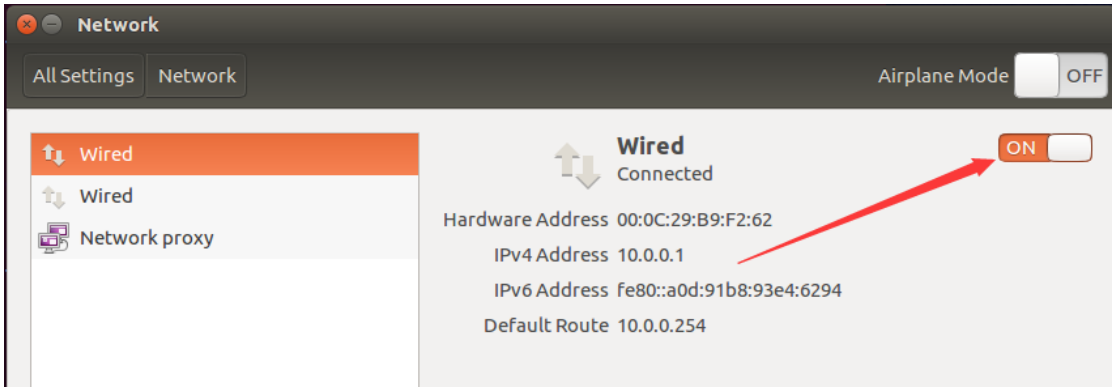
虚拟机载体：VMware

Task1: Network Setup

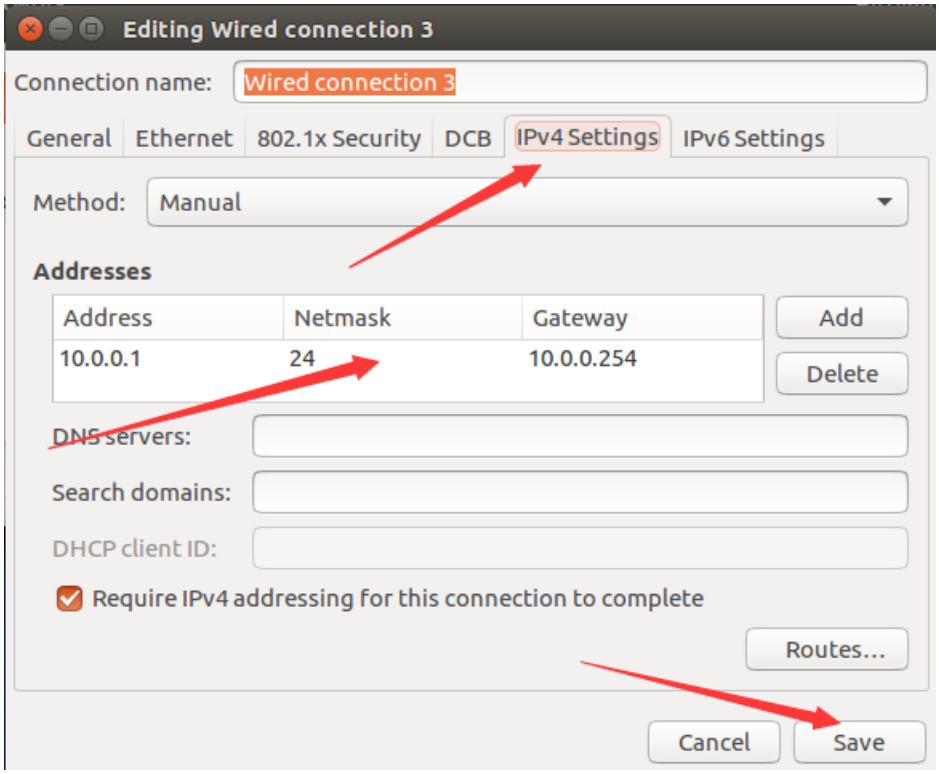
为虚拟机 G 添加新的网卡，连接模式为桥接（第一张网卡为 NAT）



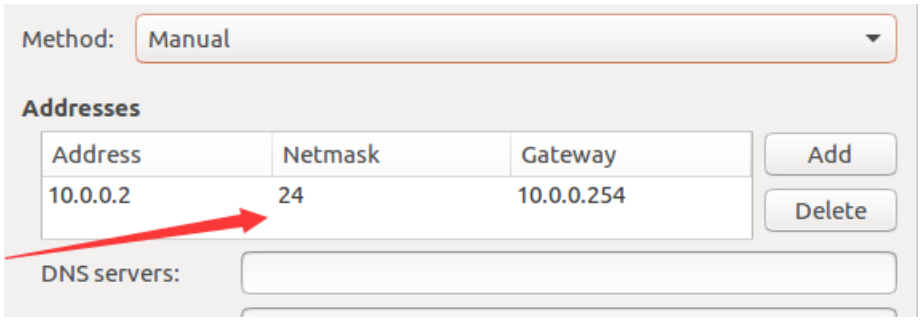
打开虚拟机 G 的 Network 设置面板，并将新网卡打开



进入 Network 面板右下角的 options，设置相应的 ip 信息(ip 10.0.0.1)



打开虚拟机 V，将其网卡设置为桥接模式。同样进入 Network 面板，设置 ip(10.0.0.2)



随后重启 G 和 V

查看 G 的 IP

```
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKN
OWN group default qlen 1000
    link/ether 00:0c:29:b9:f2:58 brd ff:ff:ff:ff:ff:ff
    inet 192.168.248.136/24 brd 192.168.248.255 scope global dynamic ens33
       valid_lft 1067sec preferred_lft 1067sec
    inet6 fe80::950b:dde1:619a:73ac/64 scope link
       valid_lft forever preferred_lft forever
3: ens38: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKN
OWN group default qlen 1000
    link/ether 00:0c:29:b9:f2:62 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.1/24 brd 10.0.0.255 scope global ens38
       valid_lft forever preferred_lft forever
    inet6 fe80::a0d:91b8:93e4:6294/64 scope link
       valid_lft forever preferred_lft forever
```

查看 V 的 IP

```
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKN
OWN group default qlen 1000
    link/ether 00:0c:29:e1:f5:f1 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.2/24 brd 10.0.0.255 scope global ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::465a:6edc:fb3f:e3d9/64 scope link
       valid_lft forever preferred_lft forever
[09/22/20]seed@VM:~$
```

查看 U 的 IP（DHCP）

```
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP g
roup default qlen 1000
    link/ether 00:0c:29:a3:29:3f brd ff:ff:ff:ff:ff:ff
    inet 192.168.248.135/24 brd 192.168.248.255 scope global dynamic ens33
       valid_lft 1702sec preferred_lft 1702sec
    inet6 fe80::5356:69ed:1302:860f/64 scope link
       valid_lft forever preferred_lft forever
[09/22/20]seed@VM:~$
```

此时 G 可以 ping V 和 U

```
[09/22/20]seed@VM:~$ ping 192.168.248.135 -c 1
PING 192.168.248.135 (192.168.248.135) 56(84) bytes of data.
64 bytes from 192.168.248.135: icmp_seq=1 ttl=64 time=1.20 ms

--- 192.168.248.135 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.207/1.207/1.207/0.000 ms
[09/22/20]seed@VM:~$ ping 10.0.0.2 -c 1
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.974 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.974/0.974/0.974/0.000 ms
```

但 U 和 V 之间不互通，同 U ping V

```
[09/22/20]seed@VM:~$ ping 10.0.0.2 -c 1
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
```

完成配置之后的 IP 信息如下

G 192.168.248.136 10.0.0.1

V 10.0.0.1

U 192.168.248.135

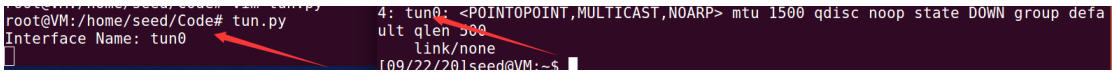
Task2: Create and Configure TUN Interface

2.a Name of the Interface

在 U 中运行如下 python 程序

```python
#!/usr/bin/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'tun%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))
```

开启另一个终端，执行命令 ip address

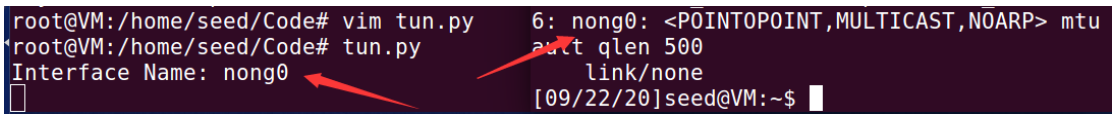


发现增加了一个名为 tun0 的网卡

修改 tun.py

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

tun0 变成了 nong0



2.b: Set up the TUN Interface

修改脚本，为新建的 tun 网卡分配 ip，并启动

```
# 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(iframe))
os.system("ip link set dev {} up".format(ifname))

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

重新执行 ip address，发现 tun 虚拟网卡获得了 ip 地址 192.168.53.99

```
8: nong0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast st
ate UNKNOWN group default qlen 500
    link/none
    inet 192.168.53.99/24 scope global nong0
       valid_lft forever preferred_lft forever
    inet6 fe80::c1fb:3413:fa6a:7afc/64 scope link flags 800
       valid_lft forever preferred_lft forever
```

Task2.c: Read from the TUN Interface

修改脚本，使其能够从 tun 虚拟网卡中读取报文

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

运行 tun.py

让虚拟机 U ping 192.168.53.1，tun 虚拟网卡监听到发往 192.168.53.1 的报文

```
  flags     = DF
  frag      = 0
  ttl       = 64
  proto     = icmp
  chksum    = 0x147
  src       = 192.168.53.99
  dst       = 192.168.53.1
  \options   \
###[ ICMP ]###
     type      = echo-request
     code      = 0
     chksum    = 0xcafb
     id        = 0x2008
     seq       = 0x1
###[ Raw ]###
        load      = '\x06Kj_\xa5M\x0c\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10
x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./012
```

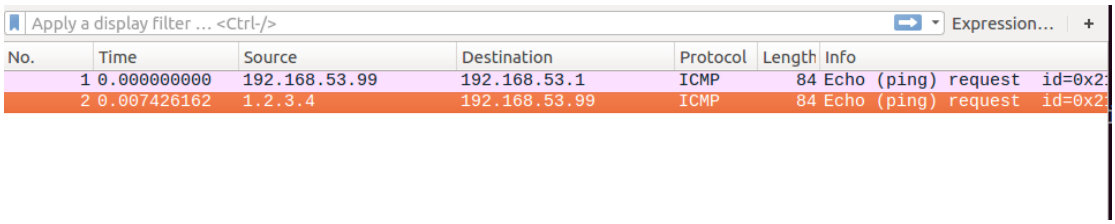
让虚拟机 U ping 10.0.0.2（即 V 的 IP），tun.py 进程无反应，说明此时发往 10.0.0.2 的报文并不会经过 tun 虚拟网卡

Task2.d: Write to the TUN Interface

根据从 tun 读取的报文构造新的报文，并通过 write 的方式交付给 tun

```
while True:
        # Get a packet from the tun interface
        packet = os.read(tun, 2048)
        if True:
                ip = IP(packet)
                # send out a spoof packet using the tun interface
                newip = IP(src='1.2.3.4', dst=ip.src)
                newpkt = newip/ip.payload
                os.write(tun, bytes(newpkt))
```

运行 tun.py 后，向 192.168.53.1 发送 icmp 报文，并用 wireshark（虚拟机 U 内的）观察 tun 虚拟网卡



可以看到，我们构造的报文成功被发送了

修改脚本，向 tun 网卡写入随机的数据

```
ip = IP(packet)
# send out a spoof packet using the tun interface
newip = IP(src='1.2.3.4', dst=ip.src)
newpkt = newip/ip.payload
# os.write(tun, bytes(newpkt))
rd = bytes("123456789", encoding = "utf8")
os.write(tun, rd)
```

重新运行并测试，程序报错

```
root@VM:/home/seed/Code# tun.py
Interface Name: nong0
Traceback (most recent call last):
  File "./tun.py", line 36, in <module>
    os.write(tun, rd)
OSError: [Errno 22] Invalid argument
root@VM:/home/seed/Code#
```

Task3: Send the IP Packet to VPN Server Through a Tunnel

在虚拟机编写 VPN 程序 tun_server.py

```
#!/usr/bin/python3

from scapy.all import *

IP_A = "0.0.0.0"
PORT = 9999

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))
```

在虚拟机 U 编写 tun_client.py（基于 tun.py）

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

# Create UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

while True:
        # Get a packet from the tun interface
        packet = os.read(tun, 2048)
        if True:
                # Send the packet via the tunnel
                sock.sendto(packet, ("192.168.248.136", 9999))
```

分别在服务端和客户端运行 tun_server.py 和 tun_client.py

用客户端 U 向 192.168.53.1 发起 ping 请求

```
root@VM:/home/seed/Code# tun_server.py
192.168.248.135:54602 --> 0.0.0.0:9999
    Inside: 0.0.0.0 --> 218.148.201.47
192.168.248.135:54602 --> 0.0.0.0:9999
    Inside: 0.0.0.0 --> 218.148.201.47
192.168.248.135:54602 --> 0.0.0.0:9999
    Inside: 0.0.0.0 --> 218.148.201.47
192.168.248.135:54602 --> 0.0.0.0:9999
    Inside: 192.168.53.99 --> 192.168.53.1
```

服务端打印出发往 192.168.53.1 的报文。原因是发往 192.168.53.1 的报文被 tun 网卡重新封装在 UDP 报文中，当服务端获取到此报文时可以还原出原始报文。

让虚拟机 U ping 虚拟机 V(10.0.0.2)，服务端 G 未收到任何消息。原因是发往 10.0.0.2 的消息不会经过虚拟网卡 tun。

添加一条路由，使 U 上发往 V 的报文都交给 192.168.53.99

```
[09/22/20]seed@VM:~$ sudo ip route add 10.0.0.2 dev nong0 via 192.168.53.99
```

再次 ping V

```
root@VM:/home/seed/Code# tun_server.py
192.168.248.135:54602 --> 0.0.0.0:9999
    Inside: 192.168.53.99 --> 10.0.0.2
```

现在服务端监听到来自 U 发往 V 的报文了。

Task4: Set up the VPN Server

修改 tun_server.py，使其创建 tun 虚拟网卡，并将 sock 收到的报文的数据部分用 scapy 转成 IP 报文，交给 tun 网卡

```
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'tun%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 192.168.53.66/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
```

```
while True:
        data, (ip, port) = sock.recvfrom(2048)
        pkt = IP(data)
        os.write(tun, bytes(pkt))
```

开启报文转发功能，使主机能像网关一样转发报文

```
root@VM:/home/seed/Code# sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
root@VM:/home/seed/Code#
```

在 V 打开 wireshark 监听报文，用 U 向 V 发起 ping 请求

No.	Time	Source	Destination	Protocol	Length	Info
1	2020-09-22 20:24:30.9239546…	Vmware_b9:f2:62	Broadcast	ARP	60	Who has 10.0
2	2020-09-22 20:24:30.9240597…	Vmware_e1:f5:f1	Vmware_b9:f2:62	ARP	42	10.0.0.2 is
3	2020-09-22 20:24:30.9247551…	192.168.53.99	10.0.0.2	ICMP	98	Echo (ping)
4	2020-09-22 20:24:30.9250205…	Vmware_e1:f5:f1	Broadcast	ARP	42	Who has 10.0
5	2020-09-22 20:24:31.9404531…	Vmware_e1:f5:f1	Broadcast	ARP	42	Who has 10.0
6	2020-09-22 20:24:32.9638344…	Vmware_e1:f5:f1	Broadcast	ARP	42	Who has 10.0

可以看到，V 收到了来自 U 的虚拟网卡 tun 的 ping 报文

Task5: Handling Traffc in Both Directions

为了方便实验，在主机 V 上添加一条路由，让发往 192.168.53.0/24 的流量转向 10.0.0.1

```
[09/22/20]seed@VM:~$ sudo ip route add 192.168.53.0/24 dev ens33 via 10.0.0.1
[09/22/20]seed@VM:~$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.0.0.254      0.0.0.0         UG    100    0        0 ens33
10.0.0.0        *               255.255.255.0   U     100    0        0 ens33
link-local      *               255.255.0.0     U     1000   0        0 ens33
192.168.53.0    10.0.0.1        255.255.255.0   UG    0      0        0 ens33
```

对于客户端 U，设置好虚拟网卡 tun 的 IP 和路由（让发往 10.0.0.0/24 的数据通过 tun）
代码如下

```
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 10.0.0.0/24 {}".format(ifname))
```

使用系统调用的 select，同时对 sock 和 tun 进行堵塞监听

```
import select as sl
```

```
ready,_,_ = sl.select([sock, tun], [], [])
```

sock 里的消息是 VPN 服务端发送回来的报文，UDP 报文的数据即为真正的报文，将其交给 tun，由于此时报文的目的 IP 即为 tun 的 IP，因此不会继续转发；
tun 里的消息是从客户端发出的消息，我们需要将其封装在 UDP 报文里，并发送给 VPN 服务器。代码如下：

```
ready,_,_ = sl.select([sock, tun], [], [])

for fd in ready:
        if fd is sock:
                # from server
                data, (ip, port) = sock.recvfrom(2048)
                pkt = IP(data)
                print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))
                os.write(tun, bytes(pkt))
        if fd is tun:
                # from itself
                rpkt = os.read(tun, 2048)
                pkt = IP(rpkt)
                print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))
                sock.sendto(rpkt, ("192.168.248.136", 9999))
```

对于服务端 U，设置网卡并启动，并且开启 UDP 服务。

```
os.system("ip addr add 192.168.53.66/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

IP_A = "0.0.0.0"
PORT = 9999

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

同样，对 sock 和 tun 进行阻塞监听。提取 sock 的消息（来自客户端）的数据部分并封装成 IP 报文，如果 IP 报文的源地址是客户端虚拟网卡的地址，则将其交给自身的虚拟网卡 tun。虚拟网卡 tun 会根据目的地址转发给 V。同时，记录客户端的端口号；
虚拟网卡 tun 收到的消息是来自 V 的，所以我们需要将其封装到 UDP 报文中，并发送给客户端。代码如下

```
# this will block until at least one interface is ready
ready, _, _ = sl.select([sock, tun], [], [])

for fd in ready:
        if fd is sock:
                # from client
                data, (ip, port) = sock.recvfrom(2048)
                pkt = IP(data)
                print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))
                if pkt.src=="192.168.53.99":
                        os.write(tun, bytes(pkt))
                        dport = port
        if fd is tun:
                # from V
                rpkt = os.read(tun, 2048)
                pkt = IP(rpkt)
                print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))
                if pkt.dst == "192.168.53.99":
                        sock.sendto(rpkt, (dip,dport))
```

进行测试，用 U ping V，成功连通

```
[09/23/20]seed@VM:~$ ping 10.0.0.2 -c 1
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=63 time=4.42 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 4.423/4.423/4.423/0.000 ms
[09/23/20]seed@VM:~$
```

```
From tun ==>: 192.168.53.99 --> 10.0.0.2
From socket <==: 10.0.0.2 --> 192.168.53.99
```

→	1	2020-09-23 08:03:39.6550287…	192.168.53.99	10.0.0.2	ICMP	84 Echo (ping)
←	2	2020-09-23 08:03:39.6556103…	10.0.0.2	192.168.53.99	ICMP	84 Echo (ping)

用 U 对 V 发起 telnet 连接，成功发起尝试。

```
[09/23/20]seed@VM:~$ telnet 10.0.0.2
Trying 10.0.0.2...
Connected to 10.0.0.2.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login:
```

```
From tun ==>: 192.168.53.99 --> 10.0.0.2
From tun ==>: 192.168.53.99 --> 10.0.0.2
From socket <==: 10.0.0.2 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 10.0.0.2
From socket <==: 10.0.0.2 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 10.0.0.2
From socket <==: 10.0.0.2 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 10.0.0.2
```

1	2020-09-23 08:13:53.5136487…	192.168.53.99	10.0.0.2	TCP	60	59560 → 23 [SYN] Seq=8819833
2	2020-09-23 08:13:53.5142115…	10.0.0.2	192.168.53.99	TCP	60	23 → 59560 [SYN, ACK] Seq=29
3	2020-09-23 08:13:53.5240856…	192.168.53.99	10.0.0.2	TCP	52	59560 → 23 [ACK] Seq=8819833
4	2020-09-23 08:13:53.5268324…	192.168.53.99	10.0.0.2	TELNET	79	Telnet Data ...
5	2020-09-23 08:13:53.5279545…	10.0.0.2	192.168.53.99	TCP	52	23 → 59560 [ACK] Seq=2958989
6	2020-09-23 08:13:53.5323527…	10.0.0.2	192.168.53.99	TELNET	64	Telnet Data ...
7	2020-09-23 08:13:53.5458711…	192.168.53.99	10.0.0.2	TCP	52	59560 → 23 [ACK] Seq=8819834
8	2020-09-23 08:13:53.5470098…	10.0.0.2	192.168.53.99	TELNET	91	Telnet Data ...
9	2020-09-23 08:13:53.5623655…	192.168.53.99	10.0.0.2	TCP	52	59560 → 23 [ACK] Seq=8819834
10	2020-09-23 08:13:53.5678813…	192.168.53.99	10.0.0.2	TELNET	127	Telnet Data ...
11	2020-09-23 08:13:53.5698839…	10.0.0.2	192.168.53.99	TELNET	55	Telnet Data ...
12	2020-09-23 08:13:53.5827854…	192.168.53.99	10.0.0.2	TELNET	55	Telnet Data ...
13	2020-09-23 08:13:53.5835869…	10.0.0.2	192.168.53.99	TELNET	55	Telnet Data ...
14	2020-09-23 08:13:53.5946443…	192.168.53.99	10.0.0.2	TELNET	55	Telnet Data ...
15	2020-09-23 08:13:53.5953382…	10.0.0.2	192.168.53.99	TELNET	82	Telnet Data ...

Task6: Tunning-Breaking Experiment

首先让 U 与 V 建立 telnet 连接

```
[09/23/20]seed@VM:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN gro
t qlen 1
    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
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast s
OWN group default qlen 1000
    link/ether 00:0c:29:e1:f5:f1 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.2/24 brd 10.0.0.255 scope global ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::465a:6edc:fb3f:e3d9/64 scope link
       valid_lft forever preferred_lft forever
```

然后关闭客户端的 VPN 程序，此时在 telnet 界面输入任何命令都没有反应(包括回车换行)

```
[09/23/20]seed@VM:~$
[09/23/20]seed@VM:~$
[09/23/20]seed@VM:~$
[09/23/20]seed@VM:~$
[09/23/20]seed@VM:~$
```

重新建立连接，又可以正常输入 telnet 命令了，同时还会出现之前输入过的命令，可能是因为服务端由于连接中断对之前的数据发起了重传

```
[09/23/20]seed@VM:~$
[09/23/20]seed@VM:~$
[09/23/20]seed@VM:~$
[09/23/20]seed@VM:~$ ip a
```

Task7: Routing Experiment on Host V

本实验的要求是将 V 的路由设置为向 192.168.53.0/24 网络的数据包才发往 VPN 服务器，而在 Task5 中，我就是这么设置的，即在 V 中执行下列命令

```
[09/22/20]seed@VM:~$ sudo ip route add 192.168.53.0/24 dev ens33 via 10.0.0.1
```

实验结果与 Task5 Task6 一致。

Task8: Experiment with the TUN IP Address

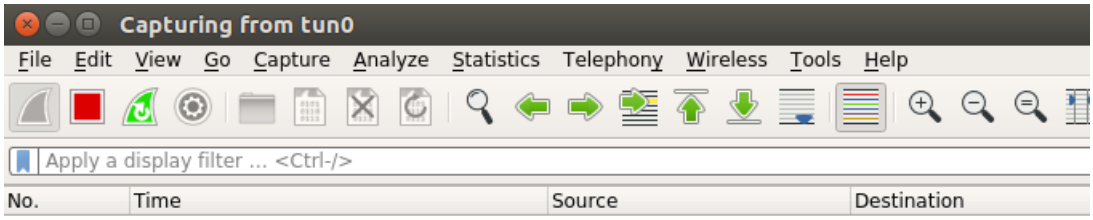
修改客户端 tun 网卡的网络号

```
os.system("ip addr add 192.168.22.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
```

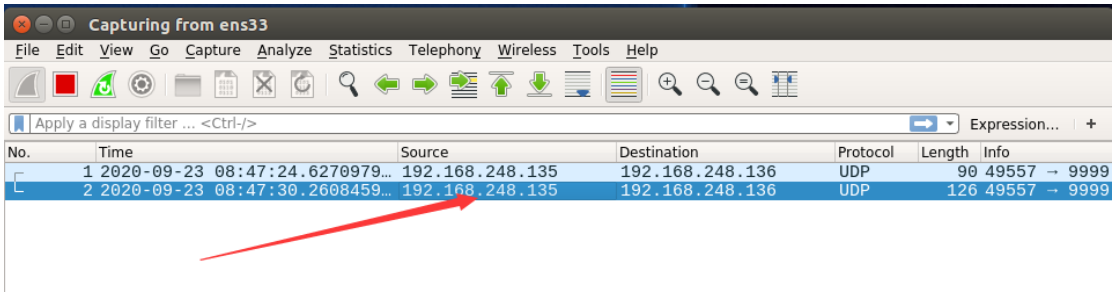
重新运行客户端，并用 U 对 V 发起 ping 请求，此时 ping 命令没有收到回复

```
[09/23/20]seed@VM:~$ ping 10.0.0.2 -c 1
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

[09/23/20]seed@VM:~$
```

监听 G（VPN 服务端）的 tun 网卡，没有收到任何消息



而 G 的 ens33 网卡收到了来自客户端的 UDP 报文，这说明此时 G 的 tun 网卡不能处理 UDP 里的 IP 报文



Task9: Experiment with the TAP Interface

TUN 网卡只能捕捉 IP 层的数据，而 TAP 能捕捉数据链路层的数据。创建 TAP 虚拟网卡和创建 TUN 虚拟网卡的方法相同，只需要将 IFF_TUN 改成 IFF_TAP 即可。
运行如下代码

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

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

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

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

while True:
        packet = os.read(tap, 2048)
        if True:
                ether = Ether(packet)
                ether.show()
```

ping 192.168.53.2

```
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 76:2f:ea:84:6b:31
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = 6
     plen      = 4
     op        = who-has
     hwsrc     = 76:2f:ea:84:6b:31
     psrc      = 192.168.53.99
     hwdst     = 00:00:00:00:00:00
     pdst      = 192.168.53.2
```

可以看到，tap 网卡捕捉到了以太层的数据。