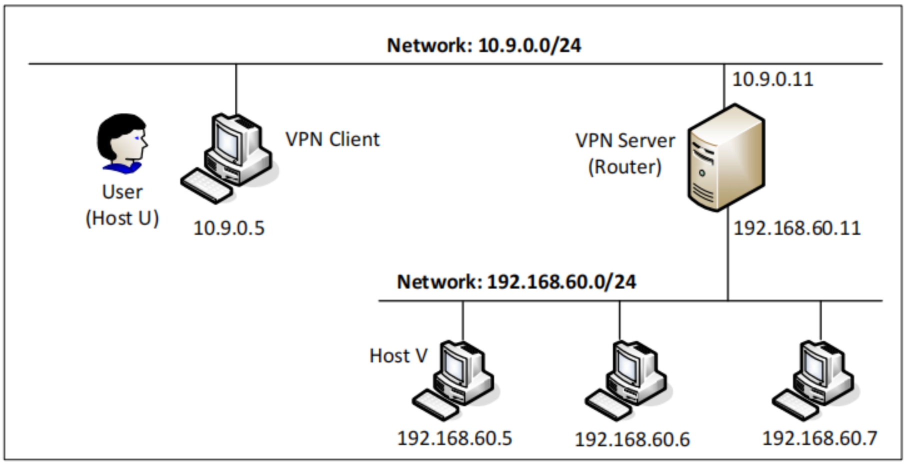
Lab 7: VPN- The Container Version

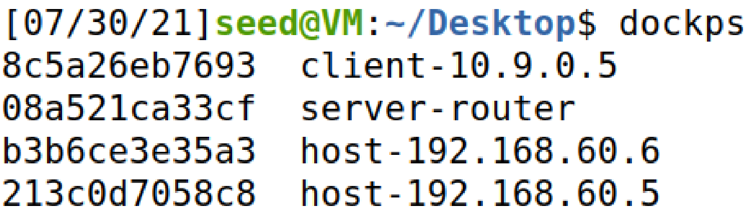
57118101　卞郡菁

准备工作：

一、网络拓扑图



二、容器构建环境

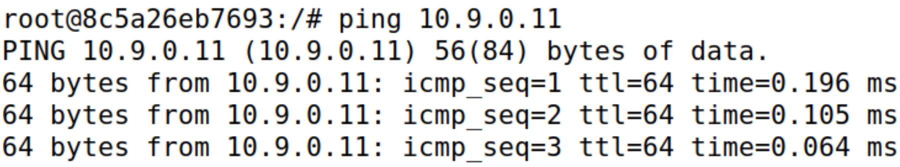


```
[07/30/21]seed@VM:~/Desktop$ dockps
8c5a26eb7693  client-10.9.0.5
08a521ca33cf  server-router
b3b6ce3e35a3  host-192.168.60.6
213c0d7058c8  host-192.168.60.5
```

Task 1: Network Setup

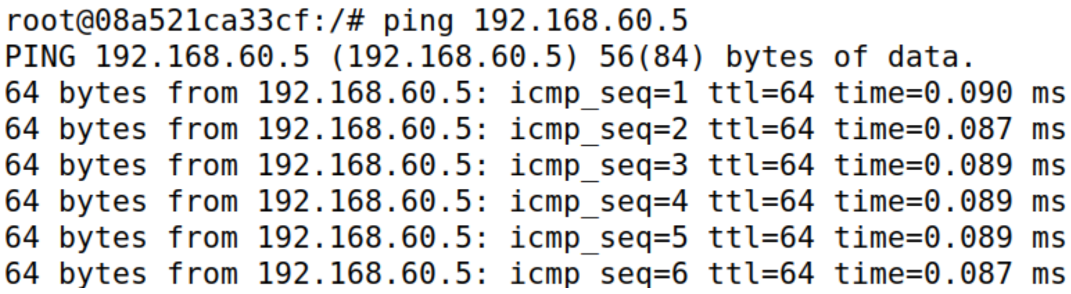
测试三台机器:VPN 客户端、VPN 服务器、主机的相互连通性：

（1）在 host U 上 ping VPN server，两者可以通信：



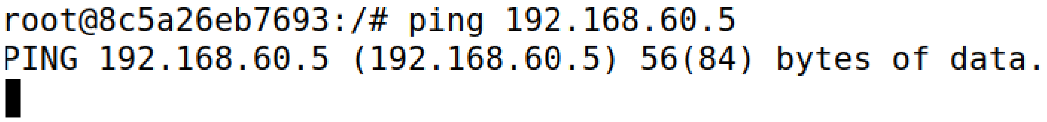
```
root@8c5a26eb7693:/# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.196 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.105 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.064 ms
```

（2）在 VPN server 上 ping host V，两者可以通信：



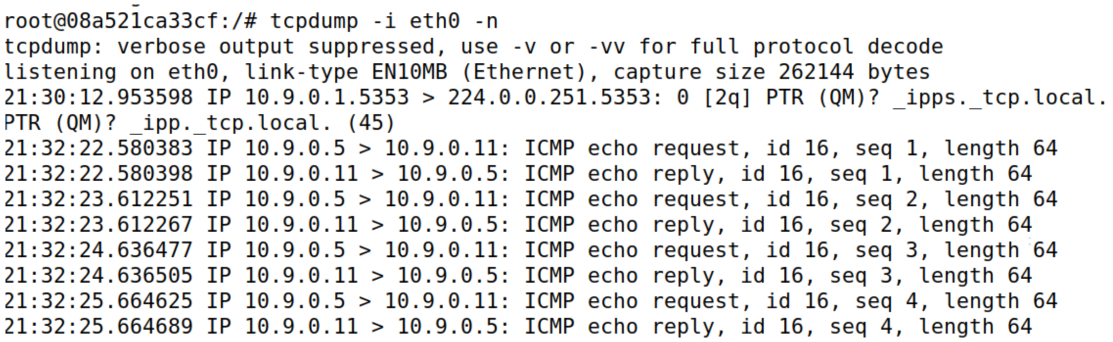
```
root@08a521ca33cf:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=64 time=0.090 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=64 time=0.087 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=64 time=0.089 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=64 time=0.089 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=64 time=0.089 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=64 time=0.087 ms
```

（3）在 host U 上 ping host V，两者不可通信：



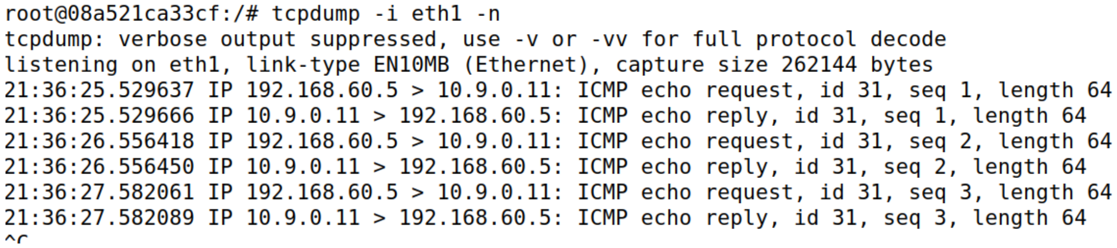
```
root@8c5a26eb7693:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
```

（4）路由器上运行 tcpdump，可监听端口 eth0：



```
root@08a521ca33cf:/# 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
21:30:12.953598 IP 10.9.0.1.5353 > 224.0.0.251.5353: 0 [2q] PTR (QM)? _ipps._tcp.local.
PTR (QM)? _ipp._tcp.local. (45)
21:32:22.580383 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 16, seq 1, length 64
21:32:22.580398 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 16, seq 1, length 64
21:32:23.612251 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 16, seq 2, length 64
21:32:23.612267 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 16, seq 2, length 64
21:32:24.636477 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 16, seq 3, length 64
21:32:24.636505 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 16, seq 3, length 64
21:32:25.664625 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 16, seq 4, length 64
21:32:25.664689 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 16, seq 4, length 64
```

（5）路由器上运行 tcpdump，可监听端口 eth1：

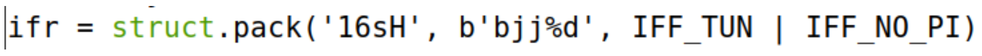


```
root@08a521ca33cf:/# tcpdump -i eth1 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
21:36:25.529637 IP 192.168.60.5 > 10.9.0.11: ICMP echo request, id 31, seq 1, length 64
21:36:25.529666 IP 10.9.0.11 > 192.168.60.5: ICMP echo reply, id 31, seq 1, length 64
21:36:26.556418 IP 192.168.60.5 > 10.9.0.11: ICMP echo request, id 31, seq 2, length 64
21:36:26.556450 IP 10.9.0.11 > 192.168.60.5: ICMP echo reply, id 31, seq 2, length 64
21:36:27.582061 IP 192.168.60.5 > 10.9.0.11: ICMP echo request, id 31, seq 3, length 64
21:36:27.582089 IP 10.9.0.11 > 192.168.60.5: ICMP echo reply, id 31, seq 3, length 64
^C
```

Task 2: Create and Configure TUN Interface--TUN/TAP 技术

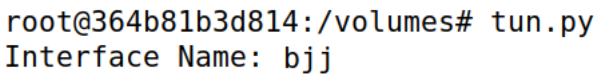
Task 2.A: Name of the Interface

在代码中修改端口名为“bjj”



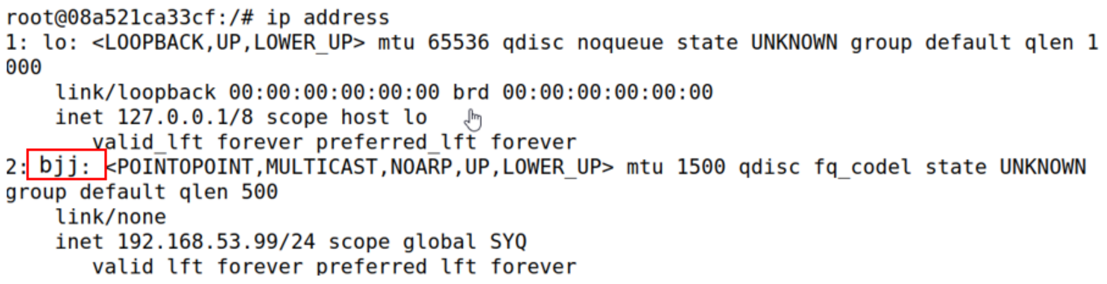
```
ifr = struct.pack('16sH', b'bjj%d', IFF_TUN | IFF_NO_PI)
```

在主机 U 上运行程序 tun.py



```
root@364b81b3d814:/volumes# tun.py
Interface Name: bjj
```

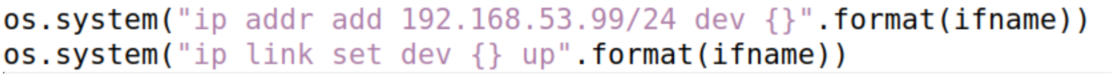
打开另一个终端查看，可以看到存在一个名字为 bjj 的接口



```
root@08a521ca33cf:/# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
000
    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: bjj: <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 SYQ
       valid lft forever preferred lft forever
```

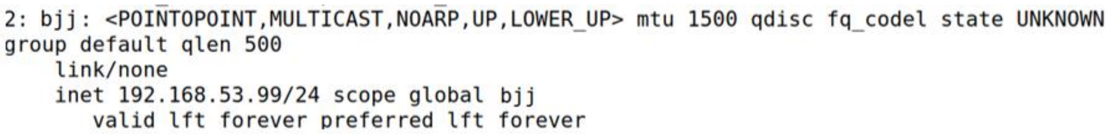
Task 2.B: Set up the TUN Interface

在 tun.py 中添加以下代码，给端口 wang0 自动配置 ip 地址



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

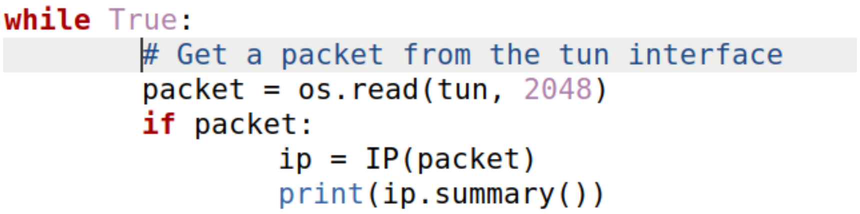
再次在主机 U 上运行程序 tun.py，并另开一个终端输入 ip address 命令，发现此端口已被分配 ip 地址



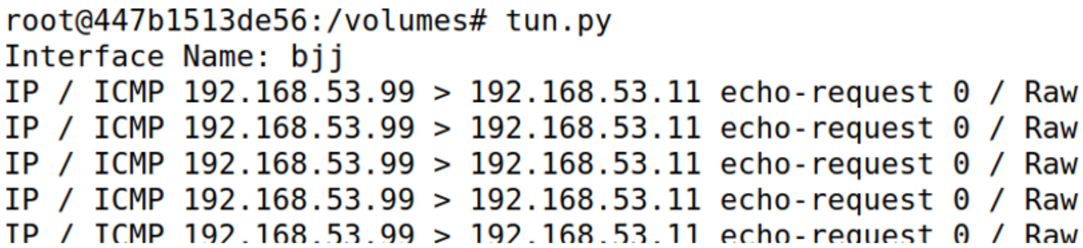
```
2: bjj: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN
group default qlen 500
    link/none
    inet 192.168.53.99/24 scope global bjj
       valid lft forever preferred lft forever
```

Task 2.c: Read from the TUN Interface

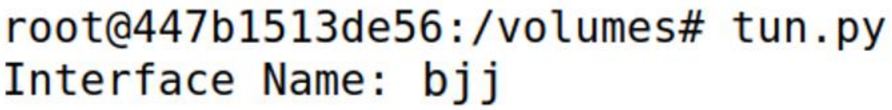
修改程序中的 while 循环如下



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



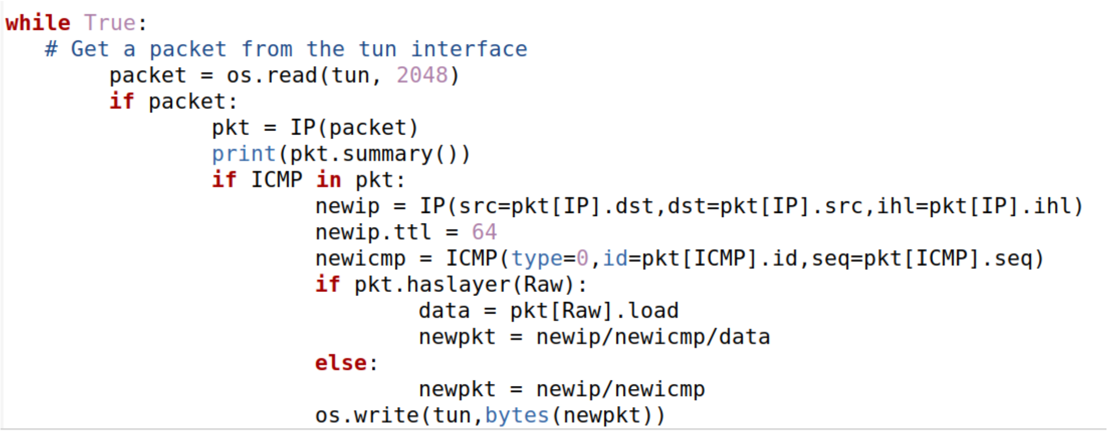
```
root@447b1513de56:/volumes# tun.py
Interface Name: bjj
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
```



```
root@447b1513de56:/volumes# tun.py
Interface Name: bjj
```

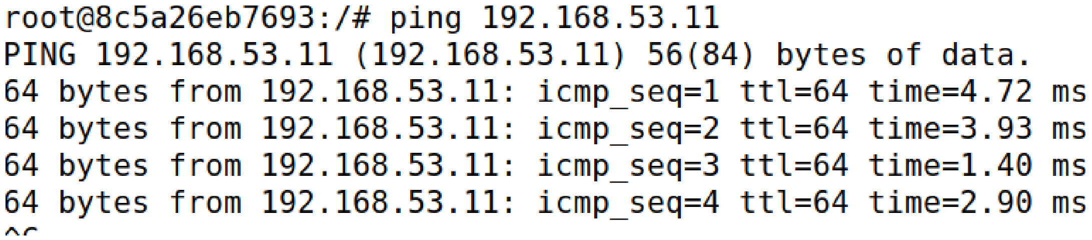
Task 2.D: Write to the TUN Interface

修改 while 循环如下：



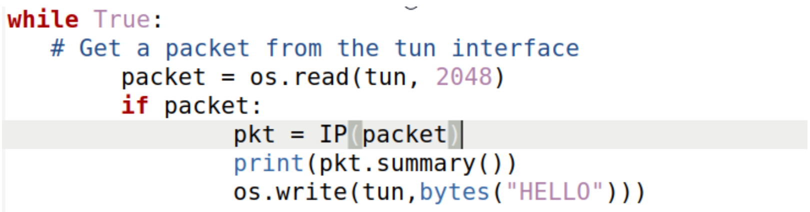
```
while True:
   # Get a packet from the tun interface
        packet = os.read(tun, 2048)
        if packet:
                pkt = IP(packet)
                print(pkt.summary())
                if ICMP in pkt:
                        newip = IP(src=pkt[IP].dst,dst=pkt[IP].src,ihl=pkt[IP].ihl)
                        newip.ttl = 64
                        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))
```

运行程序，然后再次 ping 192.168.53.11，显示可以ping通，说明伪造响应包成功



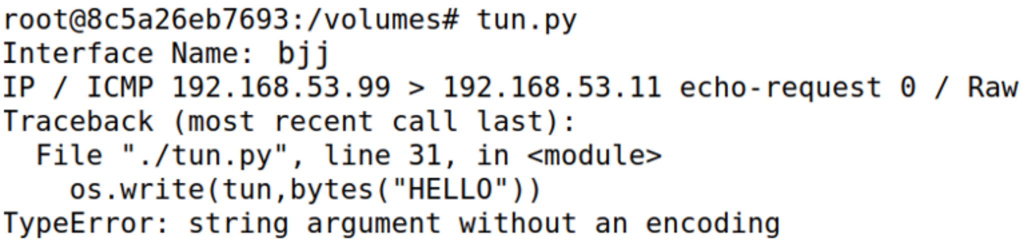
```
root@8c5a26eb7693:/# ping 192.168.53.11
PING 192.168.53.11 (192.168.53.11) 56(84) bytes of data.
64 bytes from 192.168.53.11: icmp_seq=1 ttl=64 time=4.72 ms
64 bytes from 192.168.53.11: icmp_seq=2 ttl=64 time=3.93 ms
64 bytes from 192.168.53.11: icmp_seq=3 ttl=64 time=1.40 ms
64 bytes from 192.168.53.11: icmp_seq=4 ttl=64 time=2.90 ms
^C
```

再次修改while循环



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

运行程序，然后ping 192.168.53.11，显示ping不通

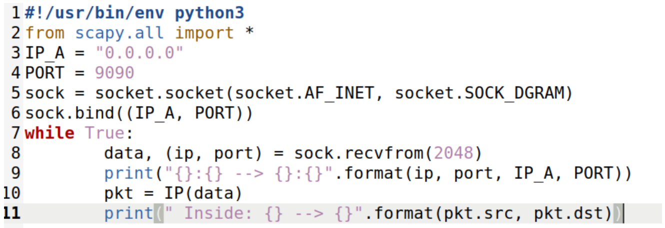


```
root@8c5a26eb7693:/volumes# tun.py
Interface Name: bjj
IP / ICMP 192.168.53.99 > 192.168.53.11 echo-request 0 / Raw
Traceback (most recent call last):
  File "./tun.py", line 31, in <module>
    os.write(tun,bytes("HELLO"))
TypeError: string argument without an encoding
```

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

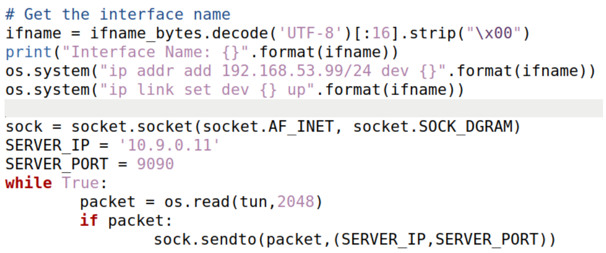
—— IP隧道

tun_server.py代码如下：



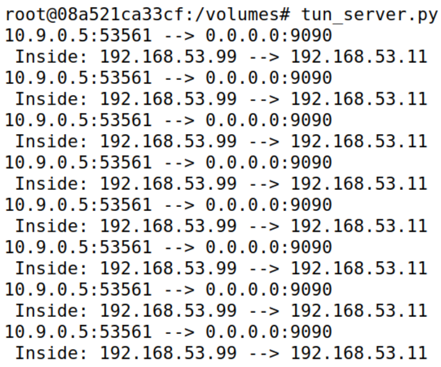
```
#!/usr/bin/env python3
from scapy.all import *
IP_A = "0.0.0.0"
PORT = 9090
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))
while True:
        data, (ip, port) = sock.recvfrom(2048)
        print("{}:{} --> {}:{}".format(ip, port, IP_A, PORT))
        pkt = IP(data)
        print(" Inside: {} --> {}".format(pkt.src, pkt.dst))
```

修改 tun.py 后半部分，编写 tun_client.py:



```
# 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))

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
SERVER_IP = '10.9.0.11'
SERVER_PORT = 9090
while True:
        packet = os.read(tun,2048)
        if packet:
                sock.sendto(packet,(SERVER_IP,SERVER_PORT))
```

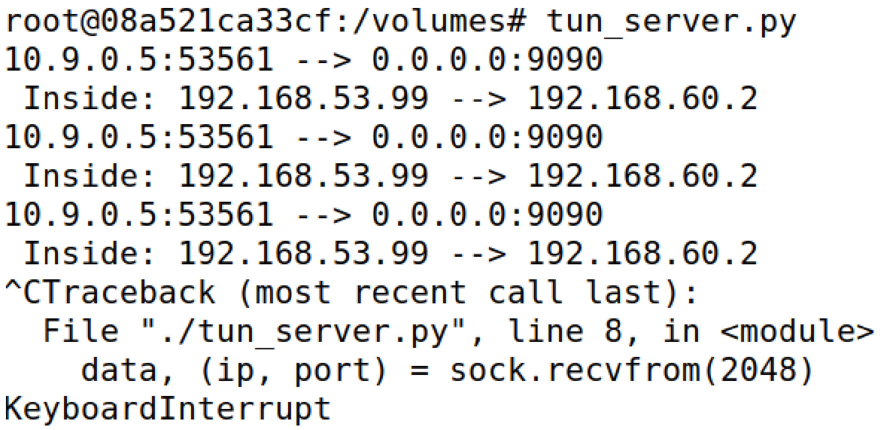
在 host U 上运行 tun_client.py, 在 VPN server 上运行 tun_server.py, 在 host U 上 ping 192.168.53.11:



```
root@08a521ca33cf:/volumes# tun_server.py
10.9.0.5:53561 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.11
10.9.0.5:53561 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.11
10.9.0.5:53561 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.11
10.9.0.5:53561 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.11
10.9.0.5:53561 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.11
10.9.0.5:53561 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.11
10.9.0.5:53561 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.11
10.9.0.5:53561 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.11
```

可以看到 VPN server 中收到报文，但是 ping 192.168.60.0/24 网段的 IP 则没有反应。

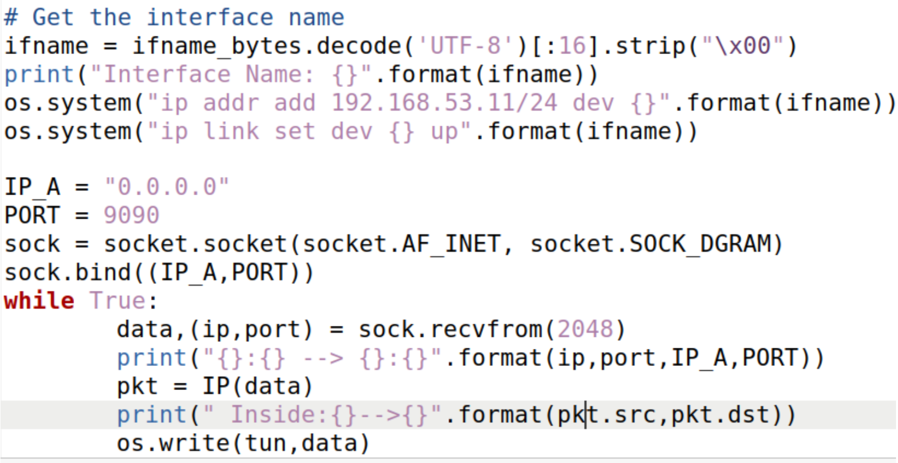
使用 ip route add 192.168.60.0/24 dev bjj 向 host U 增加一条路由，再次 ping192.168.60.0/24 网段中的地址，服务器接收到报文



```
root@08a521ca33cf:/volumes# tun_server.py
10.9.0.5:53561 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.2
10.9.0.5:53561 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.2
10.9.0.5:53561 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.2
^CTraceback (most recent call last):
  File "./tun_server.py", line 8, in <module>
    data, (ip, port) = sock.recvfrom(2048)
KeyboardInterrupt
```

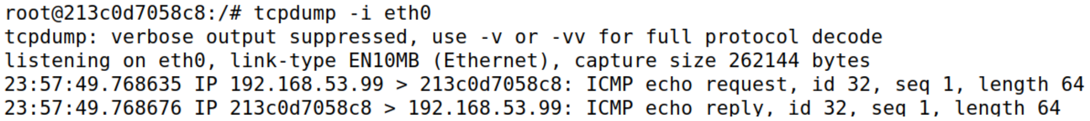
Task4 : Spoofing NS Records for Another Domain

将 tun.py 后半部分进行如下修改，作为 tun_server.py 代码：



```python
# 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))
        os.write(tun,data)
```

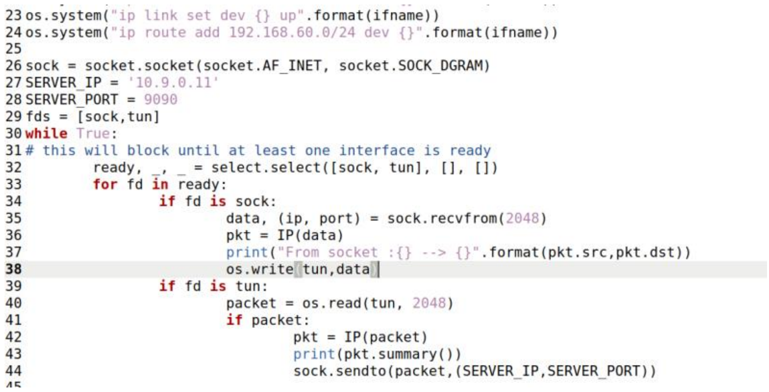
重复上一节操作，运行 server 和 client，在 hostU 上 ping host V，在 host V 上进行 tcp dump 收到相应报文：



```
root@213c0d7058c8:/# tcpdump -i eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
23:57:49.768635 IP 192.168.53.99 > 213c0d7058c8: ICMP echo request, id 32, seq 1, length 64
23:57:49.768676 IP 213c0d7058c8 > 192.168.53.99: ICMP echo reply, id 32, seq 1, length 64
```

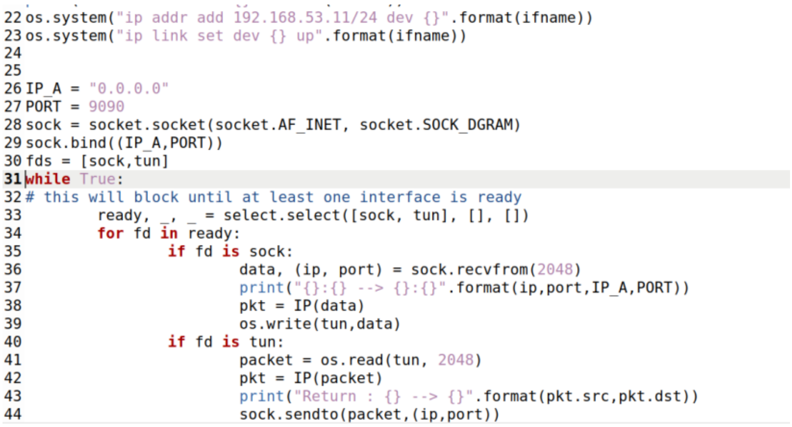
Task 5: Handling Traffic in Both Directions

将 tun_client.py 后半部分进行如下修改：



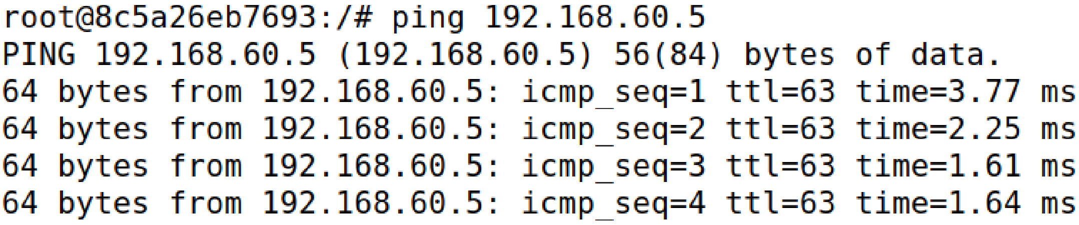
```python
os.system("ip link set dev {} up".format(ifname))
os.system("ip route add 192.168.60.0/24 dev {}".format(ifname))

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([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)
                        if packet:
                                pkt = IP(packet)
                                print(pkt.summary())
                                sock.sendto(packet,(SERVER_IP,SERVER_PORT))
```

将 tun_server.py 后半部分进行如下修改：



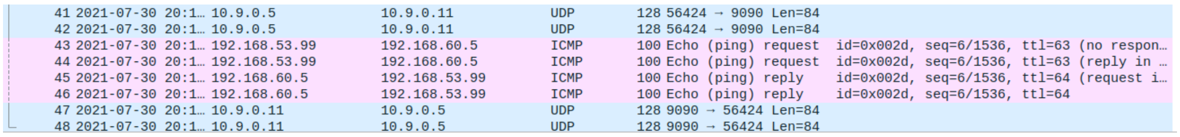
```python
os.system("ip addr add 192.168.53.11/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))


IP_A = "0.0.0.0"
PORT = 9090
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A,PORT))
fds = [sock,tun]
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, (ip, port) = sock.recvfrom(2048)
                        print("{}:{} --> {}:{}".format(ip,port,IP_A,PORT))
                        pkt = IP(data)
                        os.write(tun,data)
                if fd is tun:
                        packet = os.read(tun, 2048)
                        pkt = IP(packet)
                        print("Return : {} --> {}".format(pkt.src,pkt.dst))
                        sock.sendto(packet,(ip,port))
```

运行后可以看到Host U可以ping通Host V:



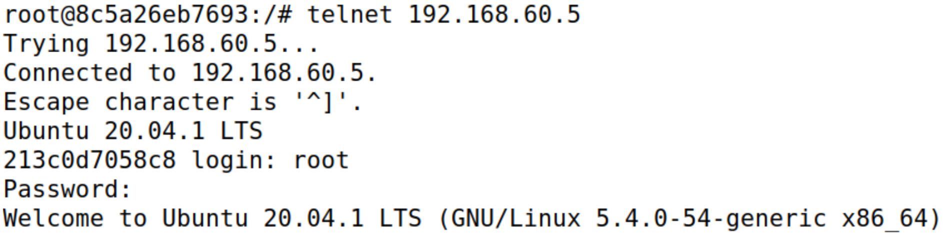
```
root@8c5a26eb7693:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=3.77 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=2.25 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=1.61 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=1.64 ms
```

在wireshark可以详细看到报文的路径:



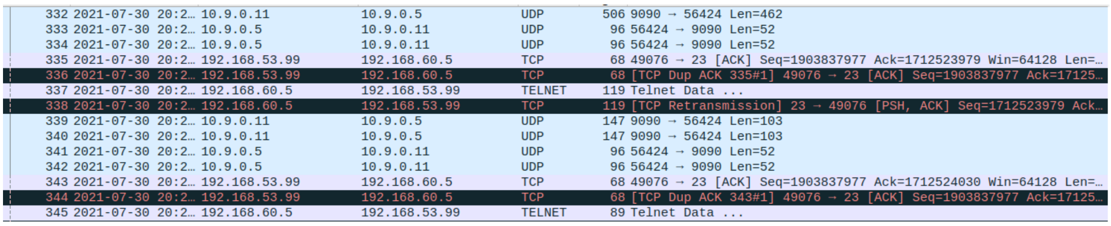
```
41 2021-07-30 20:1… 10.9.0.5       10.9.0.11      UDP    128 56424 → 9090 Len=84
42 2021-07-30 20:1… 10.9.0.5       10.9.0.11      UDP    128 56424 → 9090 Len=84
43 2021-07-30 20:1… 192.168.53.99  192.168.60.5   ICMP   100 Echo (ping) request  id=0x002d, seq=6/1536, ttl=63 (no respon…
44 2021-07-30 20:1… 192.168.53.99  192.168.60.5   ICMP   100 Echo (ping) request  id=0x002d, seq=6/1536, ttl=63 (reply in …
45 2021-07-30 20:1… 192.168.60.5   192.168.53.99  ICMP   100 Echo (ping) reply    id=0x002d, seq=6/1536, ttl=64 (request i…
46 2021-07-30 20:1… 192.168.60.5   192.168.53.99  ICMP   100 Echo (ping) reply    id=0x002d, seq=6/1536, ttl=64
47 2021-07-30 20:1… 10.9.0.11      10.9.0.5       UDP    128 9090 → 56424 Len=84
48 2021-07-30 20:1… 10.9.0.11      10.9.0.5       UDP    128 9090 → 56424 Len=84
```

执行telnet同样也成功:



```
root@8c5a26eb7693:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
213c0d7058c8 login: root
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

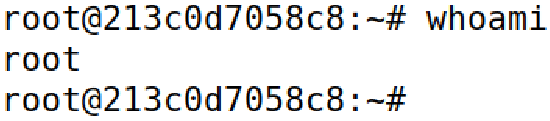
Wireshark中也可以看到telnet产生的报文:



```
332 2021-07-30 20:2… 10.9.0.11      10.9.0.5       UDP     506 9090 → 56424 Len=462
333 2021-07-30 20:2… 10.9.0.5       10.9.0.11      UDP      96 56424 → 9090 Len=52
334 2021-07-30 20:2… 10.9.0.5       10.9.0.11      UDP      96 56424 → 9090 Len=52
335 2021-07-30 20:2… 192.168.53.99  192.168.60.5   TCP      68 49076 → 23 [ACK] Seq=1903837977 Ack=1712523979 Win=64128 Len=…
336 2021-07-30 20:2… 192.168.53.99  192.168.60.5   TCP      68 [TCP Dup ACK 335#1] 49076 → 23 [ACK] Seq=1903837977 Ack=17125…
337 2021-07-30 20:2… 192.168.60.5   192.168.53.99  TELNET  119 Telnet Data ...
338 2021-07-30 20:2… 192.168.60.5   192.168.53.99  TCP     119 [TCP Retransmission] 23 → 49076 [PSH, ACK] Seq=1712523979 Ack…
339 2021-07-30 20:2… 10.9.0.11      10.9.0.5       UDP     147 9090 → 56424 Len=103
340 2021-07-30 20:2… 10.9.0.11      10.9.0.5       UDP     147 9090 → 56424 Len=103
341 2021-07-30 20:2… 10.9.0.5       10.9.0.11      UDP      96 56424 → 9090 Len=52
342 2021-07-30 20:2… 10.9.0.5       10.9.0.11      UDP      96 56424 → 9090 Len=52
343 2021-07-30 20:2… 192.168.53.99  192.168.60.5   TCP      68 49076 → 23 [ACK] Seq=1903837977 Ack=1712524030 Win=64128 Len=…
344 2021-07-30 20:2… 192.168.53.99  192.168.60.5   TCP      68 [TCP Dup ACK 343#1] 49076 → 23 [ACK] Seq=1903837977 Ack=17125…
345 2021-07-30 20:2… 192.168.60.5   192.168.53.99  TELNET   89 Telnet Data ...
```

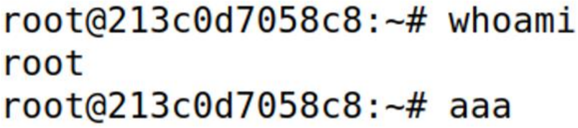
Task 6: Tunnel-Breaking Experiment

在主机U上，telnet到主机v。在保持telnet连接活动的同时，终止client.py程序来中断VPN隧道，用户输入不会显示:



```
root@213c0d7058c8:~# whoami
root
root@213c0d7058c8:~#
```

重新运行client.py，用户的输入一次性显示出来



```
root@213c0d7058c8:~# whoami
root
root@213c0d7058c8:~# aaa
```