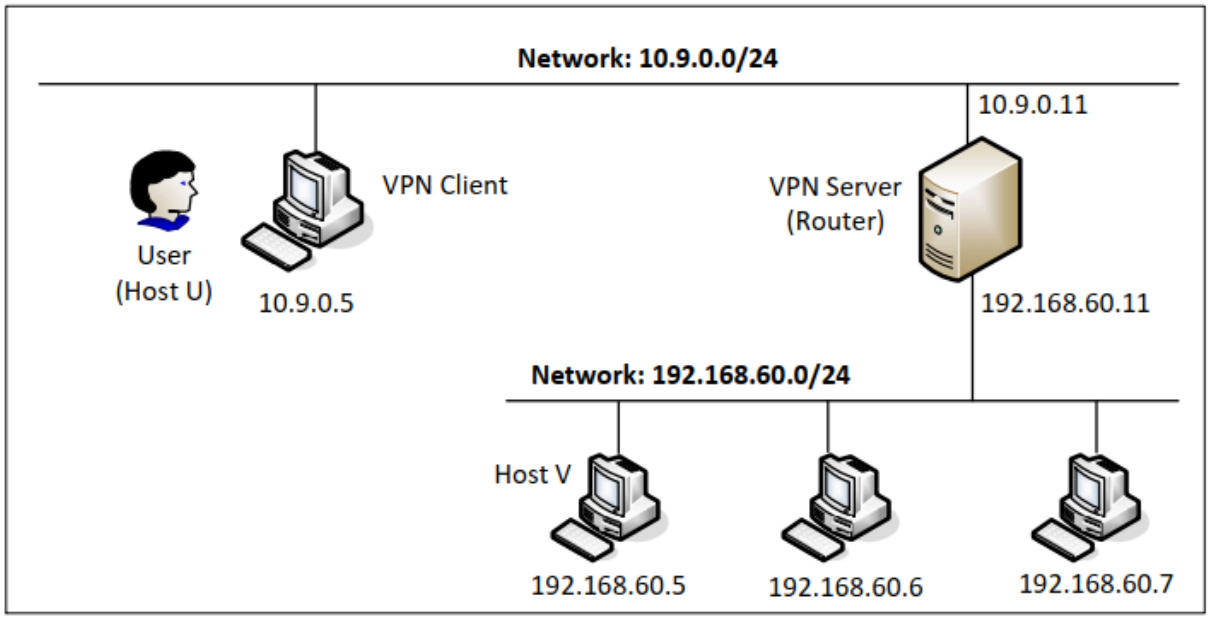
VPN Tunneling Lab

Task1：设置网络



搭建网络，并简单记录一下id

```
[08/09/25]seed@VM:~/.../VPN_tunneling_Lab$ dockps
ef4cbccc72d6  server-router
5e605608ce99  host-192.168.60.6
12c4ee5548b7  client-10.9.0.5
9891c0cf6ae6  host-192.168.60.5
```

按要求进行测试

- U与VPN Server

```
/ U$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=1.31 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.093 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.134 ms
^C
--- 10.9.0.11 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2034ms
rtt min/avg/max/mdev = 0.093/0.511/1.307/0.562 ms
```

- Server与V

```
/ Server$ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=64 time=0.429 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=64 time=0.103 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=64 time=0.068 ms
^C
--- 192.168.60.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2054ms
rtt min/avg/max/mdev = 0.068/0.200/0.429/0.162 ms
/ Server$
```

- U不能与V

```
/ U$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 ---
6 packets transmitted, 0 received, 100% packet loss, time 5128ms
```

- 路由器上运行tcpdump（这个省略，习惯用wireshark）

可见实验环境设置正确

Task2：创建和配置TUN接口

Task2.A：接口名称

在主机U上运行tun.py程序

```
// 使程序可执行
# chmod a+x tun.py
// 使用root权限运行它
# tun.py
```

```
root@12c4ee5548b7:/# 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
2: tun0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group defa
ult qlen 500
    link/none
6: eth0@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP g
roup default
    link/ether 02:42:0a:09:00:05 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.9.0.5/24 brd 10.9.0.255 scope global eth0
       valid_lft forever preferred_lft forever
root@12c4ee5548b7:/#
```

可以看到接口tun0

接下来让我们修改程序

其实就只要改一行

```
before: ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
now: ifr = struct.pack('16sH', b'yuan%d', IFF_TUN | IFF_NO_PI)
```

运行后再次查看ip地址

```
root@12c4ee5548b7:/# 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
3: yuan0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group def
ault qlen 500
    link/none
6: eth0@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP g
roup default
    link/ether 02:42:0a:09:00:05 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.9.0.5/24 brd 10.9.0.255 scope global eth0
       valid_lft forever preferred_lft forever
root@12c4ee5548b7:/#
```

可见名称修改成功

Task2.B：设置TUN接口

在代码中添加两行，配置一下接口

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

再次查看，可见现在配置了IP地址，并打开了接口(state不再是DOWN)

```
5: yuan0: <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 yuan0
       valid_lft forever preferred_lft forever
```

Task2.C：从TUN接口读取数据包

将以下循环替换源代码中的循环

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

在主机U上运行修改后的程序，在主机U上进行进一步实验

- ping一下192.168.53.0/24网络中的主机

```
root@12c4ee5548b7:/# 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 ---
8 packets transmitted, 0 received, 100% packet loss, time 7147ms

root@12c4ee5548b7:/#
```

```
/volumes U$tun.py
Interface Name: yuan0
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
```

可以看到tun接口可以接受数据包，但不会返回任何内容

- ping一下内部网络192.168.60.0/24中的主机

```
root@12c4ee5548b7:/# ping 192.168.60.1
PING 192.168.60.1 (192.168.60.1) 56(84) bytes of data.
64 bytes from 192.168.60.1: icmp_seq=1 ttl=64 time=0.067 ms
64 bytes from 192.168.60.1: icmp_seq=2 ttl=64 time=0.093 ms
64 bytes from 192.168.60.1: icmp_seq=3 ttl=64 time=0.112 ms
64 bytes from 192.168.60.1: icmp_seq=4 ttl=64 time=0.080 ms
64 bytes from 192.168.60.1: icmp_seq=5 ttl=64 time=0.065 ms
64 bytes from 192.168.60.1: icmp_seq=6 ttl=64 time=0.069 ms
^C
--- 192.168.60.1 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5104ms
rtt min/avg/max/mdev = 0.065/0.081/0.112/0.016 ms
root@12c4ee5548b7:/#
```

```
/volumes U$tun.py
Interface Name: yuan0
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
```

可以看到ping得通，但是不是经过tun接口，保留的信息还是之前的

Task2.D：把数据包写入TUN接口

按要求修改代码，要求如下：

- 从 TUN 接口得到一个数据包后，如果这个包是 ICMP echo request 数据包，则构造一个对应的echo reply 数据包写入到 TUN 接口。请证明你的代码按预期工作了。

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'yuan%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))

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

        if ICMP in ip:
            newip = IP(src=ip[IP].dst, dst=ip[IP].src, ihl=ip[IP].ihl)
            newip.ttl = 99
            newicmp = ICMP(type=0, id=ip[ICMP].id, seq=ip[ICMP].seq)
            if ip.haslayer(Raw):
                data = ip[Raw].load
                newpkt = newip/newicmp/data
            else:
                newpkt = newip/newicmp

            os.write(tun, bytes(newpkt))
```

ping一下tun的网络中的主机

可以看到现在可以ping通了

- 不要将 IP 数据包写入接口，而是将一些任意数据写入接口，并报告你的观察结果。

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'yuan%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))

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

        if ICMP in ip:
           os.write(tun, bytes("Hello,world!", encoding='utf-8'))
```

还是ping一下tun的网络中的主机

```
root@12c4ee5548b7:/# 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 ---
4 packets transmitted, 0 received, 100% packet loss, time 3076ms

root@12c4ee5548b7:/#
```

```
Traceback (most recent call last):
  File "./tun.py", line 28, in <module>
    packet = os.read(tun, 2048)
KeyboardInterrupt

/volumes U$tun.py
Interface Name: yuan0
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
```

此时收到报文，但无法返回正确内容

Task3：通过隧道将IP数据包发送到VPN服务器

按要求编写下列代码：

tun_server.py(直接抄源代码即可)

```
#!/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))
```

tun_client.py(在tun.py的基础上修改)

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

import fcntl
import struct
```

```python
import os
import time
from scapy.all import *

#create UDP socket and set the information
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
SERVER_IP, SERVER_PORT = '10.9.0.11', 9090

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'yuan%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))

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

在VPN Server上运行tun_server.py程序，主机U上运行tun_client.py,接下来在主机U上ping一下任何一个属于192.168.53.0/24 网络的 IP 地址

```
PING 192.168.53.1 (192.168.53.1) 56(84) bytes of data.
^C
--- 192.168.53.1 ping statistics ---
9 packets transmitted, 0 received, 100% packet loss, time 8204ms

root@12c4ee5548b7:/#
```

```
/volumes Server$python3 tun_server.py
10.9.0.5:47974 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
10.9.0.5:47974 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
10.9.0.5:47974 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
10.9.0.5:47974 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
10.9.0.5:47974 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
10.9.0.5:47974 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
10.9.0.5:47974 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
```

此时ping失败

如果再尝试Ping一下主机V

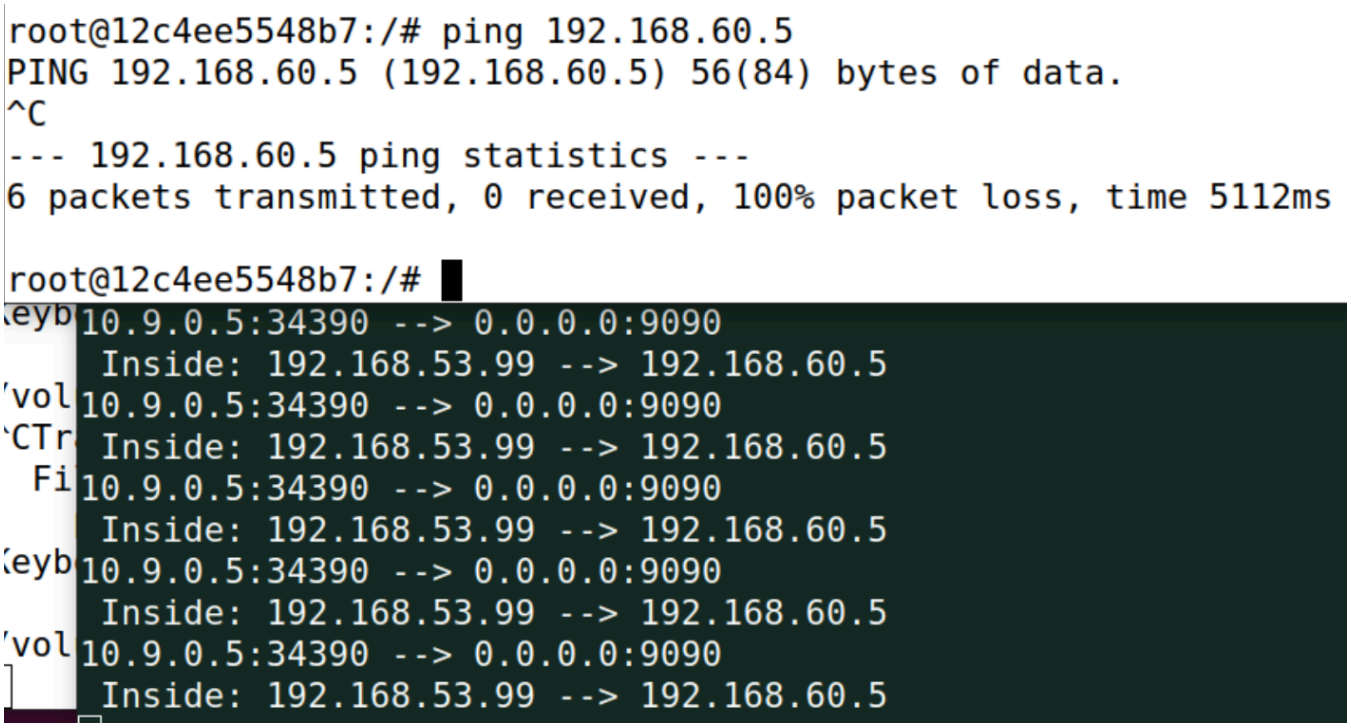
```
root@12c4ee5548b7:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4084ms

root@12c4ee5548b7:/#
```

```
10.9.0.5:47974 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
10.9.0.5:47974 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
10.9.0.5:47974 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
10.9.0.5:47974 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
10.9.0.5:47974 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
```

会发现ICMP数据包根本没有经过隧道，这是因为数据包没有进过tun接口，因此我们要配置一下路由表

```
root@12c4ee5548b7:/# ip route add 192.168.60.0/24 dev yuan0 via 192.168.53.99
root@12c4ee5548b7:/# 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 yuan0 proto kernel scope link src 192.168.53.99
192.168.60.0/24 via 192.168.53.99 dev yuan0
```

再次尝试ping主机V



可见虽然未通，但是已经通过隧道到达。

Task4:设置VPN服务器

接下来我们要修改tun_server.py，使其执行以下操作：

• 创建一个 TUN 接口并对其进行配置。

• 从 socket 接口获取数据，将接收到的数据视为 IP 数据包。

• 将数据包写入 TUN 接口

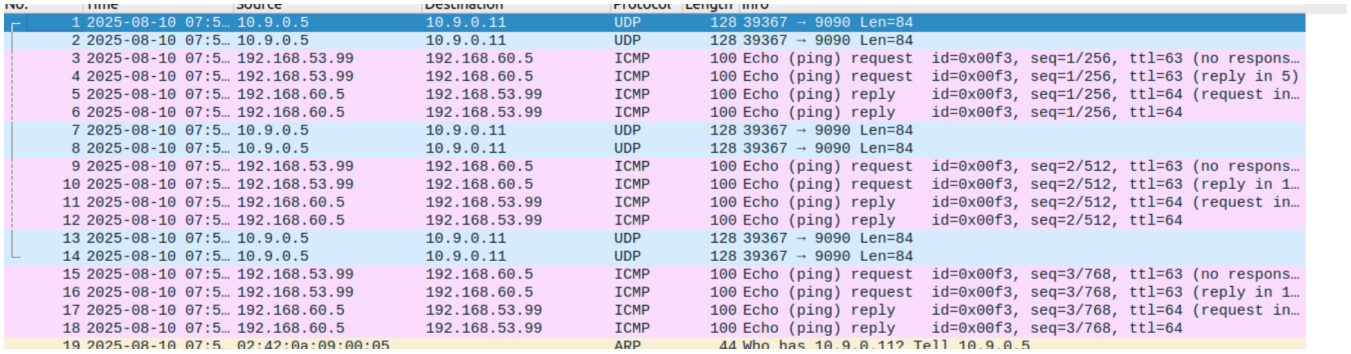
```
#!/usr/bin/env python3

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'yuan%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))

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(pkt.summary())
    os.write(tun, data)
```

对了，还要修改tun_client.py,将配置路由表的语句加入代码

```
os.system("ip route add 192.168.60.0/24 dev{} via 192.168.53.99".format(ifname))
```

此时再从主机U上ping一下主机V



No.	Time	Source	Destination	Protocol	Length	Info
1	2025-08-10 07:5…	10.9.0.5	10.9.0.11	UDP	128	39367 → 9090 Len=84
2	2025-08-10 07:5…	10.9.0.5	10.9.0.11	UDP	128	39367 → 9090 Len=84
3	2025-08-10 07:5…	192.168.53.99	192.168.60.5	ICMP	100	Echo (ping) request id=0x00f3, seq=1/256, ttl=63 (no respons…
4	2025-08-10 07:5…	192.168.53.99	192.168.60.5	ICMP	100	Echo (ping) request id=0x00f3, seq=1/256, ttl=63 (reply in 5)
5	2025-08-10 07:5…	192.168.60.5	192.168.53.99	ICMP	100	Echo (ping) reply id=0x00f3, seq=1/256, ttl=64 (request in…
6	2025-08-10 07:5…	192.168.60.5	192.168.53.99	ICMP	100	Echo (ping) reply id=0x00f3, seq=1/256, ttl=64
7	2025-08-10 07:5…	10.9.0.5	10.9.0.11	UDP	128	39367 → 9090 Len=84
8	2025-08-10 07:5…	10.9.0.5	10.9.0.11	UDP	128	39367 → 9090 Len=84
9	2025-08-10 07:5…	192.168.53.99	192.168.60.5	ICMP	100	Echo (ping) request id=0x00f3, seq=2/512, ttl=63 (no respons…
10	2025-08-10 07:5…	192.168.53.99	192.168.60.5	ICMP	100	Echo (ping) request id=0x00f3, seq=2/512, ttl=63 (reply in 1…
11	2025-08-10 07:5…	192.168.60.5	192.168.53.99	ICMP	100	Echo (ping) reply id=0x00f3, seq=2/512, ttl=64 (request in…
12	2025-08-10 07:5…	192.168.60.5	192.168.53.99	ICMP	100	Echo (ping) reply id=0x00f3, seq=2/512, ttl=64
13	2025-08-10 07:5…	10.9.0.5	10.9.0.11	UDP	128	39367 → 9090 Len=84
14	2025-08-10 07:5…	10.9.0.5	10.9.0.11	UDP	128	39367 → 9090 Len=84
15	2025-08-10 07:5…	192.168.53.99	192.168.60.5	ICMP	100	Echo (ping) request id=0x00f3, seq=3/768, ttl=63 (no respons…
16	2025-08-10 07:5…	192.168.53.99	192.168.60.5	ICMP	100	Echo (ping) request id=0x00f3, seq=3/768, ttl=63 (reply in 1…
17	2025-08-10 07:5…	192.168.60.5	192.168.53.99	ICMP	100	Echo (ping) reply id=0x00f3, seq=3/768, ttl=64 (request in…
18	2025-08-10 07:5…	192.168.60.5	192.168.53.99	ICMP	100	Echo (ping) reply id=0x00f3, seq=3/768, ttl=64
19	2025-08-10 07:5…	02:42:0a:09:00:05		ARP	44	Who has 10.9.0.11? Tell 10.9.0.5

可以看到，虽然暂时没有返回功能，但报文已经正确发送出去并得到了回应

Task5：处理双向流量

接下来我们希望VPN可以处理双向流量，因此要用到名为select的系统调用

修改`tun_server.py`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'yuan%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))

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

    for fd in ready:
        if fd is sock:
            data, (SERVER_IP, SERVER_PORT) = sock.recvfrom(2048)
            pkt = IP(data)
            print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))
            os.write(tun, bytes(pkt))
        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))
```

还要修改tun_client.py

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

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

sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
SERVER_IP, SERVER_PORT = '10.9.0.11', 9090

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'yuan%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 {} via 192.168.53.99".format(ifname))

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

    for fd in ready:
        if fd is sock:
            data, (SERVER_IP, SERVER_PORT) = sock.recvfrom(2048)
            pkt = IP(data)
            print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))
            os.write(tun, bytes(pkt))
        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))
```

让我们在主机U上ping一下主机V

```
root@12c4ee5548b7:/# 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=4.01 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=5.85 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=4.44 ms
^C
--- 192.168.60.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 4.008/4.765/5.851/0.787 ms
root@12c4ee5548b7:/#
```

在服务器上看到：

```
/volumes Server$python3 tun_server.py
Interface Name: yuan0
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.60.5 --> 192.168.53.99
```

在用户端看到：

```
/volumes U$python3 tun_client.py
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
```

再从主机U上telnet到主机V：

```
root@12c4ee5548b7:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
9891c0cf6ae6 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic 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.

seed@9891c0cf6ae6:~$
```

此时一个未加密的VPN通道已经完成了

最后用wireshark抓包U ping V的过程

1	2025-08-10 08:2…	10.9.0.5	10.9.0.11	UDP	128	43936 → 9090 Len=84
2	2025-08-10 08:2…	10.9.0.5	10.9.0.11	UDP	128	43936 → 9090 Len=84
3	2025-08-10 08:2…	192.168.53.99	192.168.60.5	ICMP	100	Echo (ping) request id=0x0101, seq=1/256, ttl=63 (no respons…
4	2025-08-10 08:2…	192.168.53.99	192.168.60.5	ICMP	100	Echo (ping) request id=0x0101, seq=1/256, ttl=63 (no respons…
5	2025-08-10 08:2…	192.168.53.99	192.168.60.5	ICMP	100	Echo (ping) request id=0x0101, seq=1/256, ttl=63 (reply in 6)
6	2025-08-10 08:2…	192.168.60.5	192.168.53.99	ICMP	100	Echo (ping) reply id=0x0101, seq=1/256, ttl=64 (request in…
7	2025-08-10 08:2…	192.168.60.5	192.168.53.99	ICMP	100	Echo (ping) reply id=0x0101, seq=1/256, ttl=64
8	2025-08-10 08:2…	10.9.0.11	10.9.0.5	UDP	128	9090 → 43936 Len=84
9	2025-08-10 08:2…	10.9.0.11	10.9.0.5	UDP	128	9090 → 43936 Len=84

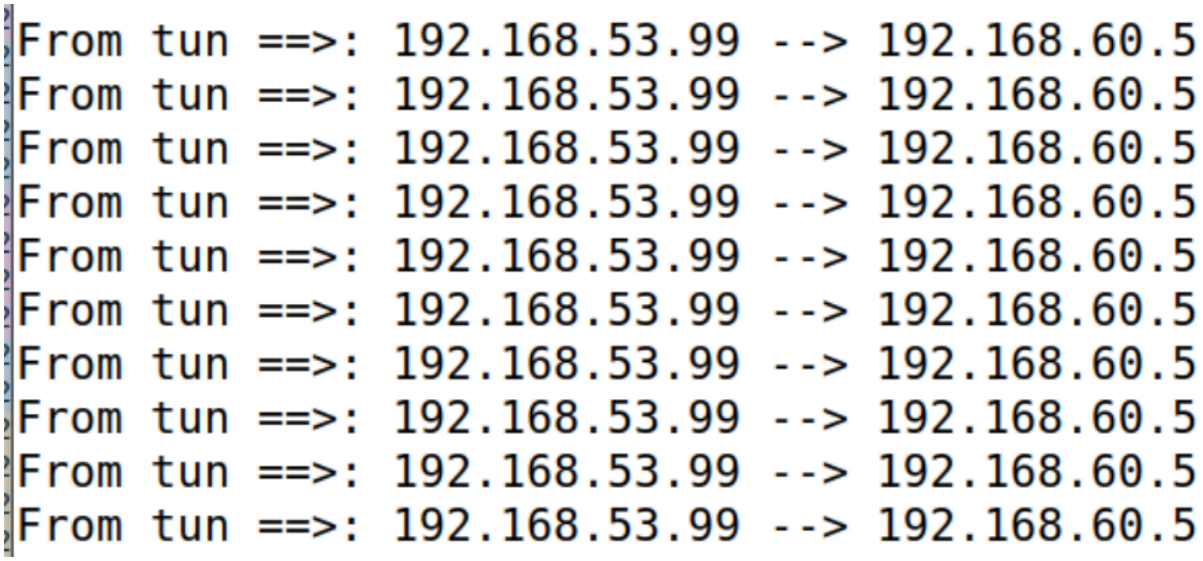
总结一下ping的过程

1. U 发送给 VPN_SERVER(UDP)
2. VPN_SERVER 发送给 V（ICMP_request)
3. V 回复 VPN_SERVER (ICMP_reply)
4. VPN_SERVER 将回复发给 U（UDP）

Task6：隧道中断实验

从U telnet到 V

让我们中断服务器端的程序，此时telnet的窗口内光标不移动，输入无反应，但连接未中断，此时输入后客户端的数据不能发送



让我们迅速重新启动，一旦隧道重新建立，telnet窗口会突然出现之前输入的内容

```
seed@9891c0cf6ae6:~$ hello
-bash: hello: command not found
seed@9891c0cf6ae6:~$ dgfvdgdfdfsdfdsfsdfgfgfghghfhfhdfdfddfdf
```

猜测这是因为这些内容保留在缓存区，一旦重新启动，会再次发送

Task7:主机V上的路由实验

为了模拟真实多跳VPN的情况，我们要手动设置路由表

先查看IP route

```
/ V$ip route
default via 192.168.60.11 dev eth0
192.168.60.0/24 dev eth0 proto kernel scope link src 192.168.60.5
```

然后删除默认条目

```
/ V$ip route del default
/ V$ip route
192.168.60.0/24 dev eth0 proto kernel scope link src 192.168.60.5
```

最后新增条目

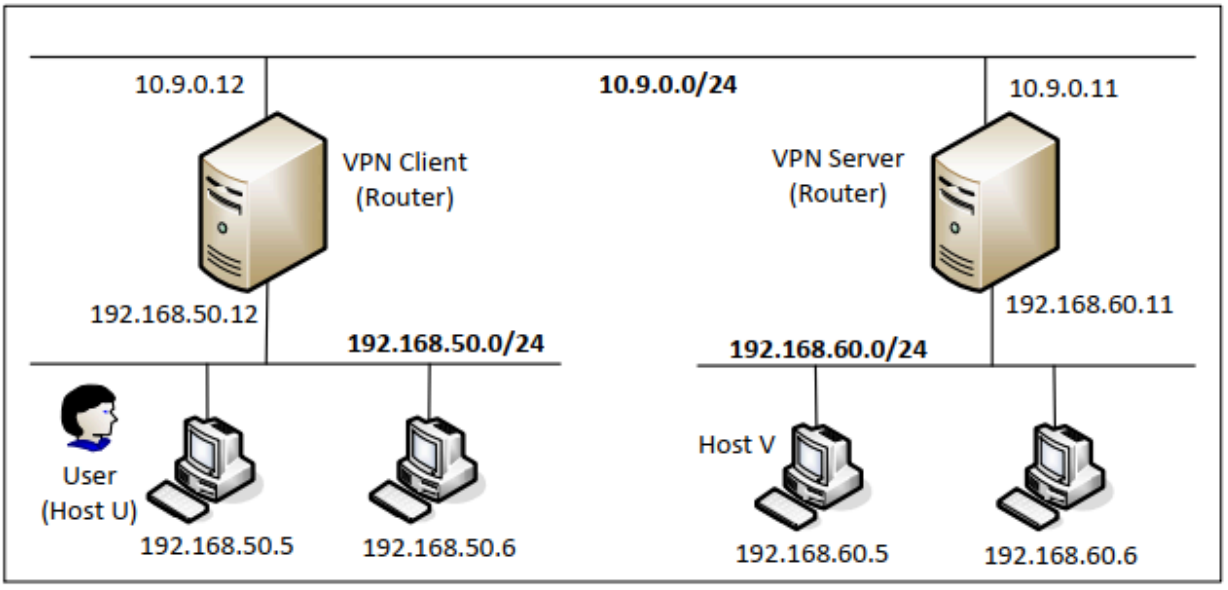
```
/ V$ip route add 192.168.53.0/24 via 192.168.60.11
/ V$ip route
192.168.53.0/24 via 192.168.60.11 dev eth0
192.168.60.0/24 dev eth0 proto kernel scope link src 192.168.60.5
```

再次尝试从U ping V，发现依然可以成功

```
root@12c4ee5548b7:/# 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=4.54 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=5.63 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=6.78 ms
^C
--- 192.168.60.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 4.538/5.647/6.780/0.915 ms
```

Task8：专用网络之间的VPN

重新建立新的网络设置




```
$ docker-compose -f docker-compose2.yml build
$ docker-compose -f docker-compose2.yml up
```

记录一下ID

```
[08/10/25]seed@VM:~/.../volumes$ dockps
af3308506534  host-192.168.60.5
23e4410bc445  client-10.9.0.5
2a3bc21aa701  host-192.168.60.6
61204f7d29ba  host-192.168.50.5
dac6cf9c3d6b  server-router
e9a01c6816fd  host-192.168.50.6
```

这里的客户端不需要改，让我们修改一下服务器端

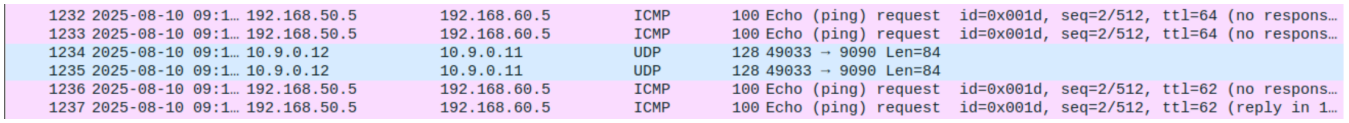
```
#!/usr/bin/env python3

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'yuan%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))

os.system("ip route add 192.168.50.0/24 dev {}".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:
    # this will block until at least one interface is ready
    ready, _, _ = select.select([sock, tun], [], [])

    for fd in ready:
        if fd is sock:
            data, (SERVER_IP, SERVER_PORT) = sock.recvfrom(2048)
            pkt = IP(data)
            print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))
            os.write(tun, bytes(pkt))
        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))
```

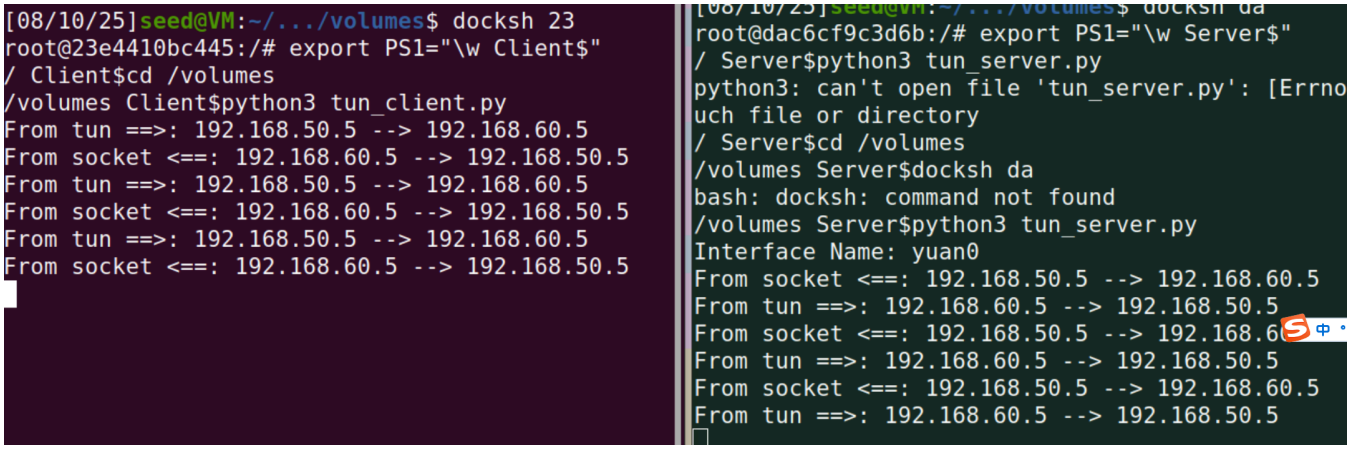
可以看到上述修改的仅仅是更新了路由表，提示要去192.168.50.0/24网络得从接口走

让我们在主机U上尝试ping一下V

```
/ U$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=62 time=4.69 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=62 time=4.10 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=62 time=4.80 ms
^C
--- 192.168.60.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 4.099/4.529/4.799/0.307 ms
```

再配上wireshark的抓包图和客户端、服务器端的图片





可以看到确实通过了VPN隧道

Task9：用TAP接口实验

修改tun.py为tap.py

```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
tap = os.open("/dev/net/tun", os.O_RDWR)
```

```
ifr = struct.pack('16sH', b'yuan%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:
    # Get a packet from the tun interface
    packet = os.read(tap, 2048)
    if packet:
        ether = Ether(packet)
        print(ether.summary())
```

再ping一下192.168.53.0/24 网络中的一个 IP 地址

```
/volumes Server$python3 tap.py
Interface Name: yuan0
Ether / ARP who has 192.168.53.1 says 192.168.53.99
Ether / ARP who has 192.168.53.1 says 192.168.53.99
Ether / ARP who has 192.168.53.1 says 192.168.53.99
```

可以看到不断广播ARP请求

接下来在进一步修改代码

```
#!/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
tap = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'yuan%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 packet:
        print("--------------------------------")
        ether = Ether(packet)
        print(ether.summary())

        # Send a spoofed ARP response
        FAKE_MAC = "aa:bb:cc:dd:ee:ff"
        if ARP in ether and ether[ARP].op == 1:
            arp = ether[ARP]
            newether = Ether(dst=ether.src, src=FAKE_MAC)
            newarp = ARP(psrc=arp.pdst, hwsrc=FAKE_MAC, pdst=arp.psrc,hwdst=ether.src, op=2)
            newpkt = newether/newarp

            print("***** Fake response: {}".format(newpkt.summary()))
            os.write(tap, bytes(newpkt))
```

在服务器上输入`arping -I yuan0 192.168.53.33`，

```
/volumes Server$python3 tap.py
Interface Name: yuan0
--------------------------------
Ether / ARP who has 192.168.53.33 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 192.168.53.33
--------------------------------
Ether / ARP who has 192.168.53.33 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 192
--------------------------------
Ether / ARP who has 192.168.53.33 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 192.168.53.33
```

在服务器上输入`arping -I yuan0 1.2.3.4`，

```
--------------------------------
Ether / ARP who has 1.2.3.4 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 1.2.3.4
--------------------------------
Ether / ARP who has 1.2.3.4 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 1.2.3.4
--------------------------------
Ether / ARP who has 1.2.3.4 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 1.2.3.4
--------------------------------
Ether / ARP who has 1.2.3.4 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 1.2.3.4
--------------------------------
Ether / ARP who has 1.2.3.4 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 1.2.3.4
```

可以看到确实收到了伪造的回复

Over！！！！！