**Task 1**

虚拟机A：Host U 192.168.1.105

虚拟机B：VPN server 192.168.1.107 192.168.10.107

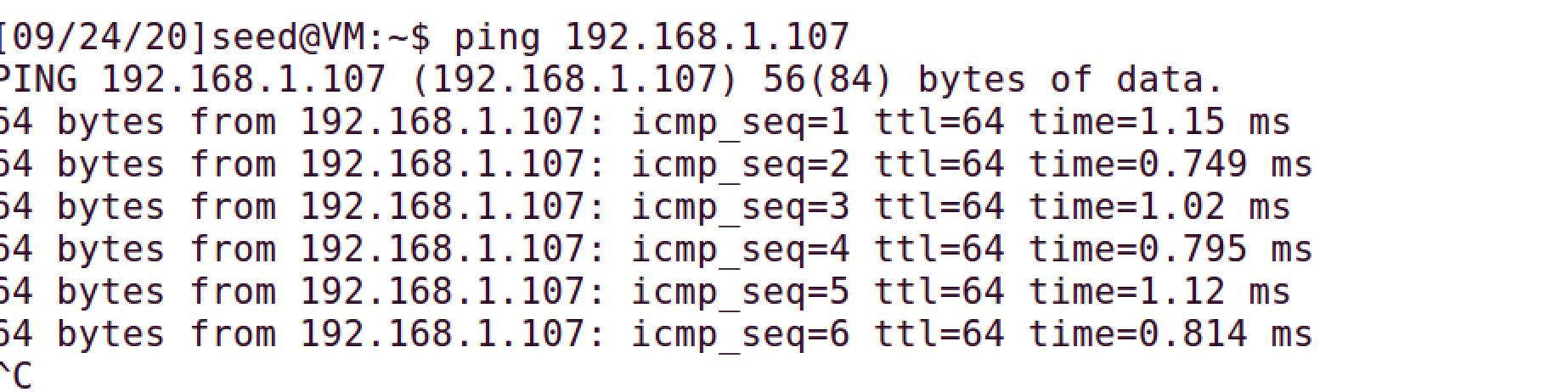
虚拟机C：Host V 192.168.1.108 192.168.10.108

使用虚拟机B作为VPN server，添加一个网卡，使用内部网络模式

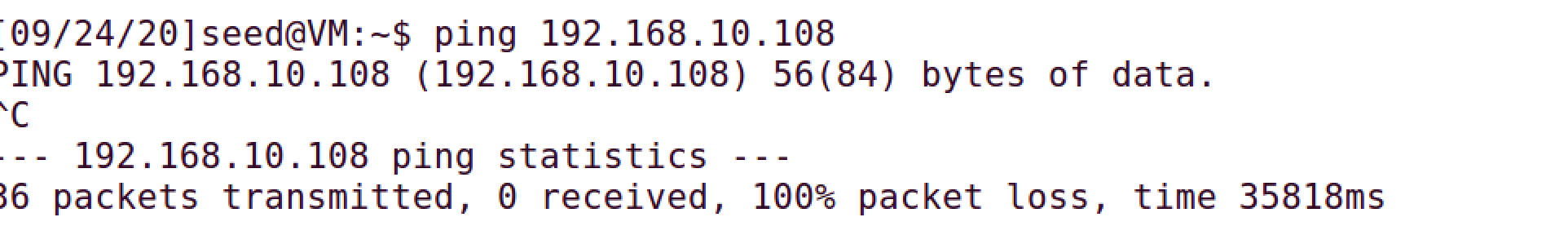
配置地址为192.168.10.107

虚拟地址C配置地址为192.168.10.108

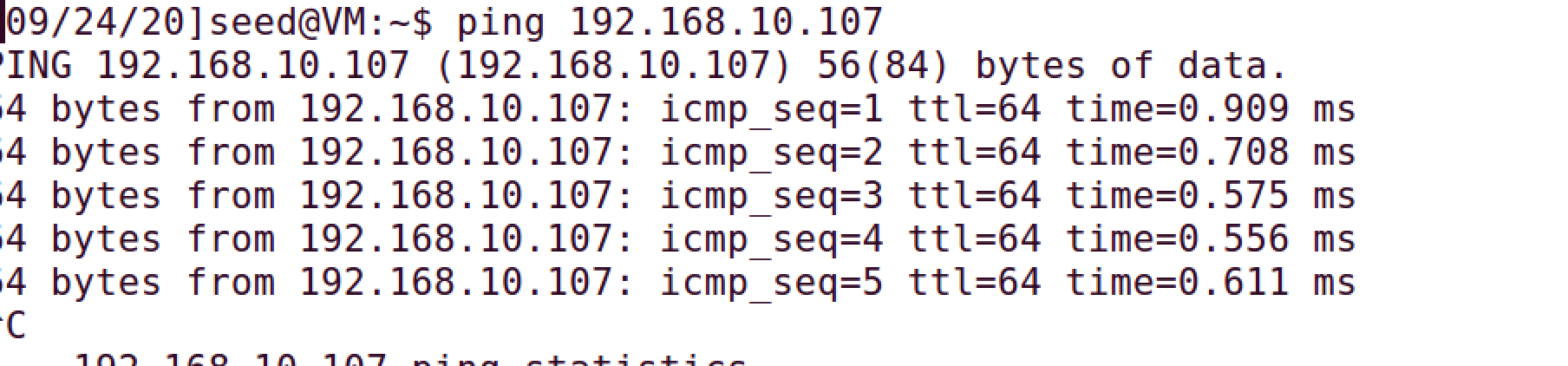
host U ping VPN server



host U ping host V



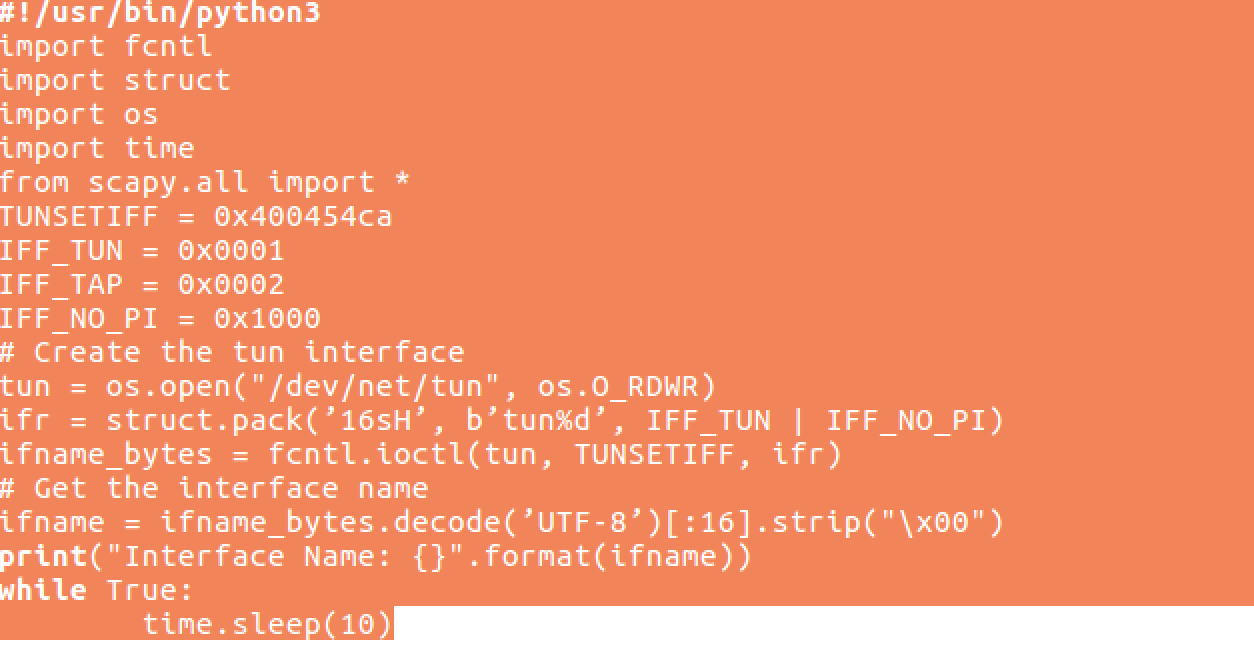
host V ping VPN server



配置完成。

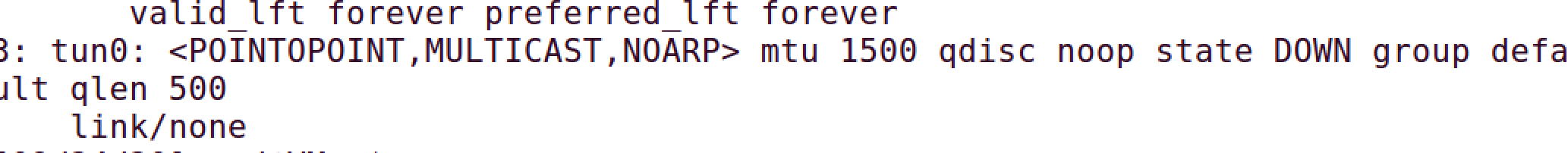
**Task 2**

在U上运行tun.py程序添加一个TUN虚拟接口



Task 2.a: Name of the Interface

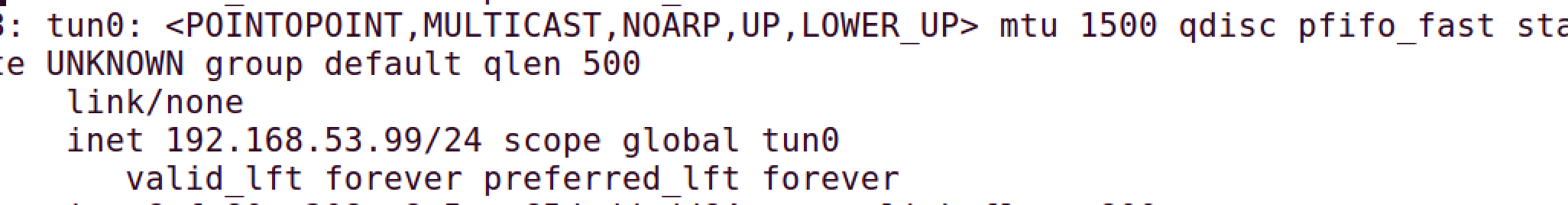
查看接口：



多出一个tun0接口

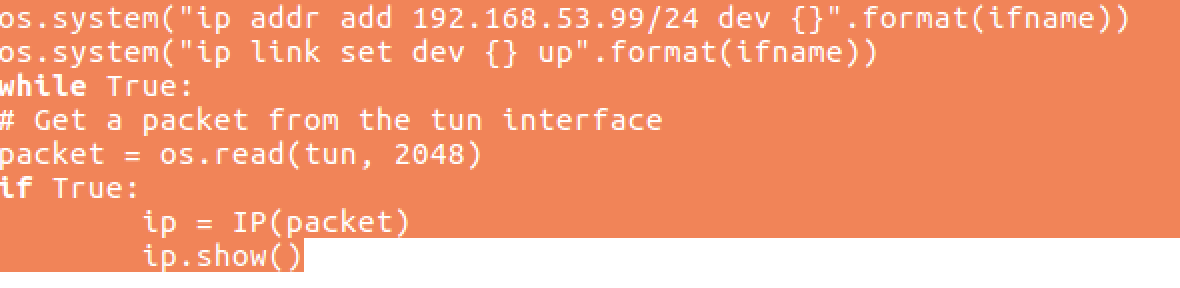
Task 2.b: Set up the TUN Interface

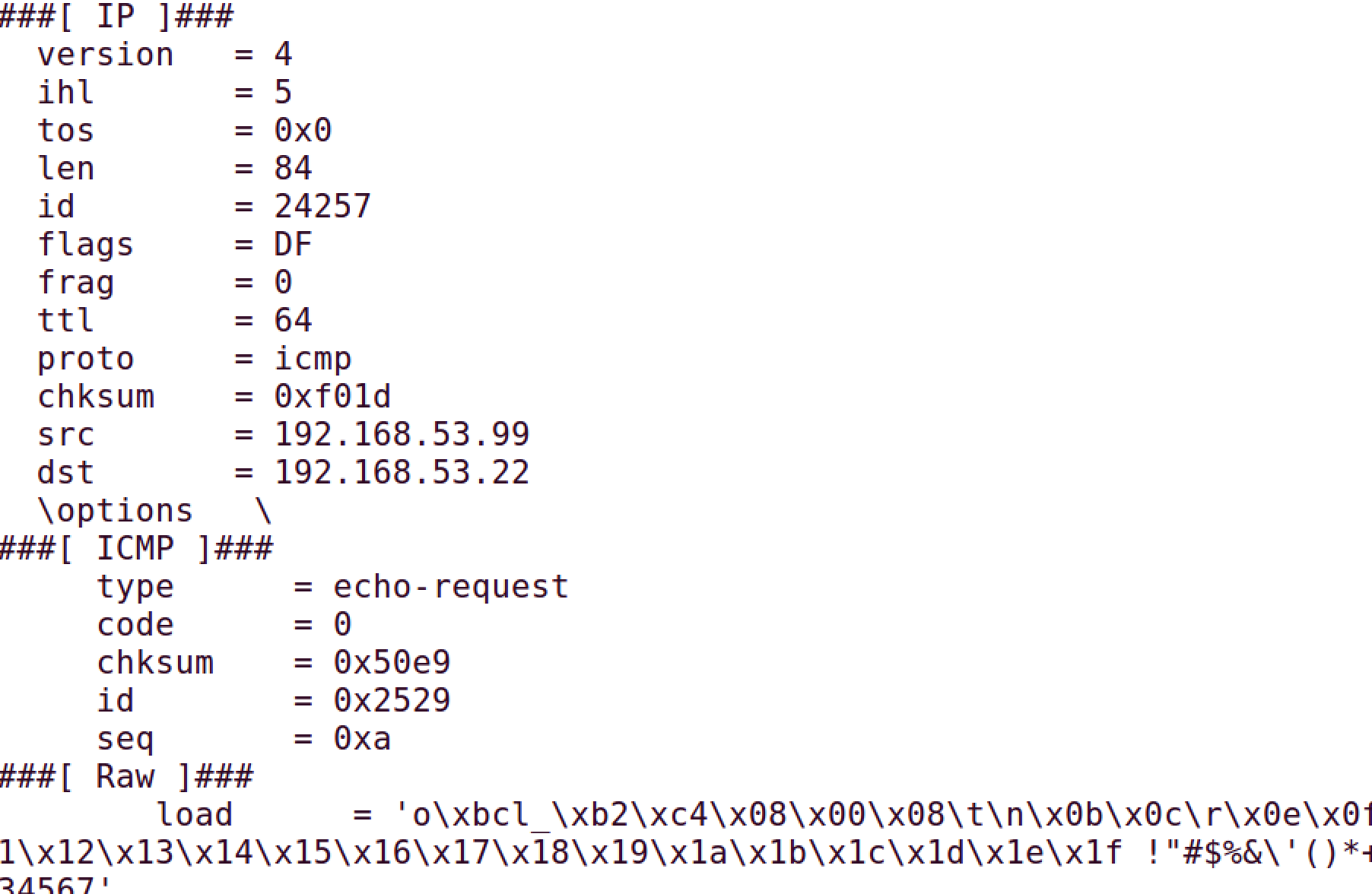
按实验手册输入命令给tun0接口分配IP地址并打开



Task 2.c: Read from the TUN Interface

添加代码（代码中if语句前应加tab，复制时空格不会被复制，导致逻辑错误）



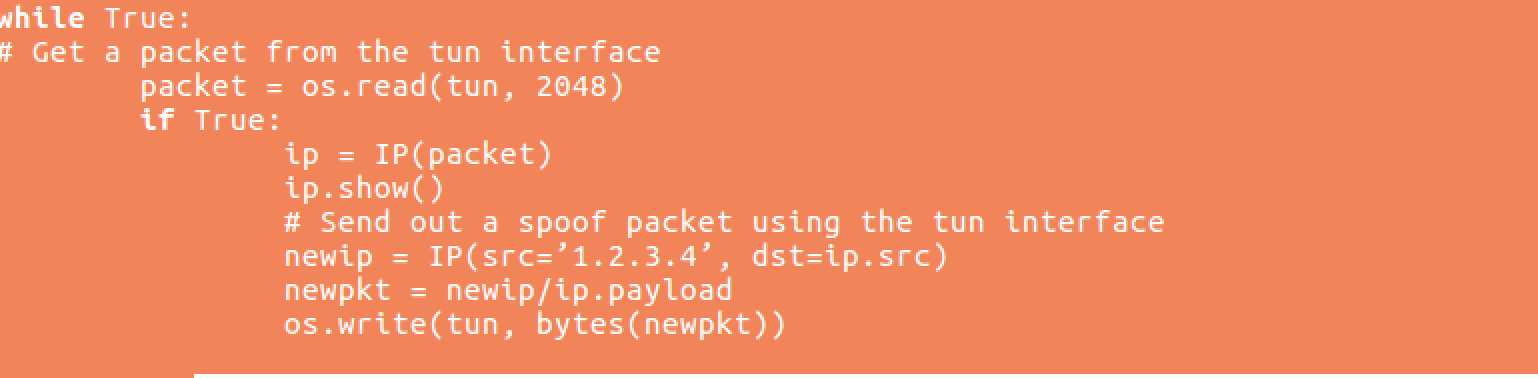


tun.py程序打印出了发往192.168.53.22的数据包，数据包在路由选择时被发给了tun0虚拟接口。数据包的源地址被修改成了tun0端口的IP地址

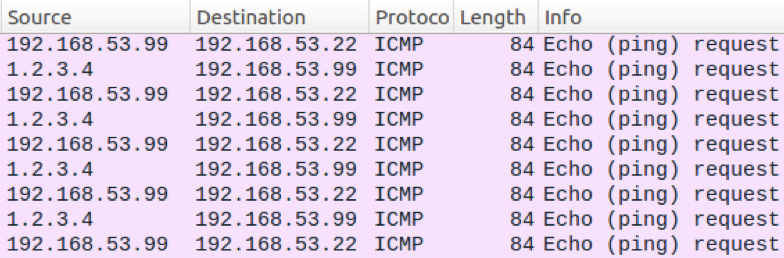
ping 192.168.70.0/24网内地址进行测试：

无事发生，此时192.168.70.0/24这一网络还不在tun0的转发范围内

Task 2.d: Write to the TUN Interface



运行程序，使用wireshark观察



既有从192.168.53.99发往192.168.53.22的ICMP请求报文，又有从1.2.3.4发往192.168.53.99的ICMP请求报文

向tun中写入任意数据，产生"Invalid argument"的错误，可知只能向TUN中写入IP数据报。

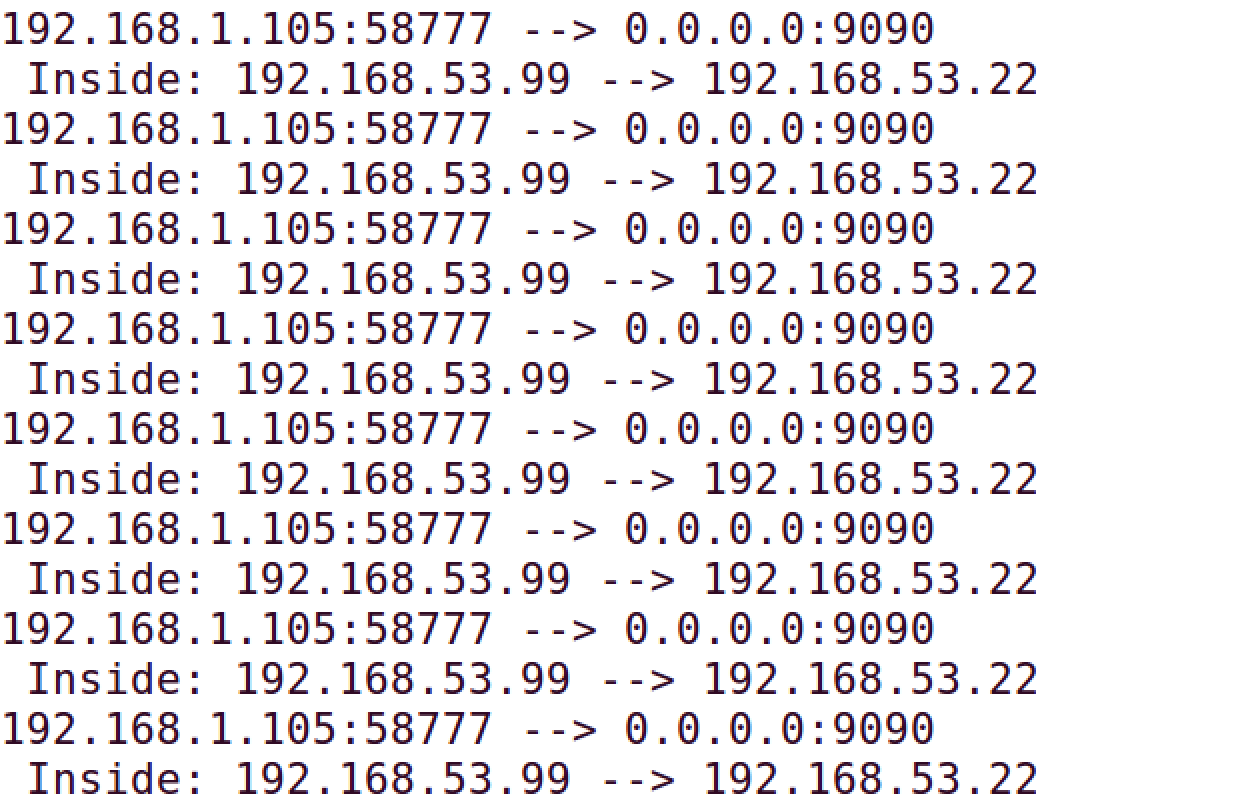
Task 3: Send the IP Packet to VPN Server Through a Tunnel

在VPN服务器上运行tun\_server.py程序，监听9090端口并打印出所有收到的数据



ping 192.168.53.0/24网内任意IP地址

VPN端输出：



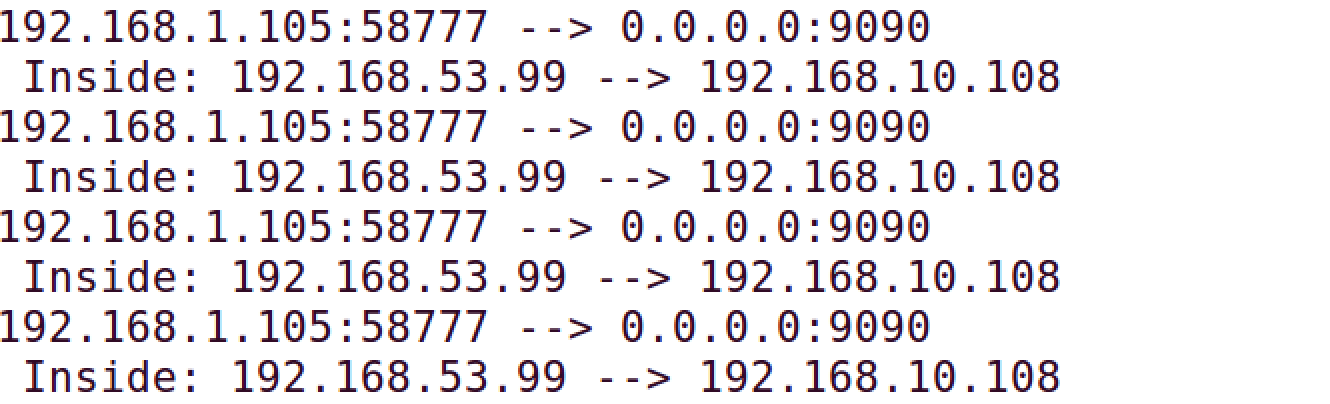
VPN服务器能够收到数据报，外层为192.168.1.105发往自身9090端口的IP数据报，内层为192.168.53.99发往192.168.53.22的IP数据报，可见内层数据报通过IP隧道在主机U与VPN服务器间实现了通信

ping 内部主机V：

只能看到主机U以0.0.0.0的IP地址发往其他IP地址的数据报，没有看到发往内部主机V的数据报，原因是目的地地址为V的数据报没有被tun接口处理

在主机U上配置路由信息，将192.168.10.0/24网段的出口设为虚拟接口tun0

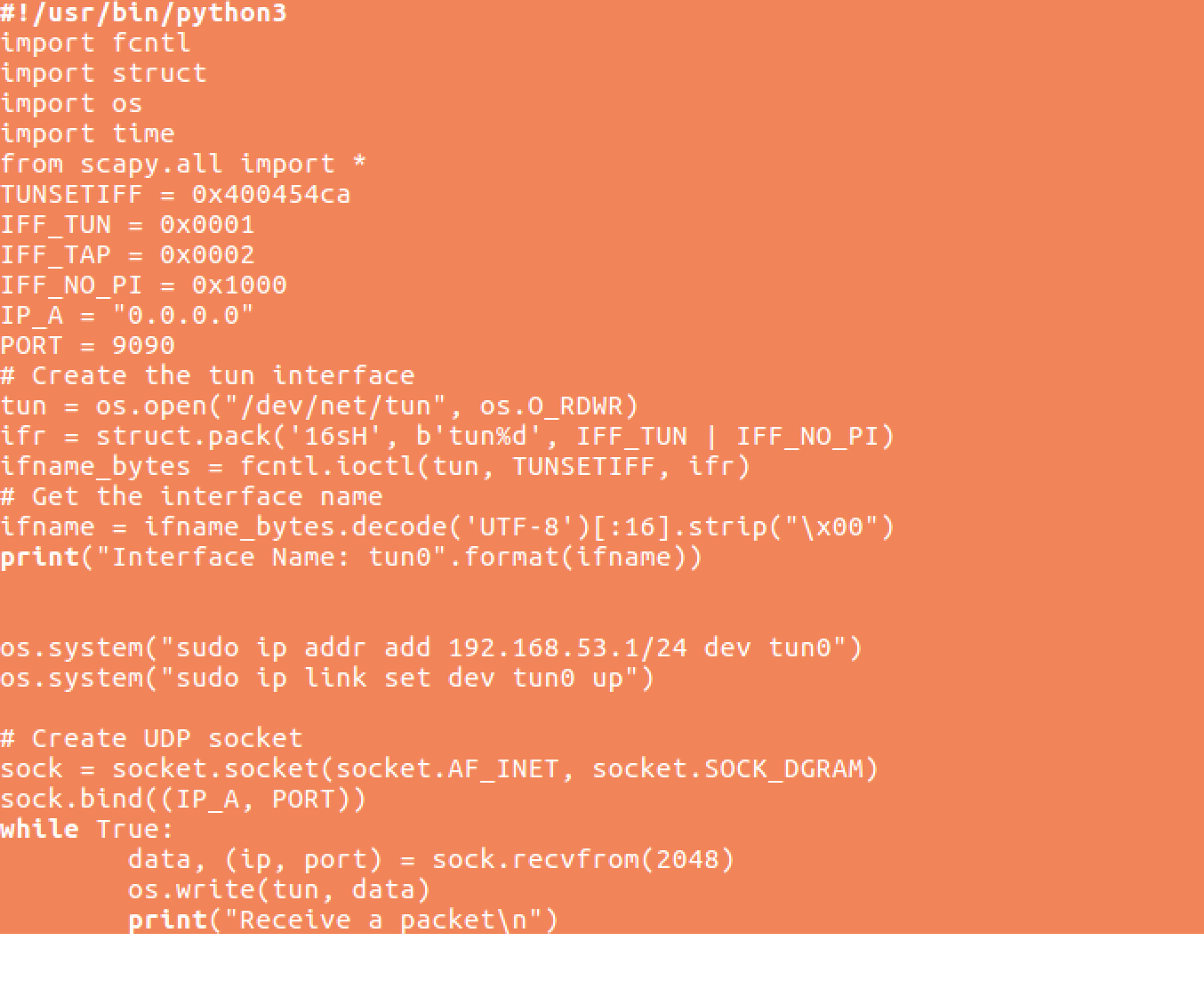
sudo ip route add 192.168.10.0/24 dev tun0 via 192.168.53.99



此时服务器端可以收到从主机U上发往192.168.10.108的报文

Task 4: Set Up the VPN Server

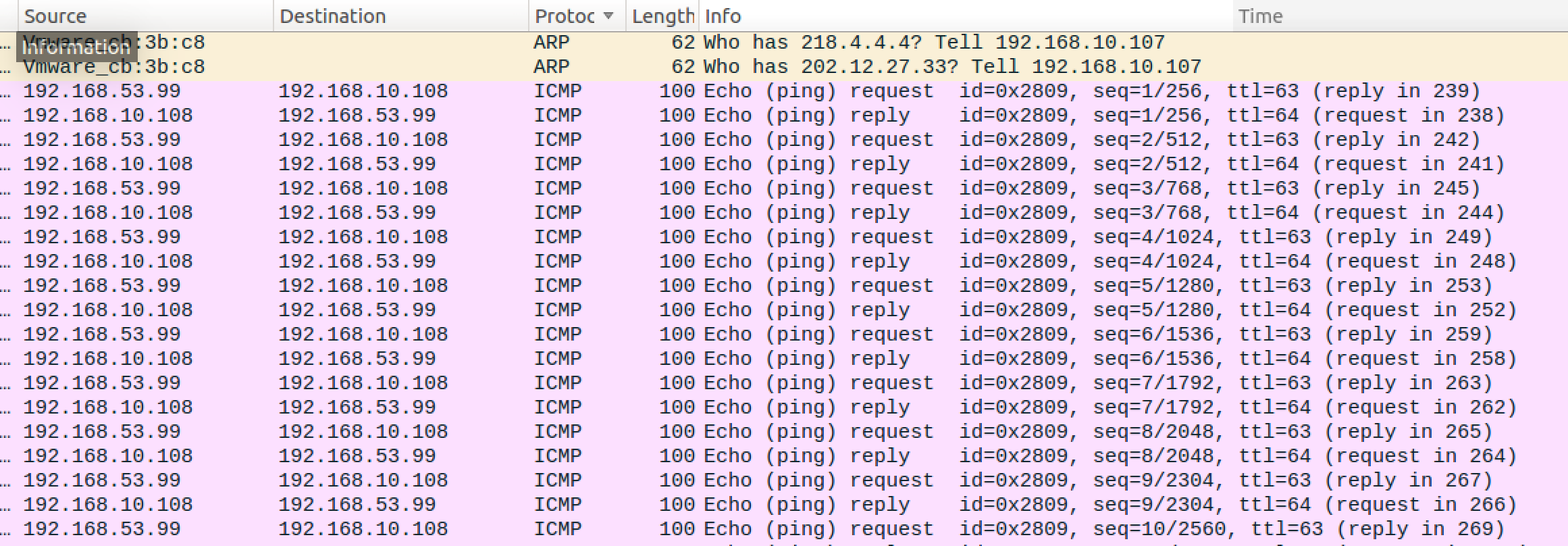
编写tun\_server.py代码：



打开VPN服务端的路由转发选项

U ping V

主机V上打开wireshark观察：



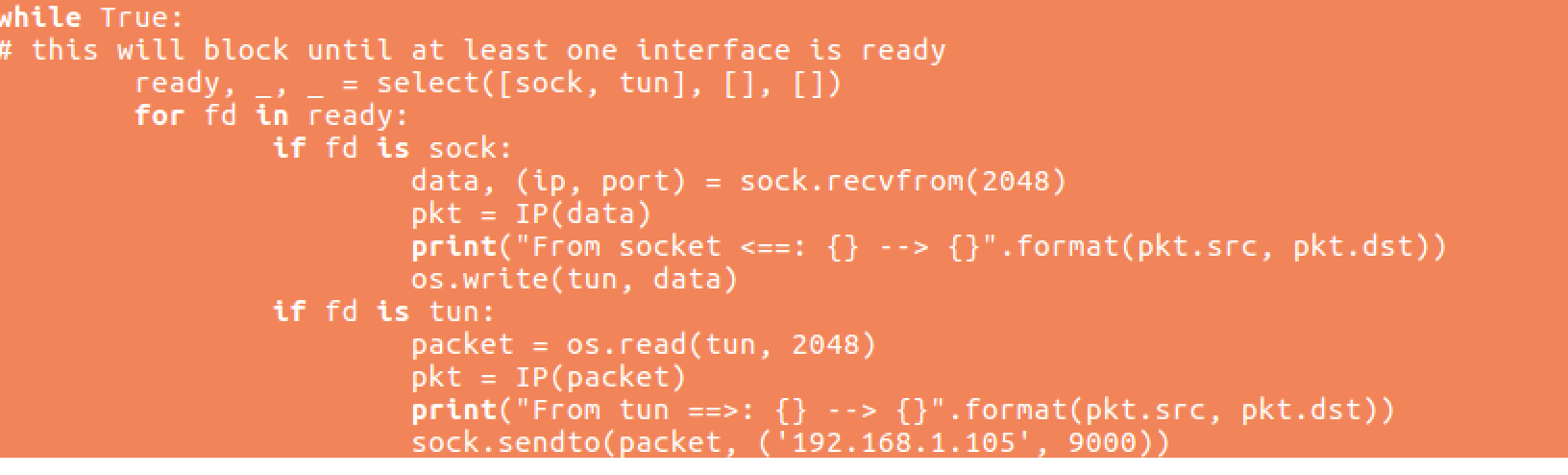
接收到了来自于主机U的ICMP报文

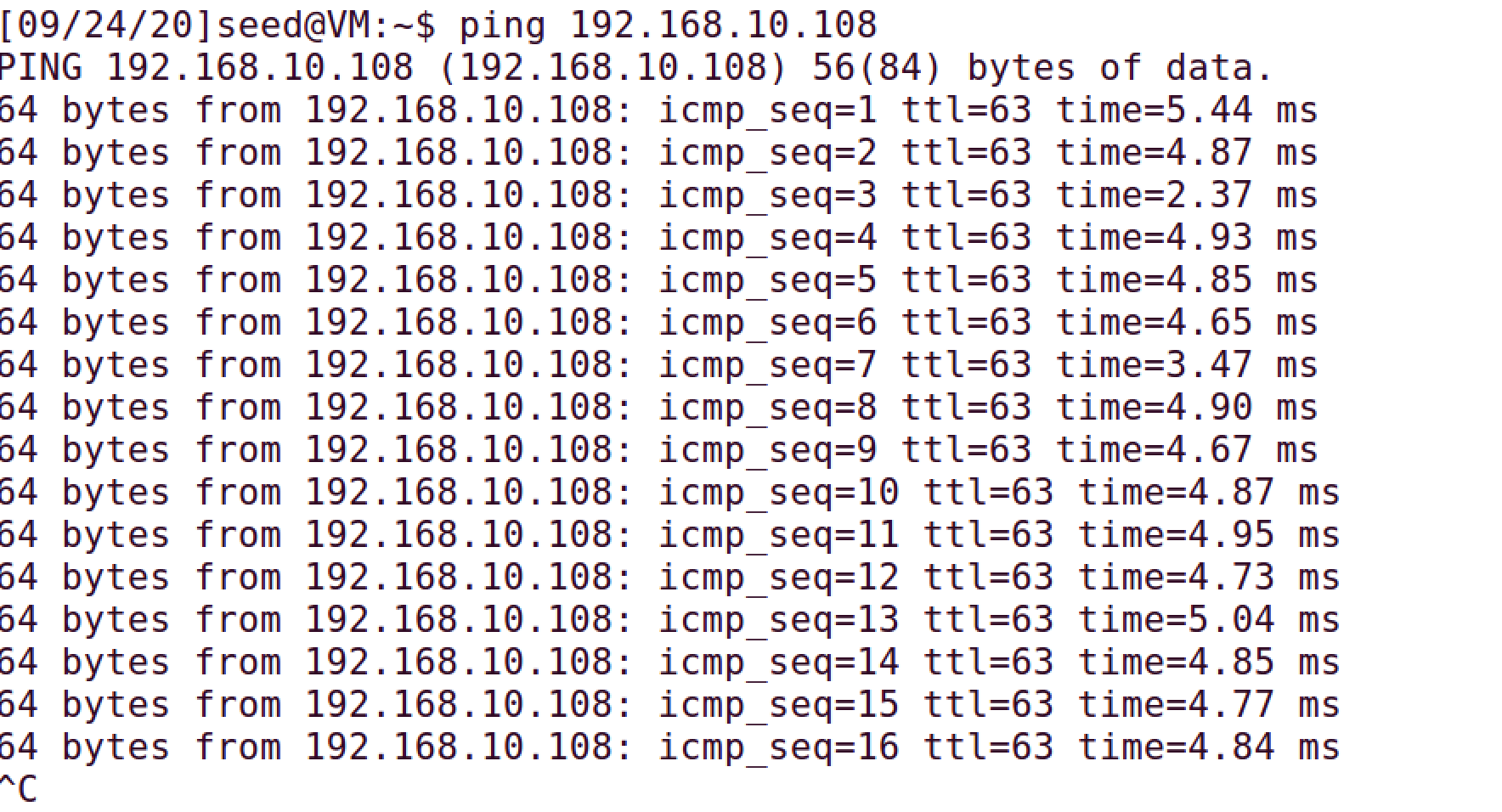
Task 5: Handling Traffic in Both Directions

客户端：



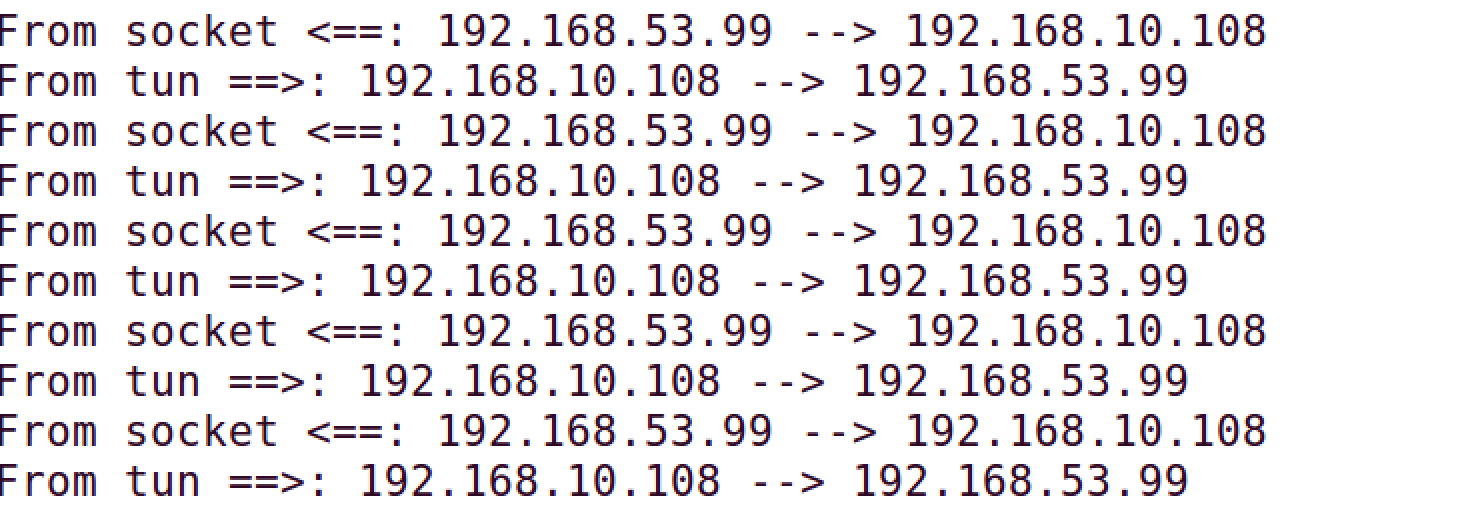
服务器端：

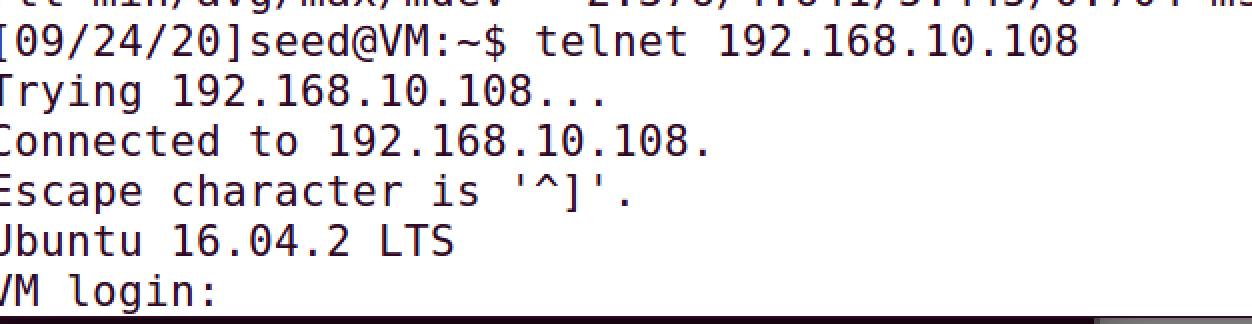




U可以ping通V，

服务器端输出：





Telnet连接也能成功建立。

Task 6: Tunnel-Breaking Experiment

使用IP隧道进行telnet通信过程中，关闭tun\_server.py。此时无法再输入命令，且客户端与服务端不再有报文传输。等待一段时间后重新连接，客户端和服务端传送大量报文，通过wireshark，得知双方重新连接了telnet服务。

当IP隧道断开后，如果能在较短时间内重新连接上，仍可以继续断开前的业务，原因可能是报文被暂存在了缓存中。

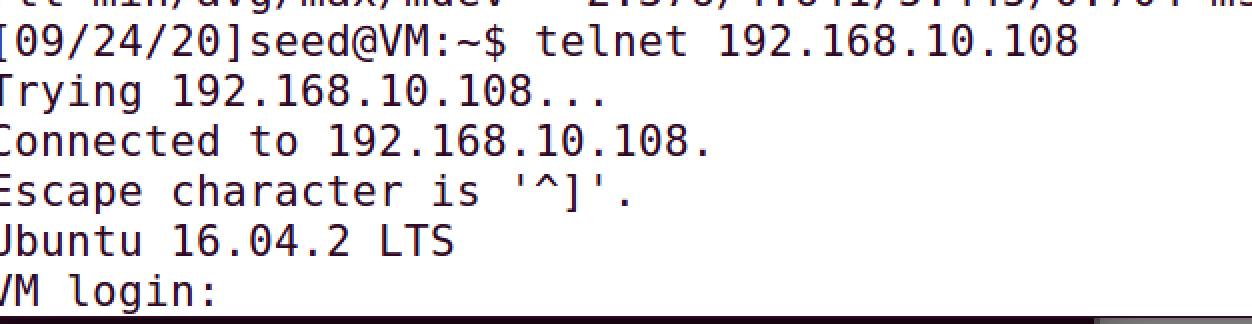
Task 7: Routing Experiment on Host V

sudo ip route del 0.0.0.0/0

sudo ip route add 192.168.53.0/24 dev ens33 via 192.168.10.107

sudo ip route add 192.168.1.0/24 dev ens33 via 192.168.10.107

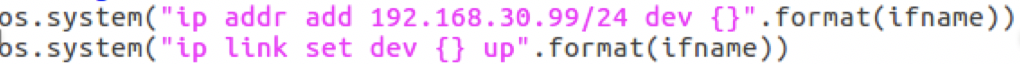
主机U向主机V发起telnet：



可以成功通信。

Task 8: Experiment with the TUN IP Address

修改客户端TUN端口IP地址，使其与服务端TUN接口IP不位于同一网段



此时无法ping通

观察主机U上wireshark：主机U上既有从ens33发出的192.168.1.105->192.168.1.107的UDP数据报，也有从tun0发出的192.168.30.99->192.168.10.108的ICMP数据报。

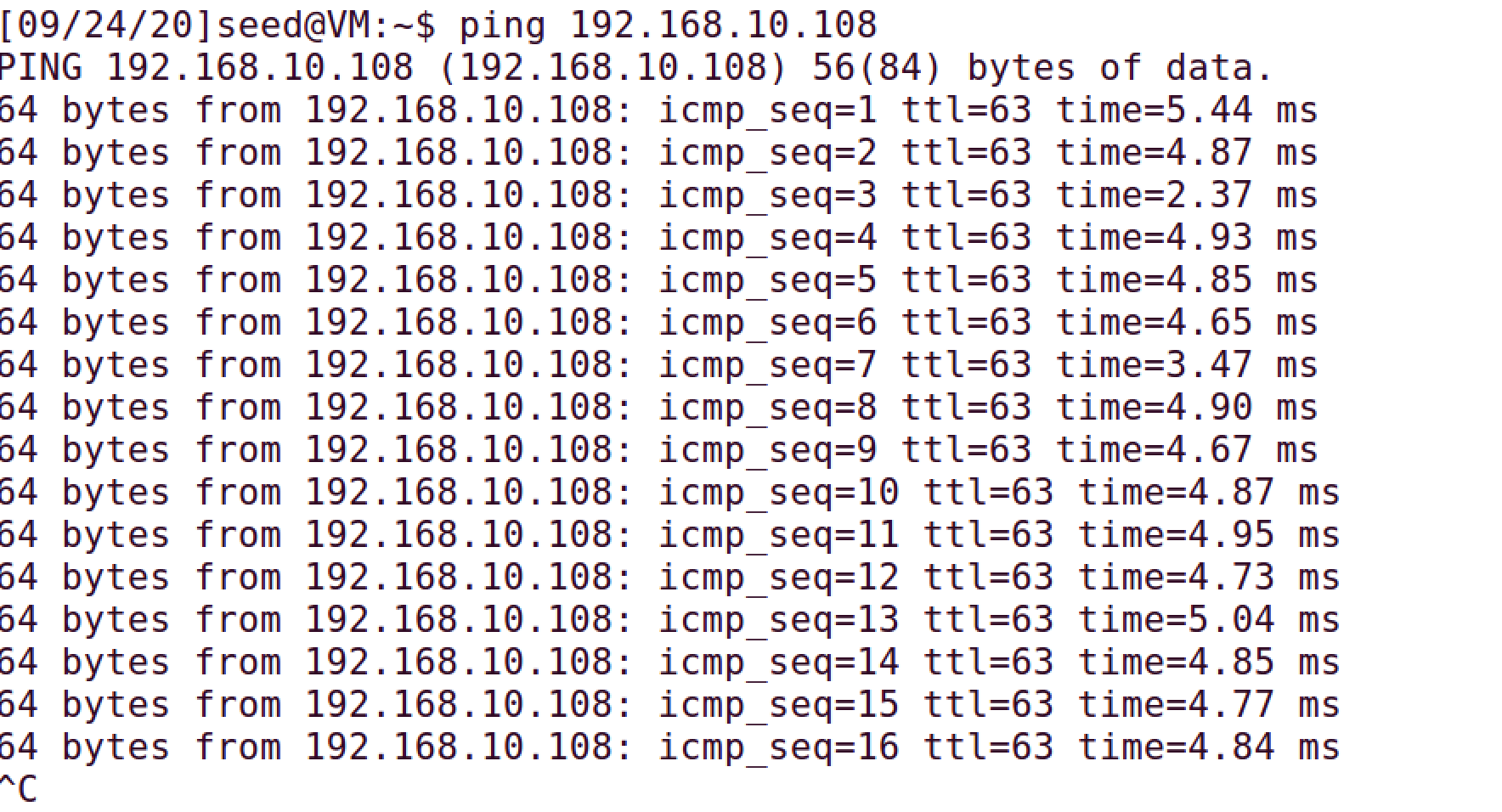
观察VPN服务器上wireshark：ens33和tun0端口都分别有收到UDP和ICMP报文，但ens38端口却没有报文发出，即VPN服务器在内层的ICMP报文进行路由选择时，并未将其从ens38端口转发出去。

Linux内核中自带RPC机制，其会对收到的IP数据报的源IP进行反向路由查找，即以该IP地址作为目的IP查找路由表，得到去往该IP地址应出发的网络设备端口，若IP数据报不是来源于该端口，则认为这是一个伪造的报文而丢弃。

在VPN服务器上添加192.168.30.0/24网络与tun0端口关联的路由表项:

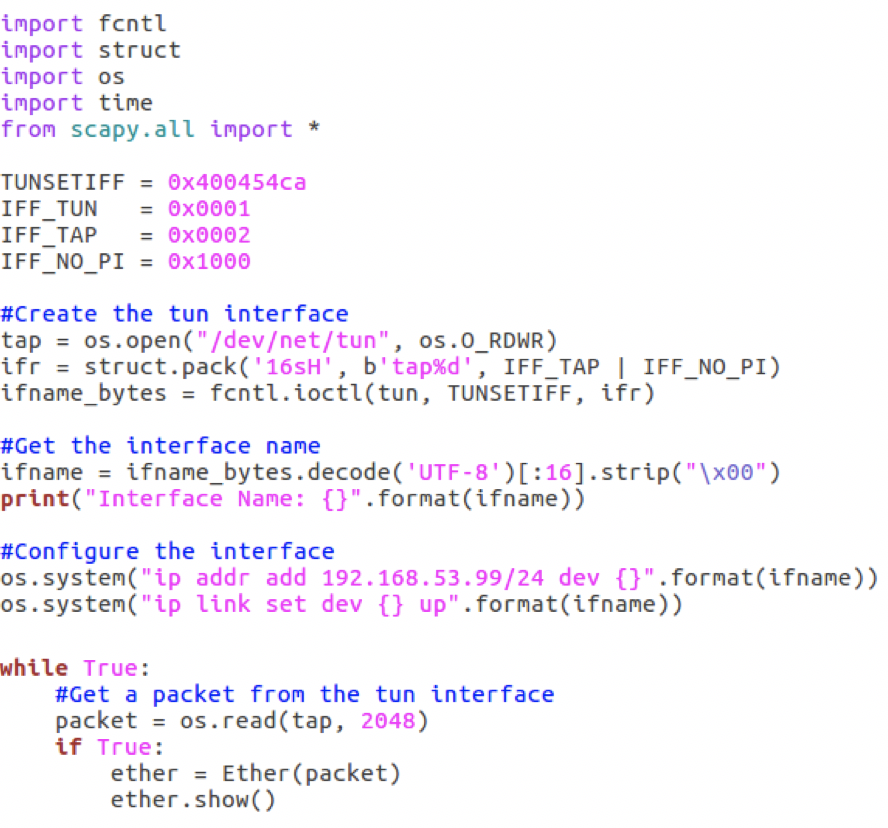
sudo ip route add 192.168.30.0/24 dev tun0

此时即可正常通信



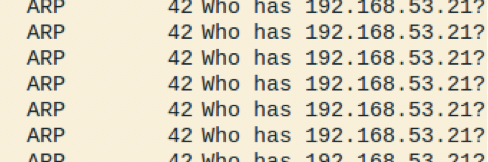
Task 9: Experiment with the TAP Interface

编写程序搭建基于TAP端口的隧道进行通信



ping 192.168.53.21进行测试，发现无响应

wireshark中显示：



tap0端口向外发送ARP请求，查询192.168.53.21的MAC地址，由于这是不存在的网络，所以ARP请求不能收到响应，导致ICMP数据包滞留在端口中不能发送出去。可以看出tap工作在MAC层。