

Task1:

将 SEED1 设置为 NAT 网络(host U), 地址 10.0.2.11; 将 SEED2 设置两个网卡 (VPN server), 一个 NAT 网络, 地址 10.0.2.10, 一个内部网络, 地址 192.168.33.1; 将 SEED3 设置为内部网络 (host V), 地址 192.168.33.101。

此时用 U 去连接 server, 能 ping 通:

```
[09/22/20]seed@VM:~$ ping 10.0.2.10
PING 10.0.2.10 (10.0.2.10) 56(84) bytes of data.
64 bytes from 10.0.2.10: icmp_seq=1 ttl=64 time=1.20 ms
64 bytes from 10.0.2.10: icmp_seq=2 ttl=64 time=1.12 ms
```

用 server 去连接 V, 能 ping 通:

```
[09/22/20]seed@VM:~$ ping 192.168.33.101
PING 192.168.33.101 (192.168.33.101) 56(84) bytes of data.
64 bytes from 192.168.33.101: icmp_seq=1 ttl=64 time=3.31 ms
64 bytes from 192.168.33.101: icmp_seq=2 ttl=64 time=1.15 ms
64 bytes from 192.168.33.101: icmp_seq=3 ttl=64 time=0.788 ms
```

用 U 直接连接 V, 不能 ping 通:

```
[09/22/20]seed@VM:~$ ping 192.168.33.101
PING 192.168.33.101 (192.168.33.101) 56(84) bytes of data.
```

Task2.a:

```
[09/22/20]seed@VM:~$ sudo python3 tun.py
Interface Name: tun0
```

打开另一个 terminal 查看接口, 可以看到 tun0 接口:

```
[09/22/20]seed@VM:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default 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: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo fast state UP group default qlen 1000
    link/ether 08:00:27:ec:af:fa brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.11/24 brd 10.0.2.255 scope global dynamic enp0s3
        valid_lft 994sec preferred_lft 994sec
    inet6 fe80::4047:69c2:85a6:87ae/64 scope link
        valid_lft forever preferred_lft forever
3: tun0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 500
    link/none
```

改变程序端口名成 hui，可以看到再次运行端口名已经改变：

```
[09/22/20]seed@VM:~$ sudo python3 tun.py
Interface Name: hui0
```

Task2.b:

激活 hui5 接口，可以看到再次 ip address 命令时该接口已经被绑定上了 ip:

```
[09/22/20]seed@VM:~$ sudo ip addr add 192.168.53.99/24
dev hui5
[09/22/20]seed@VM:~$ sudo ip link set dev hui5 up

9: hui5: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu
1500 qdisc pfifo_fast state UNKNOWN group default qlen
500
    link/none
    inet 192.168.53.99/24 scope global hui5
        valid_lft forever preferred_lft forever
    inet6 fe80::3684:62b3:c05f:fb6b/64 scope link flags
800
        valid_lft forever preferred_lft forever
```

Task2.c:

在 U 上激活一个接口，使用 ip address 查看看到该接口也被绑定上了 ip 地址:

```
3: hui: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1
500 qdisc pfifo_fast state UNKNOWN group default qlen 5
00
    link/none
    inet 192.168.53.99/24 scope global hui
        valid_lft forever preferred_lft forever
    inet6 fe80::209d:35b0:bc69:2c90/64 scope link flags
800
        valid_lft forever preferred_lft forever
```

在该机器上 ping 192.168.53.5，发现在接口上打印出了 ping 报文且持续打印:

```
###[ IP ]###
version    = 4
ihl        = 5
tos        = 0x0
len        = 84
id         = 32903
flags      = DF
frag       = 0
ttl        = 64
proto      = icmp
chksum     = 0xce68
src        = 192.168.53.99
dst        = 192.168.53.5
\options   \
###[ ICMP ]###
type       = echo-request
code       = 0
chksum     = 0xf792
id         = 0x9bc
seq        = 0x17
###[ Raw ]###
load       = '\x9e\x05j_\xfc1\x07\x00\x08\t\n\x0
b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19
```

在该机器上 ping server 和 V 的内部网段地址 192.168.33.10，发现此时接口中就没有再输出此时的报文。

在 ping 的过程中虽然没有 ping 通，但是接口输出了主机发出的 ping 报文包。综合以上结果，该接口只能捕获接口网段中传输的报文，接口网段以外的报文无法捕获。

Task2.d:

修改 tun.py 的内容，使得其在捕获一个报文后修改其 ip 包发送地址为 1.2.3.4 再发出：

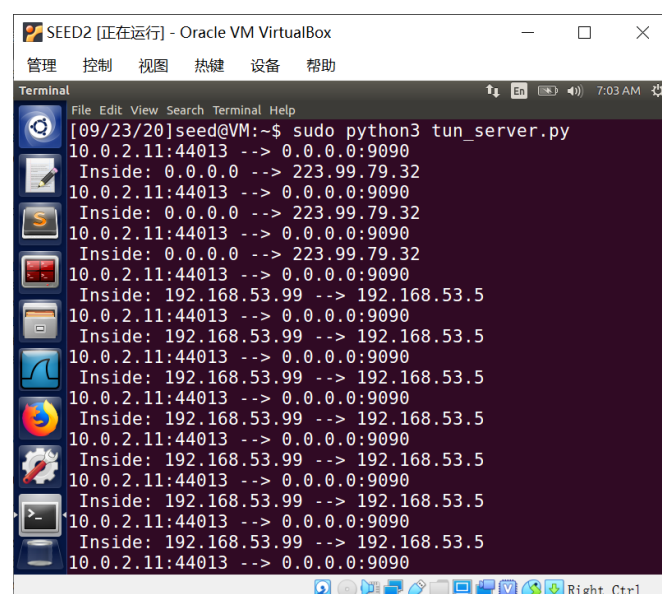
```
while True:
    packet = os.read(tun, 2048)
    if True:
        ip=IP(packet)
        newip=IP(src='1.2.3.4',dst=ip.src)
        newpkt=newip/ip.payload
        os.write(tun,bytes(newpkt))
```

运行该程序，同时在另一个终端上 ping 192.168.53.1，打开 wireshark 监听 hui 接口，可以看到接口确实接受到报文之后又发出了报文：

20-09-23 04:53:06.8080343...	192.168.53.99	192.168.53.1	ICMP
20-09-23 04:53:06.8309422...	1.2.3.4	192.168.53.99	ICMP
20-09-23 04:53:07.8388404...	192.168.53.99	192.168.53.1	ICMP
20-09-23 04:53:07.8474755...	1.2.3.4	192.168.53.99	ICMP
20-09-23 04:53:08.8625702...	192.168.53.99	192.168.53.1	ICMP
20-09-23 04:53:08.8696875...	1.2.3.4	192.168.53.99	ICMP
20-09-23 04:53:09.8865448...	192.168.53.99	192.168.53.1	ICMP
20-09-23 04:53:09.9012508...	1.2.3.4	192.168.53.99	ICMP
20-09-23 04:53:10.9101243...	192.168.53.99	192.168.53.1	ICMP
20-09-23 04:53:10.9230552...	1.2.3.4	192.168.53.99	ICMP
20-09-23 04:53:11.9340392...	192.168.53.99	192.168.53.1	ICMP
20-09-23 04:53:11.9449885...	1.2.3.4	192.168.53.99	ICMP

Task3:

运行 tun_server.py 以及 tun_client.py，在 U 上 ping 192.168.53.5，观察 server 上发现收到了发来的包，因为在 tun_client.py 中设置了将接口收到的包转发到 10.0.2.10 地址：



再使用 U 直接 ping 192.168.33.101 发现此时 server 上并没有收到数据包，因为 ping 的报文并没有经过 192.168.53.0/24 这个网段，所以接口无法将报文转发给 server。
在 U 的路由表里添加：

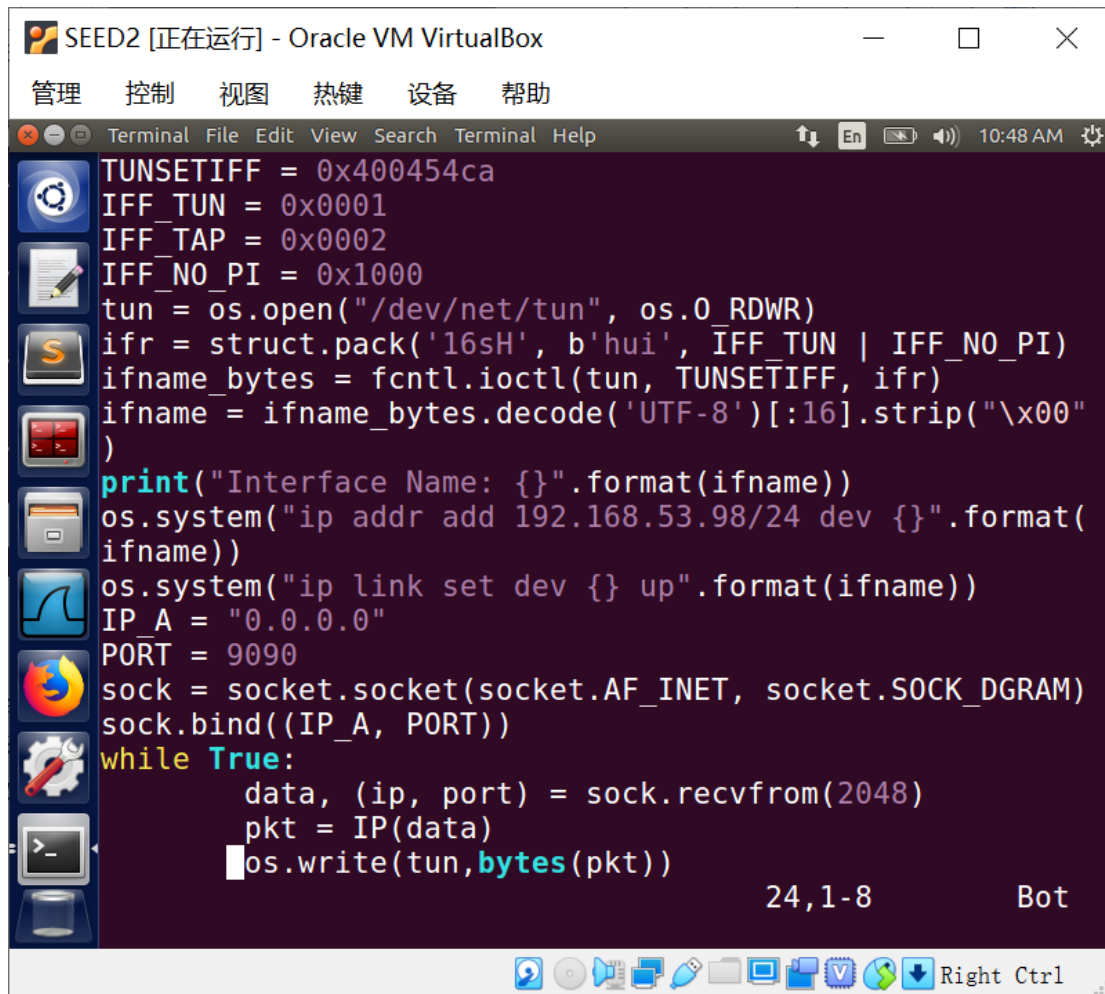
```
[09/23/20]seed@VM:~$ sudo ip route add 192.168.33.0/24  
dev hui via 192.168.53.99
```

再次在 U 上 ping 192.168.33.101，发现此时 server 上已经可以收到报文：

```
[09/23/20]seed@VM:~$ sudo python3 tun_server.py  
10.0.2.11:44013 --> 0.0.0.0:9090  
  Inside: 192.168.53.99 --> 192.168.33.101  
10.0.2.11:44013 --> 0.0.0.0:9090  
  Inside: 192.168.53.99 --> 192.168.33.101  
10.0.2.11:44013 --> 0.0.0.0:9090  
  Inside: 192.168.53.99 --> 192.168.33.101  
10.0.2.11:44013 --> 0.0.0.0:9090  
  Inside: 192.168.53.99 --> 192.168.33.101  
10.0.2.11:44013 --> 0.0.0.0:9090  
  Inside: 192.168.53.99 --> 192.168.33.101  
10.0.2.11:44013 --> 0.0.0.0:9090  
  Inside: 192.168.53.99 --> 192.168.33.101  
█
```

Task4:

修改 tun_server.py，设置 hui 接口，绑定地址 192.168.53.98，并且将 socket 包解开得到 data 数据即 ping 报文，然后传入 hui 接口：



```
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'hui', 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)
    pkt = IP(data)
    os.write(tun, bytes(pkt))
24,1-8 Bot
```

打开网关配置:

```
[09/23/20]seed@VM:~$ sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
```

打开U的hui端口,从U进行ping V的地址192.168.33.101,在V中wireshark抓包可以看到从U的tun发过来的icmp报文:

1	2020-09-23	10:16:21.8512256...	192.168.53.99	192.168.33.101	ICMP
2	2020-09-23	10:16:21.8512676...	192.168.33.101	192.168.53.99	ICMP
3	2020-09-23	10:16:22.8742168...	192.168.53.99	192.168.33.101	ICMP
4	2020-09-23	10:16:22.8742685...	192.168.33.101	192.168.53.99	ICMP
5	2020-09-23	10:16:23.8995063...	192.168.53.99	192.168.33.101	ICMP
6	2020-09-23	10:16:23.8995852...	192.168.33.101	192.168.53.99	ICMP
7	2020-09-23	10:16:24.9221896...	192.168.53.99	192.168.33.101	ICMP
8	2020-09-23	10:16:24.9222641...	192.168.33.101	192.168.53.99	ICMP
9	2020-09-23	10:16:25.9463750...	192.168.53.99	192.168.33.101	ICMP
0	2020-09-23	10:16:25.9464138...	192.168.33.101	192.168.53.99	ICMP
1	2020-09-23	10:16:26.9699136...	192.168.53.99	192.168.33.101	ICMP

Task5:

首先修改 tun_client.py 以及 tun_server.py, 使得两个程序都能处理 tun 以及 socket:

tun_client.py 片段:


```

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(("0.0.0.0",9090))
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, ("10.0.2.10",9090))

```

tun_server.py 片段:

```

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, 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,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, ("10.0.2.11",9090))

```

接着在 host V 路由表上添加路由, 使得 icmp 响应报文能够传得回去:

```

[09/26/20]seed@VM:~$ sudo ip route add 192.168.53.0/24 via 192.168.33.1

```

在 host U 上 ping host V 的地址, 发现能够收到响应:

```
[09/26/20]seed@VM:~$ ping 192.168.33.101
PING 192.168.33.101 (192.168.33.101) 56(84) bytes of data.
64 bytes from 192.168.33.101: icmp_seq=1 ttl=63 time=22.0 ms
64 bytes from 192.168.33.101: icmp_seq=2 ttl=63 time=11.3 ms
64 bytes from 192.168.33.101: icmp_seq=3 ttl=63 time=11.7 ms
64 bytes from 192.168.33.101: icmp_seq=4 ttl=63 time=11.7 ms
```

查看 host U 上 tun_client 的输出可以发现, ping 报文通过 tun 接口发送给了出去, 并且最终通过 socket 包取回了响应报文:

```
From tun ==>: 192.168.53.99 --> 192.168.33.101
From socket <==: 192.168.33.101 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.33.101
From socket <==: 192.168.33.101 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.33.101
From socket <==: 192.168.33.101 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.33.101
From socket <==: 192.168.33.101 --> 192.168.53.99
```

查看 server 上 tun_server 的输出以及 wireshark 也能发现其做了 U 和 V 通信过程中转发的角色:

```
From tun ==>: 192.168.33.101 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.33.101
From tun ==>: 192.168.33.101 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.33.101
From tun ==>: 192.168.33.101 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.33.101
From tun ==>: 192.168.33.101 --> 192.168.53.99
```

192.168.53.99	192.168.33.101	ICMP	84 Echo (ping) request
192.168.33.101	192.168.53.99	ICMP	84 Echo (ping) reply
192.168.53.99	192.168.33.101	ICMP	84 Echo (ping) request
192.168.33.101	192.168.53.99	ICMP	84 Echo (ping) reply
192.168.53.99	192.168.33.101	ICMP	84 Echo (ping) request
192.168.33.101	192.168.53.99	ICMP	84 Echo (ping) reply
192.168.53.99	192.168.33.101	ICMP	84 Echo (ping) request
192.168.33.101	192.168.53.99	ICMP	84 Echo (ping) reply

综上, 可以总结出整个报文流的传输过程为:

1. Host U 发出 ping 请求构造 icmp 报文, 根据路由表转到 tun 接口封装后, 再经过 socket 封装成外网地址, 发往 server
2. Server 接受到 socket 包并且取出负载部分即 ping request 报文, 然后通过 tun 接口写出到局域网中
3. Host V 接受到 server 转发的 request 报文, 产生 reply 报文, 并且查找路由表后发往网

关 server

4. Server 将 reply 报文进行 socket 封装成外网地址，发往 host U
5. 最后 host U 解开 socket 得到负载，然后解析 reply 报文成功收到 ping 的响应。

Task6:

在 host U 上 telnet host V 以后，我终止 tun_client 程序，然后在 host U 里面输入 ifconfig，发现看不见任何东西。此时是 tun 接口的服务断了，tcp 连接并没有断。然后重新打开 tun_client 服务，再次在 telnet 界面输入 ifconfig，可以发现能够成功显示输出。主要原因是在 telnet 建立时两个机器经过了 tcp 三报文握手建立起连接，断开 tun_client 后只是阻断了 U 和 V 交流的报文通道，实际上 tcp 还没有经过四报文握手还在等待中，只要时间不长 tcp 连接就不会断开。当恢复 tun 之后隧道又变得畅通两个机器可以继续交流。

Task7:

由于我在 task1 中设置内网网段时 192.168.60.0/24 出了些问题，因此最终使用的内网网段为 192.168.33.0/24，所以不需要删除 host V 中的默认路由。

我在 task5 中为了使得 host V 返回的 reply 报文能够顺利找到网关已经在 host V 中设置了网关路由如下：

```
[09/26/20]seed@VM:~$ sudo ip route add 192.168.53.0/24
via 192.168.33.1
```

Task8:

将 host U 上的 tun 接口地址改为 192.168.30.99 后，重启 tun，再次 ping host V 主机地址发现没有收到响应：

```
[09/26/20]seed@VM:~$ ping 192.168.33.101
PING 192.168.33.101 (192.168.33.101) 56(84) bytes of data.
```

此时查看 server 的 wireshark 和 tun_server.py 的输出，发现 server 只接收了来自 host U 的 socket 报文，并没有向 host U 返回 reply 的报文：

[illegible]


```
From socket <==: 192.168.30.99 --> 192.168.33.101
From socket <==: 192.168.30.99 --> 192.168.33.101
From socket <==: 192.168.30.99 --> 192.168.33.101
From socket <==: 192.168.30.99 --> 192.168.33.101
From socket <==: 192.168.30.99 --> 192.168.33.101
From socket <==: 192.168.30.99 --> 192.168.33.101
From socket <==: 192.168.30.99 --> 192.168.33.101
From socket <==: 192.168.30.99 --> 192.168.33.101
From socket <==: 192.168.30.99 --> 192.168.33.101
From socket <==: 192.168.30.99 --> 192.168.33.101
From socket <==: 192.168.30.99 --> 192.168.33.101
From socket <==: 192.168.30.99 --> 192.168.33.101
From socket <==: 192.168.30.99 --> 192.168.33.101
From socket <==: 192.168.30.99 --> 192.168.33.101
From socket <==: 192.168.30.99 --> 192.168.33.101
```

再查看 host V 的 wireshark，发现其实并没有收到来自 host U 的 icmp 报文。

综上可以得出结论：

1. server 有接收到 host U 的 request 报文而 host V 没有收到来自 server 转发的报文，说明包应该是在 server 中被丢弃了。
2. 由于控制变量法，实验 8 相比于前面的实验只改变了 host U 中 tun 接口的地址，因此应该是 tun_client 与 tun_server 的地址不在同一个网段时 tun_server 会拒绝将收到 socket 的负载给 write 出去。
3. 实在不太清楚应该如何解决这个问题

Task9:

将 tun_client 的代码改为 tap 的代码，执行该 tap 接口，并且在该机器上 ping 192.168.53.1，可以看到 tap 接口的输出：

```
###[ Ethernet ]###
  dst      = ff:ff:ff:ff:ff:ff
  src      = 52:66:1d:86:09:3f
  type     = ARP
###[ ARP ]###
  hwtype   = 0x1
  ptype    = IPv4
  hwlen    = 6
  plen     = 4
  op       = who-has
  hwsrc    = 52:66:1d:86:09:3f
  psrc     = 192.168.53.99
  hwdst    = 00:00:00:00:00:00
  pdst     = 192.168.53.1
```

因为 tap 接口是捕获到了数据链路层，因此使用 tap 接口输出其捕获的数据可以看到源宿的 MAC 地址，同时也能解析出上层数据，如报文类型为 ARP 报文，采用 ipv4 协议，源宿 ip 地址。