**实验1：VPN Tunneling Lab**

## Task 1: Network Setup

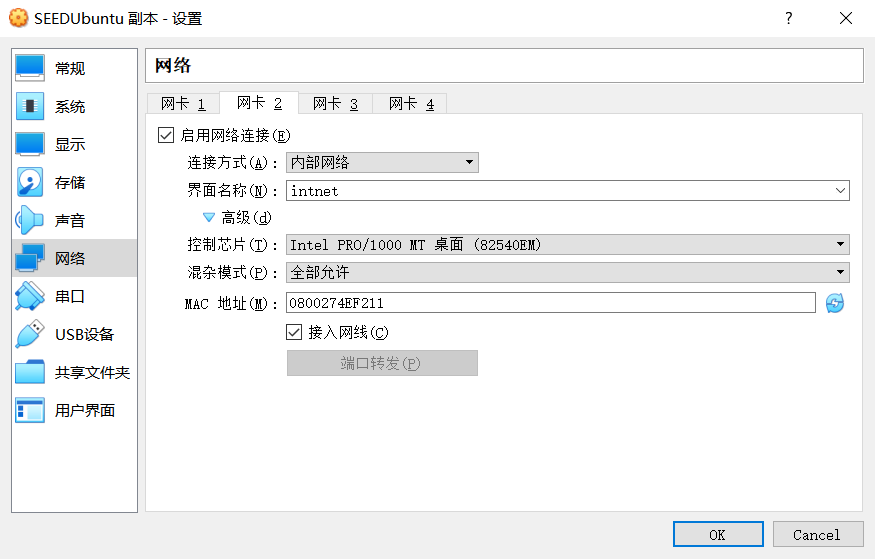
虚拟机A（Host U）：10.0.2.14

虚拟机B（VPN server）：10.0.2.13 192.168.70.1

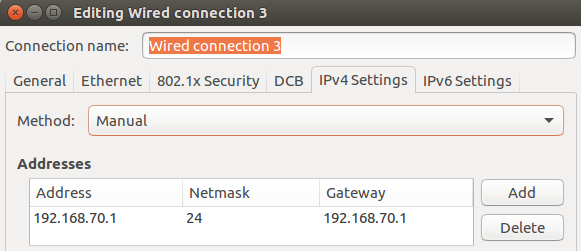
虚拟机C（Host V）：192.168.70.2

① 配置虚拟机B作为VPN server

为虚拟机B添加一块网卡，网卡设为内部网络模式

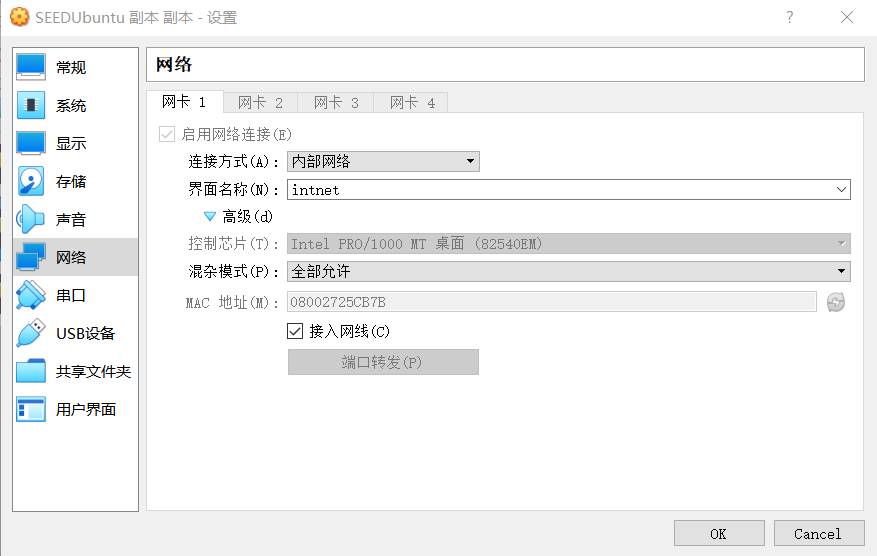


进入虚拟机，手动配置网卡所在网段的地址：192.168.70.1/24

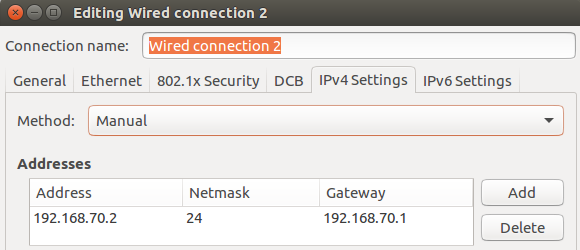


② 配置虚拟机C作为Host V

将网卡修改为内部网络模式，只能与同一网络内的主机通信

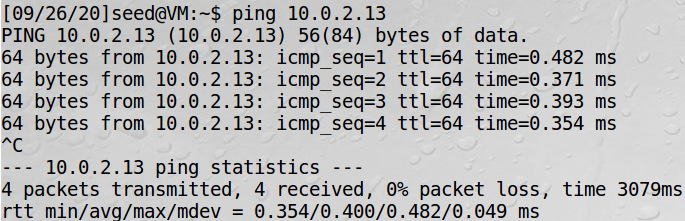


进入虚拟机，手动配置网络信息，自身IP地址为192.168.70.2，网关为VPN server的IP地址192.168.70.1

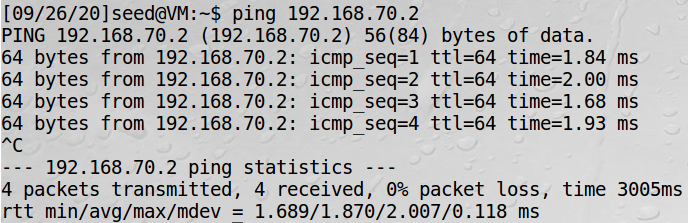


③ 效果测试

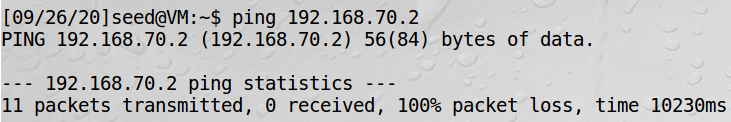
Host U ping VPN server，可以ping通



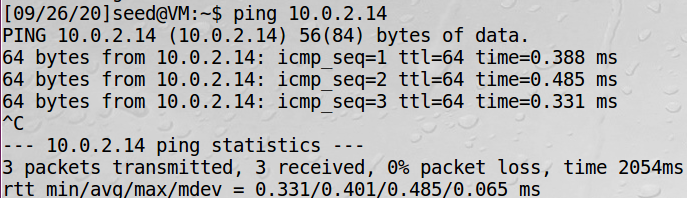
VPN server ping Host V，可以ping通



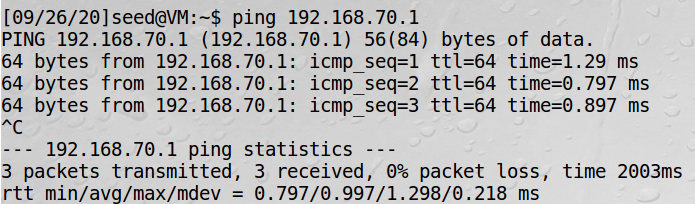
Host U ping Host V，ping不通



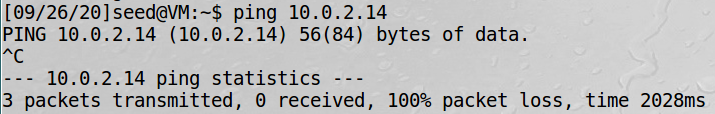
VPN server ping Host U，可以ping通



Host V ping VPN server，可以ping通



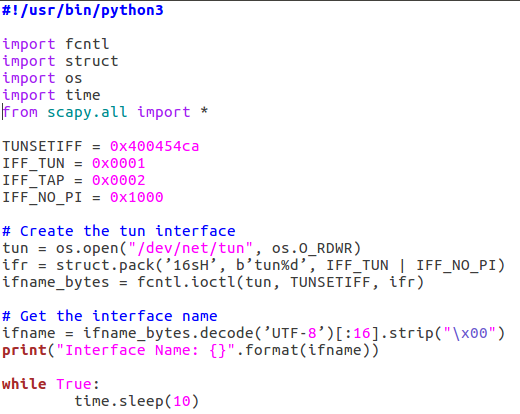
Host V ping Host U，ping不通

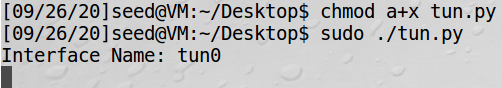


## Task 2: Create and Conﬁgure TUN Interface

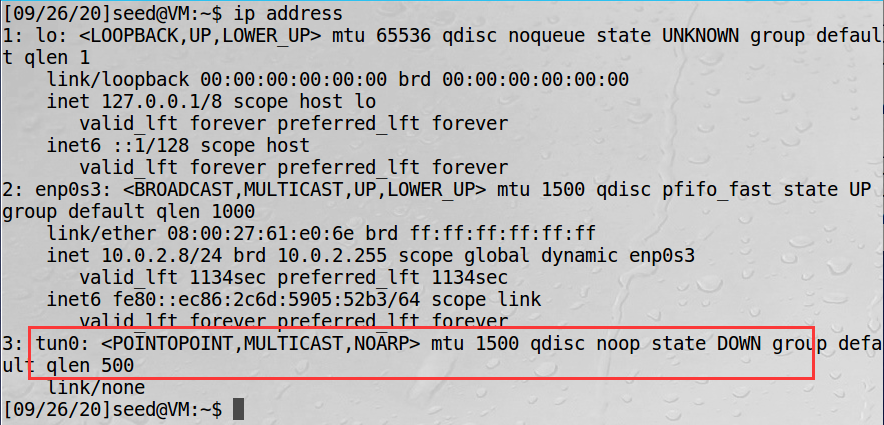
## Task 2.a: Name of the Interface

运行tun.py，创建一个TUN接口





打开一个新的终端，观察端口信息，可以观察到一个新端口

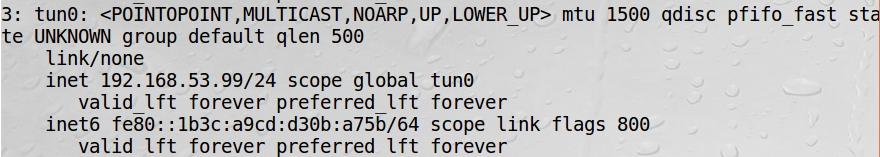


## Task 2.b: Set up the TUN Interface

为端口分配IP地址，并启用端口

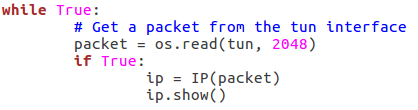


再次观察端口信息，可以看到tun0被启用，且有IP地址192.168.53.99/24

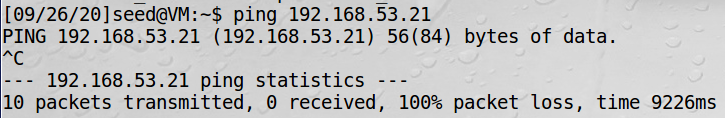


## Task 2.c: Read from the TUN Interface

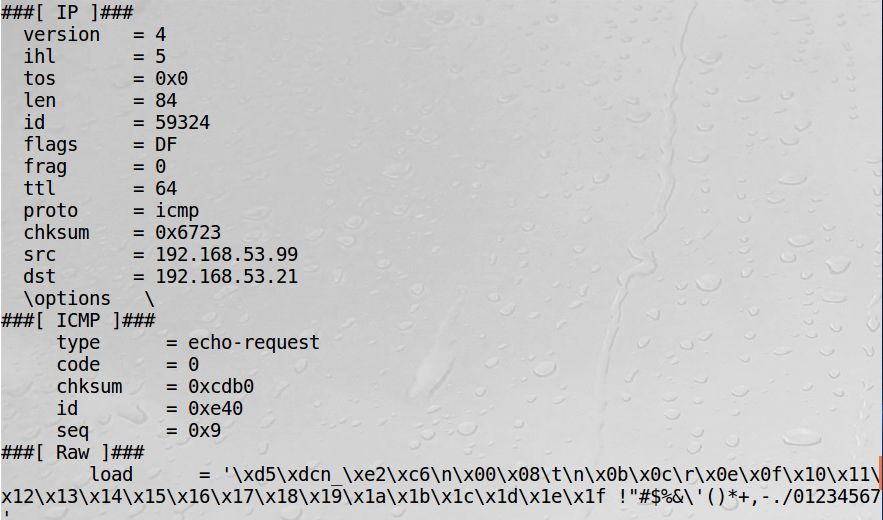
在tun.py中添加如下代码并运行



ping 192.168.53.0/24中的一个地址

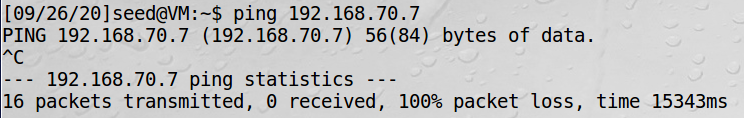


tun.py程序打印出了数据包的内容



我们发往192.168.53.21的数据包的源地址为我们设置的虚拟端口tun0的IP地址，说明数据包在路由选择时被发给了tun0接口

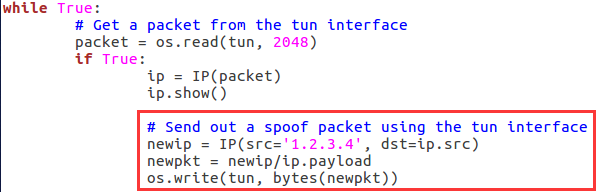
ping 192.168.70.0/24中的一个地址



tun.py程序没有输出内容，这是因为192.168.70.0/24不在tun0所连接到的网络范围内

## Task 2.d: Write to the TUN Interface

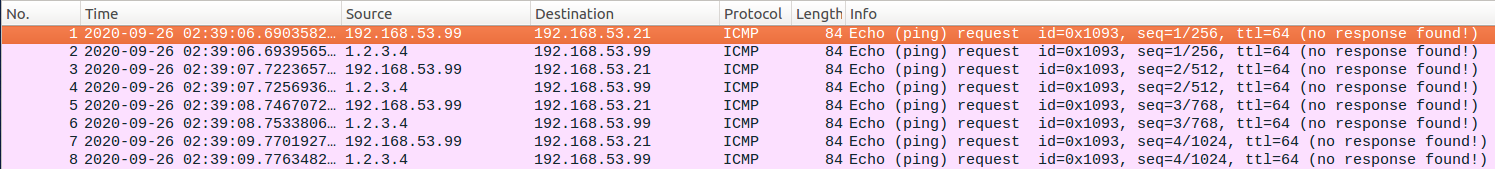
修改tun.py



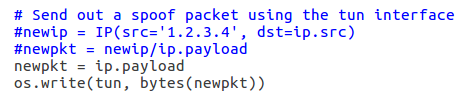
运行tun.py，用wireshark抓包并观察



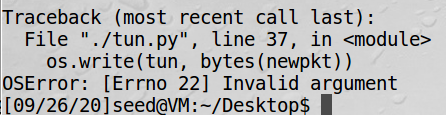
可以看到有从tun0端口（192.168.53.99）发往192.168.53.22的ICMP请求报文，也有从1.2.3.4发往tun0端口（192.168.53.99）的ICMP请求报文



向tun0发送任意数据，而不是IP报文

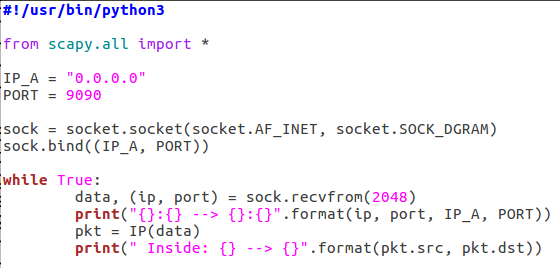


发生参数错误

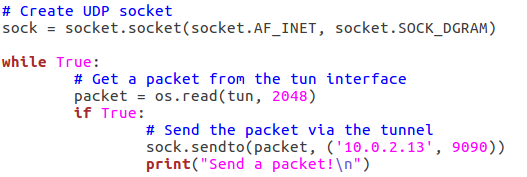


## Task 3: Send the IP Packet to VPN Server Through a Tunnel

在VPN server上运行tun\_server.py



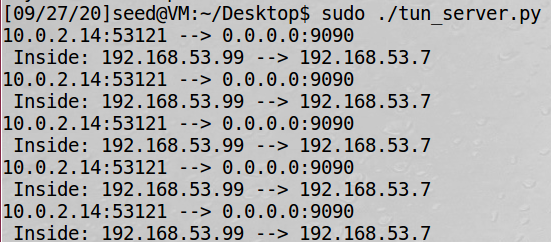
在Host U上运行tun\_client.py



ping 192.168.53.0/24内的任意IP地址



在VPN server上会显示监听到tun0接口向192.168.53.7发送报文



ping 192.168.70.2（主机V），服务器端并没有监听到来自tun0的报文



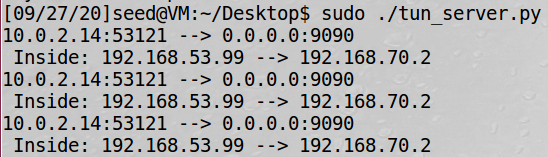
我们需要在主机U上配置静态路由，将192.168.70.0/24网段的出口设为tun0



再次测试

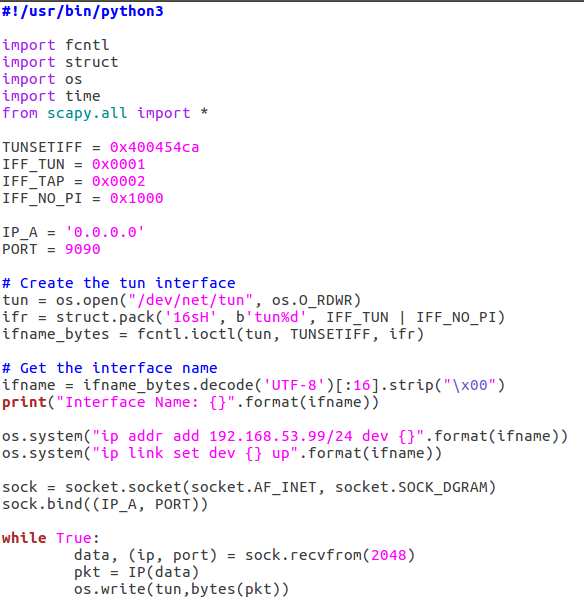


在VPN server上可以监听到tun0接口向192.168.70.2发送报文



## Task 4: Set Up the VPN Server

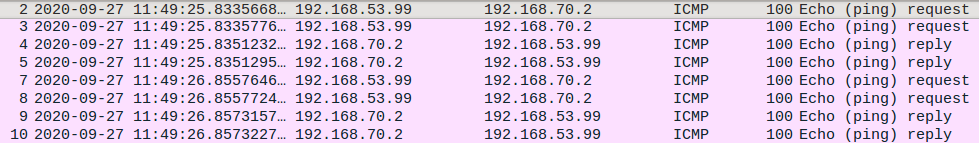
tun\_server.py修改如下



打开服务器端的路由转发功能



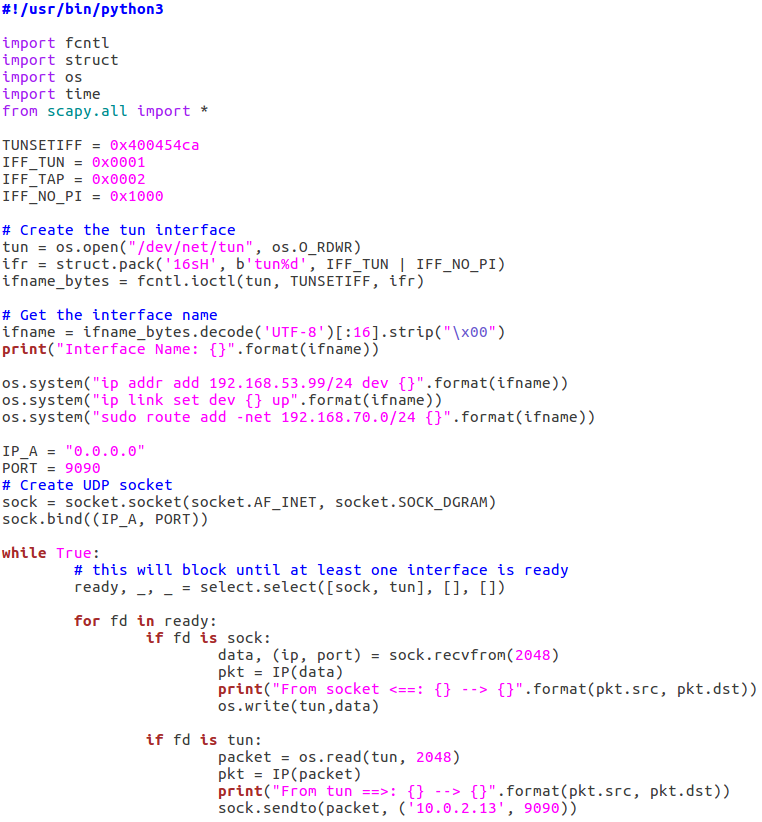
在Host U上ping Host V，并用wireshark抓包



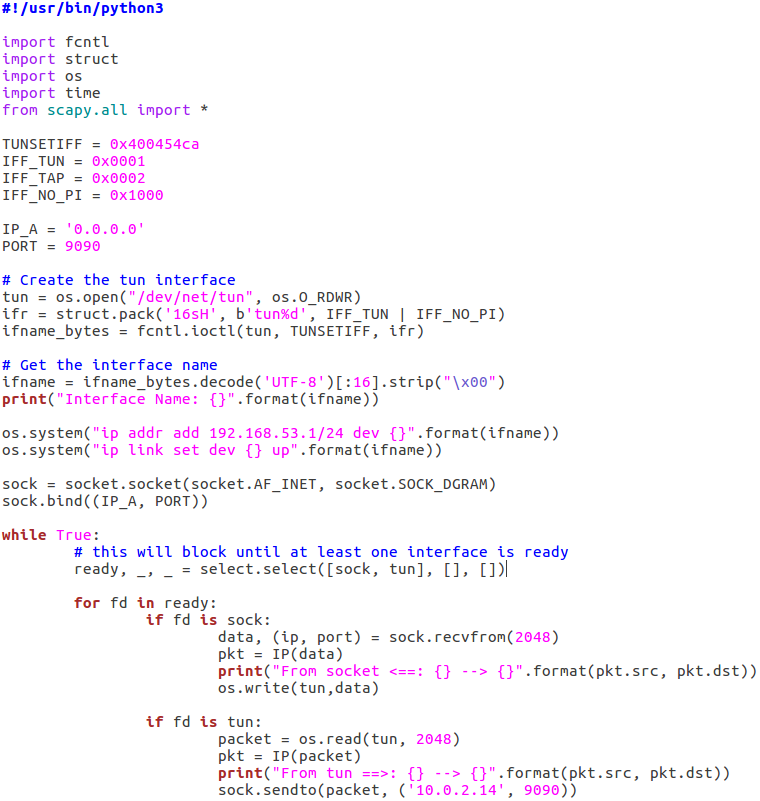
可以观察到来自192.168.53.99，发给192.168.70.2的ICMP请求报文，以及响应报文

## Task 5: Handling Trafﬁc in Both Directions

修改客户端程序tun\_client.py



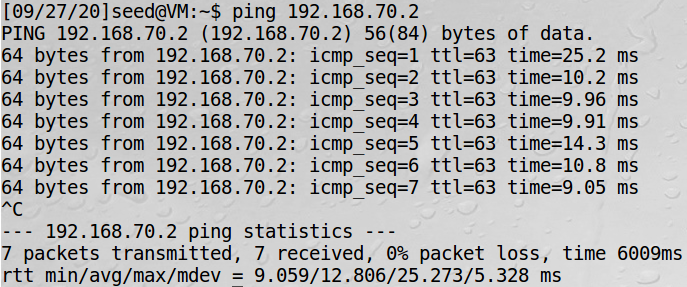
修改服务器程序tun\_server.py



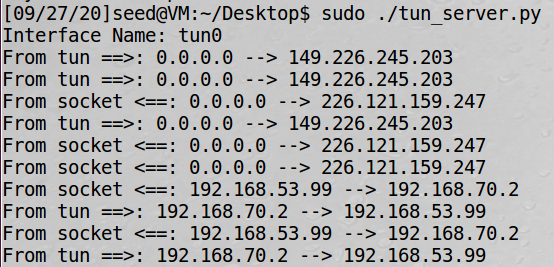
在Host V配置静态路由

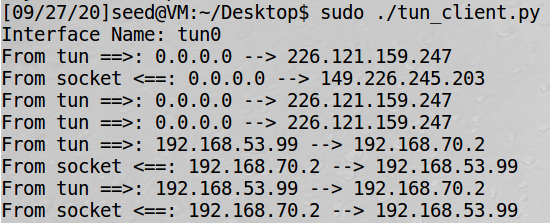


运行两个程序，在Host U上ping Host V，可以ping通



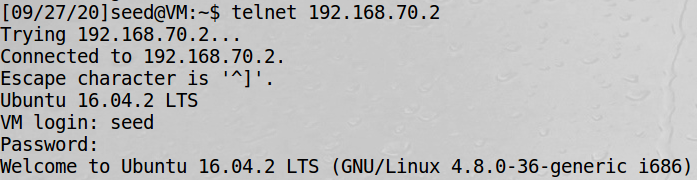
客户端和服务器端能够看到192.168.53.99与192.168.70.2的通信



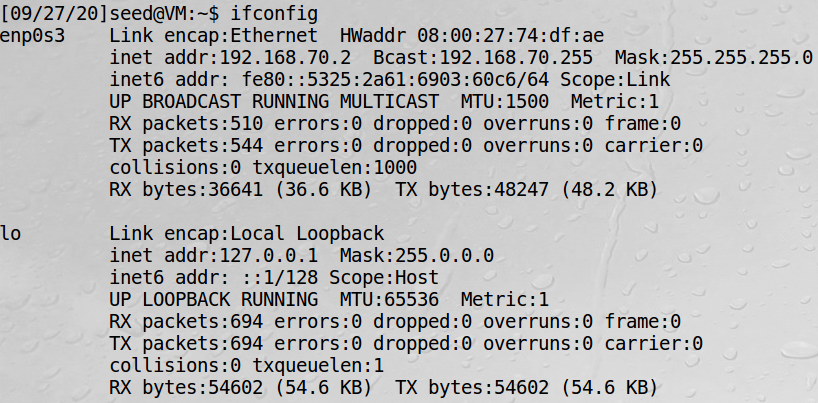


## Task 6: Tunnel-Breaking Experiment

打开隧道，在Host U上与Host V进行telnet通信



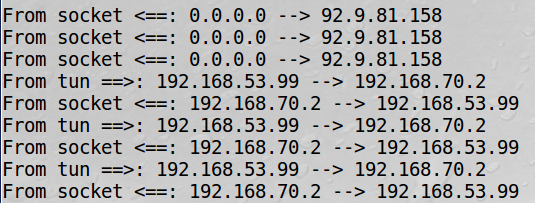
可以正常通信



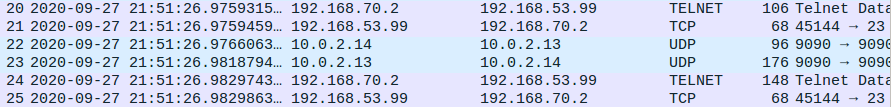
关闭tun\_server.py，此时在Host U无法再输入命令



重新打开隧道后，等待一段时间后重新连接上telnet



用wireshark抓包，可以看到双方重新建立了telnet连接

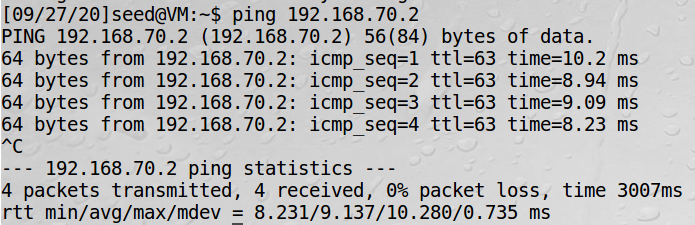


## Task 7: Routing Experiment on Host V

在Host V上配置路由



打开隧道进行测试，可以成功通信

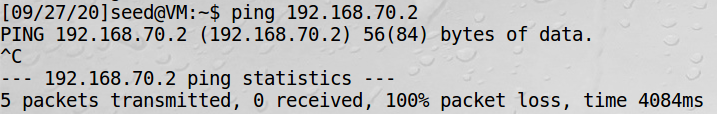


## Task 8: Experiment with the TUN IP Address

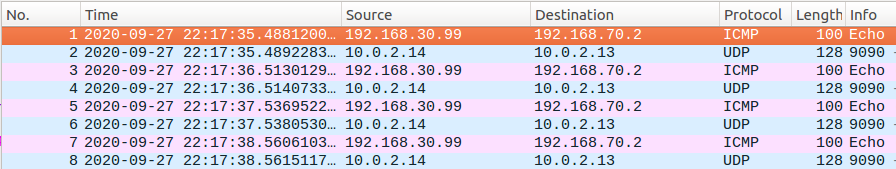
修改客户端TUN端口的IP地址，使它与服务端TUN端口不位于同一网段



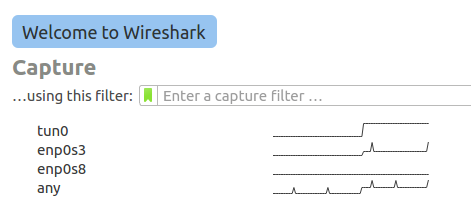
打开隧道进行测试，可以看到此时无法ping通192.168.70.2

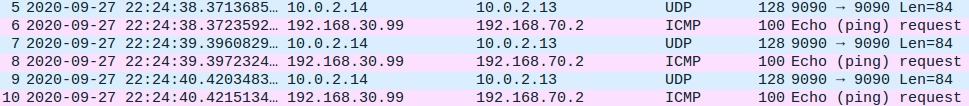


在Host U上用wireshark抓包，可以看到Host U上既有从enp0s3发出的10.0.2.14->10.0.2.13的UDP报文，也有从tun0发出的192.168.30.99->192.168.70.2的ICMP报文



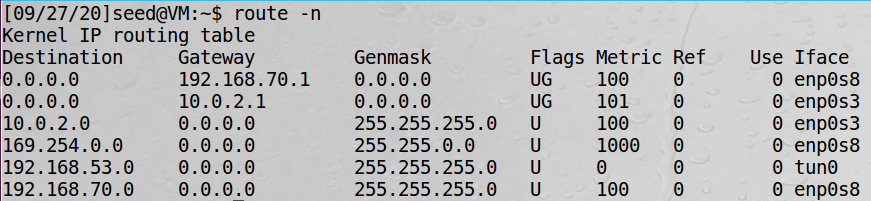
在VPN server上用wireshark抓包，可以看到enp0s3和tun0都分别有收到UDP和ICMP报文，但enp0s8没有报文发出，说明VPN server在内层的ICMP报文进行路由选择时，并未将其从enp0s8转发出去





这是因为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

观察VPN server上的路由信息

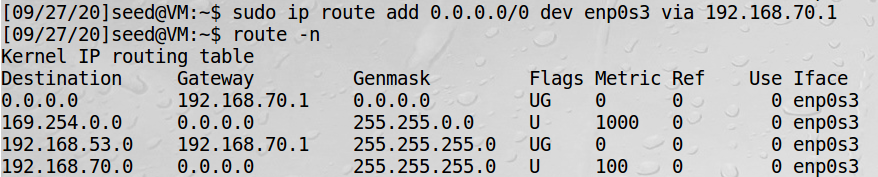


Host U上修改后的TUN端口IP地址为192.168.30.99，属于于192.168.30.0/24网络，经VPN服务器的路由查找，其应通过默认路由项0.0.0.0/0转发，即通过enp0s3端口，但实际中ICMP报文却从tun0端口来，因此服务端会认为这是一个伪造报文从而丢弃

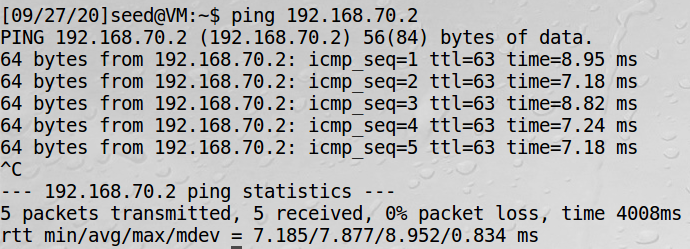
我们需要在VPN server上添加192.168.30.0/24与tun0端口关联的路由表项



在Host V上配置默认路由

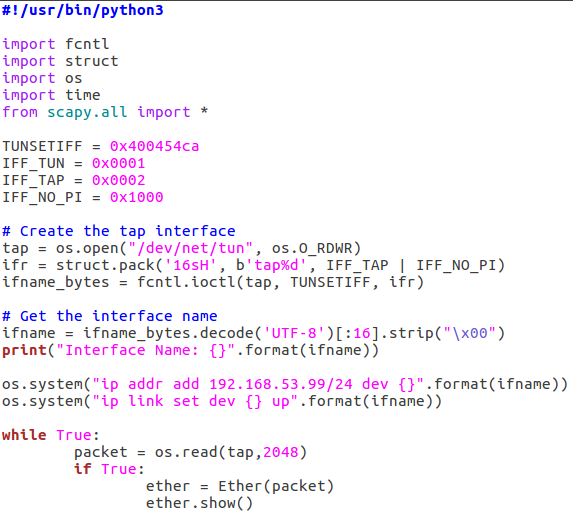


打开隧道，在Host U上ping Host V，可以ping通

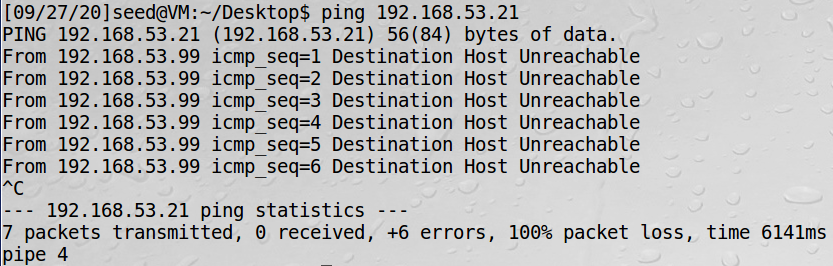


## Task 9: Experiment with the TAP Interface

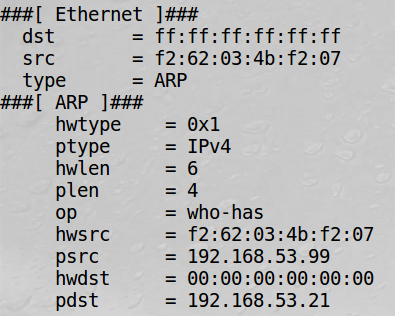
编写tap.py



在Host U上ping 192.168.53.21



然后查看python脚本的输出，可以看到TAP口确实收到了arp请求



TAP接口的内核端则连接到MAC层，TAP接口在MAC层就接管数据包，需要填写目的IP 192.168.53.21的MAC地址，所以发送ARP请求查询MAC地址。但由于这个地址是并不存在的虚拟网络，所以ARP请求并不能收到响应，也就导致ICMP数据包的滞留