**Lab7-report**

**VPN Tunneling Lab**

**Name:范心宇**

**Student Number:57117129**

虚拟机A：Host U，10.0.2.4；

虚拟机B：VPN server，10.0.2.5，192.168.70.1；

虚拟机C：Host V，192.168.70.2。

**Task 1: Network Setup**

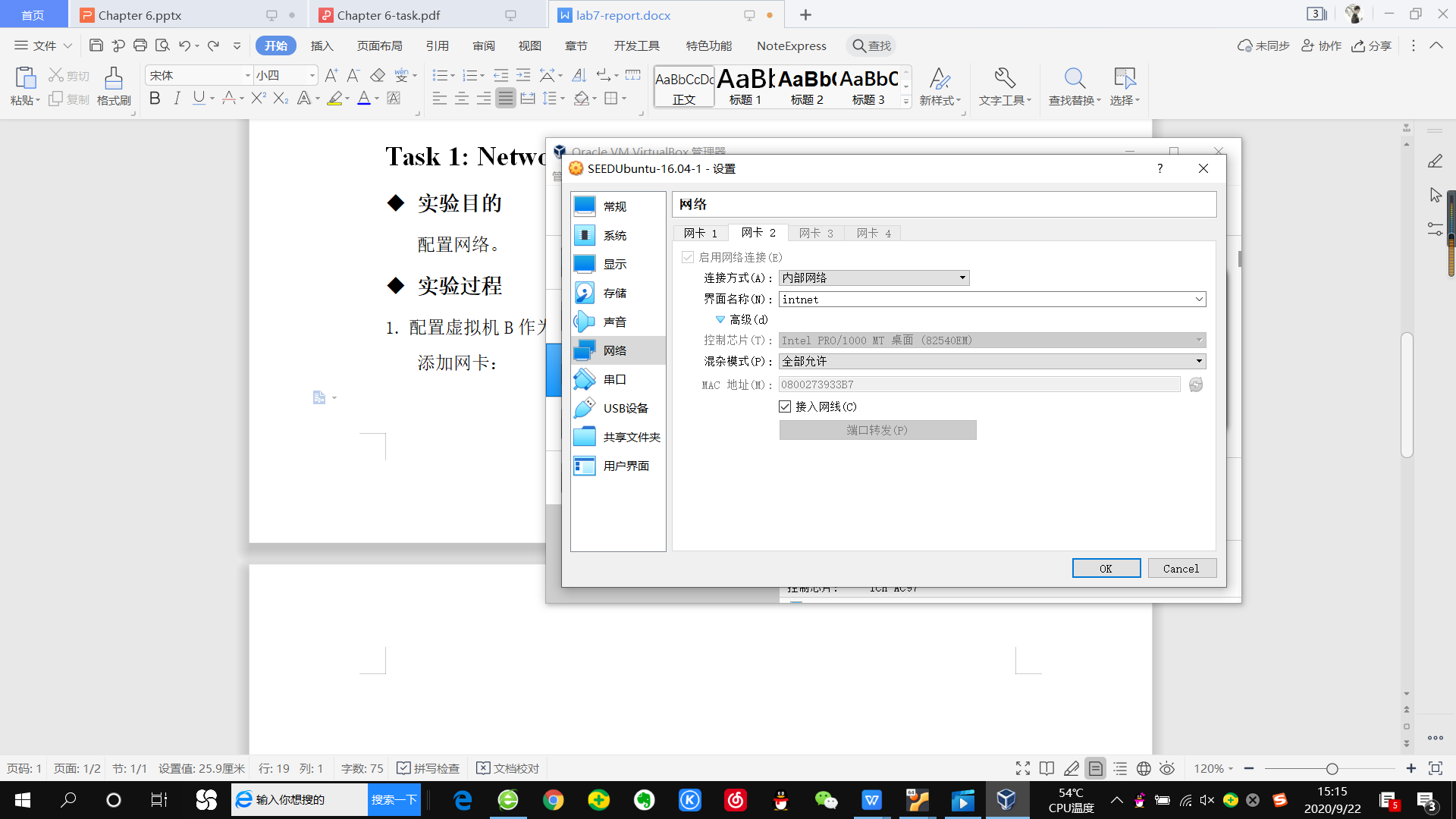
* **实验目的**

配置网络。

* **实验过程**

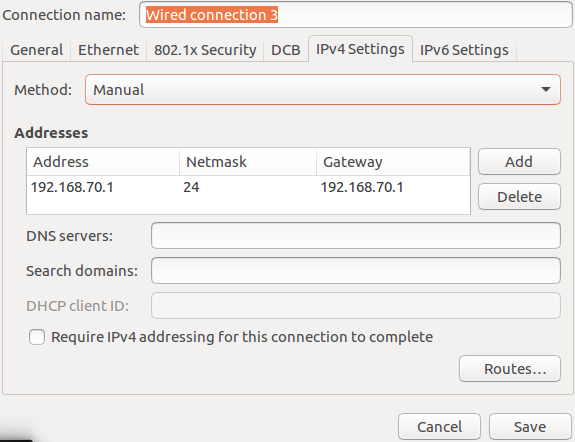
1. 配置虚拟机B作为VPN server。

添加网卡：



该网卡使用内部网络模式，作为连接VPN的网络接口设备。

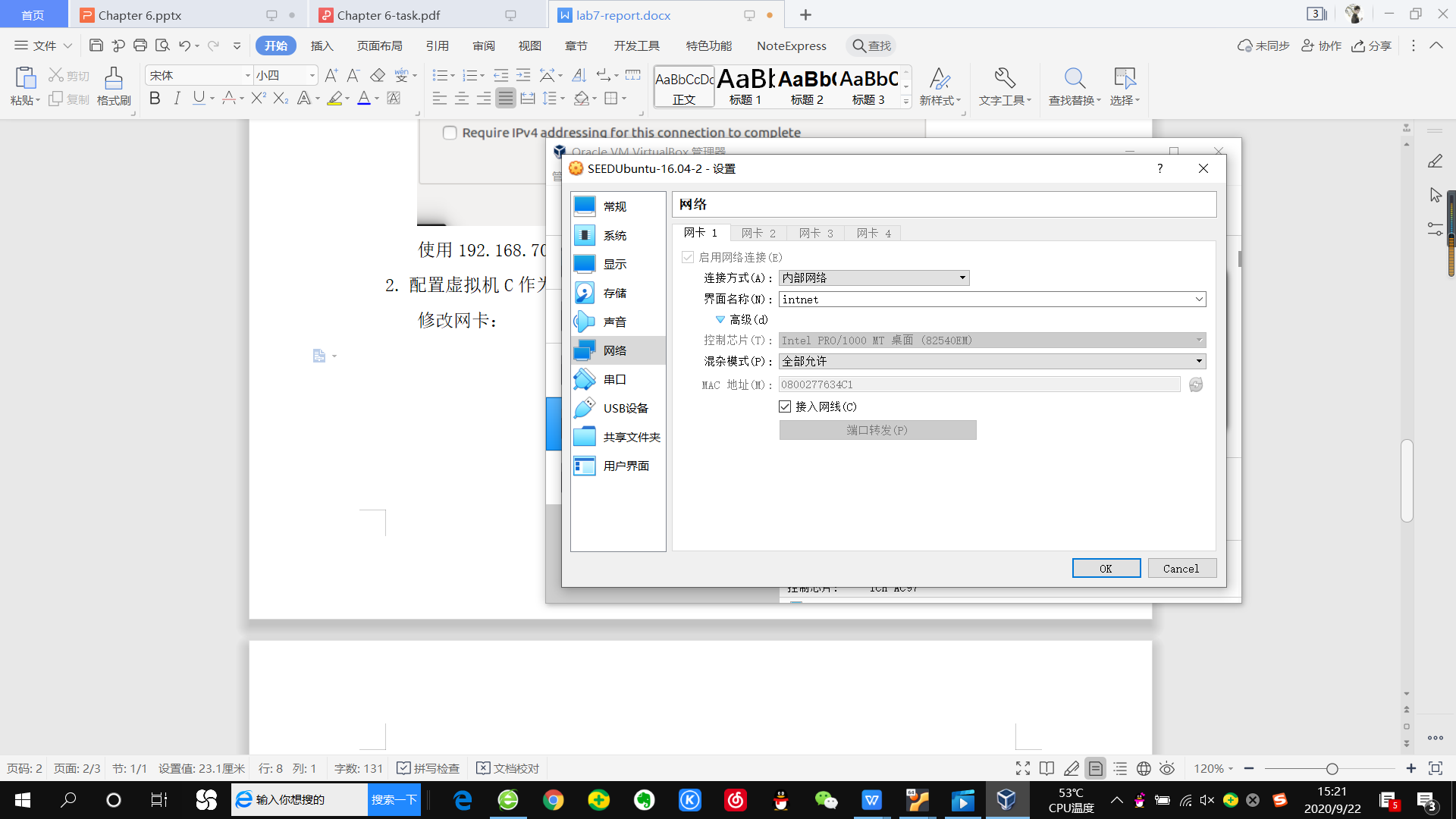
进入虚拟机配置该网卡连接信息：



使用192.168.70.0/24作为VPN网段。

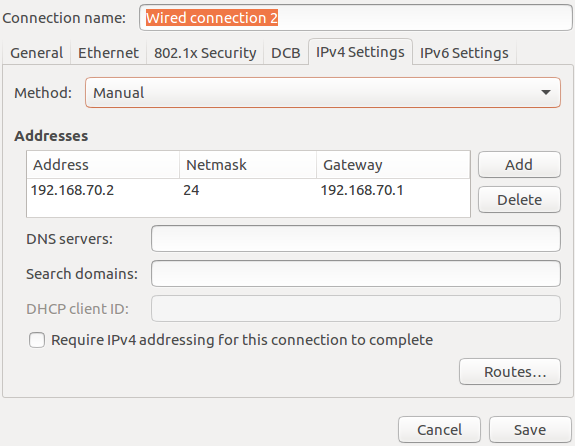
1. 配置虚拟机C作为Host V。

修改网卡：



将网卡修改为内部网络模式，只能与同处intnet内的网卡通信。

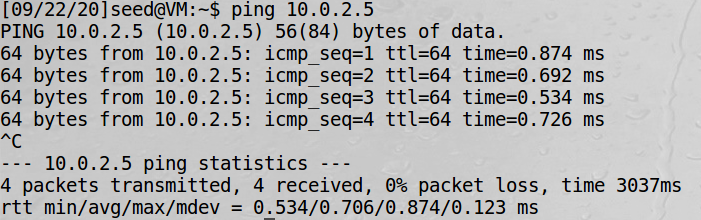
进入虚拟机配置该网卡连接信息：



自身IP手动设置为192.168.70.2，网关采用VPN服务器IP 192.168.70.1。

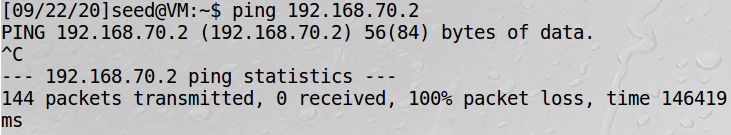
1. 测试配置后效果。

U ping VPN Server：



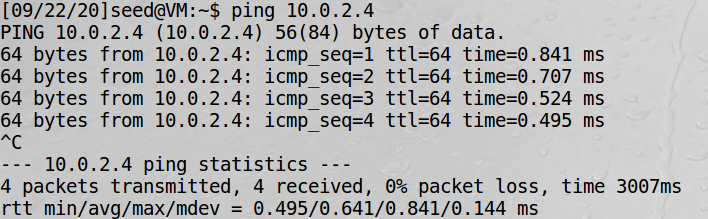
成功。

U ping V：



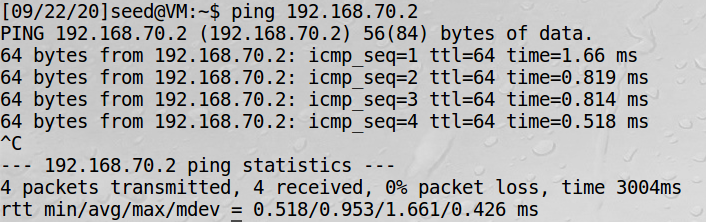
无法ping通，成功。

VPN Server ping U：



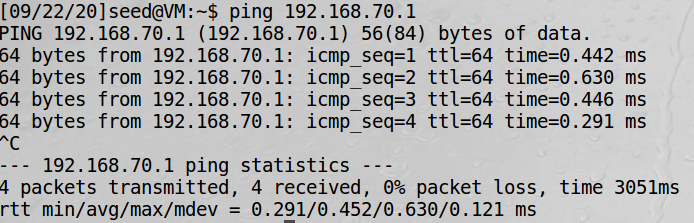
成功。

VPN Server ping V：



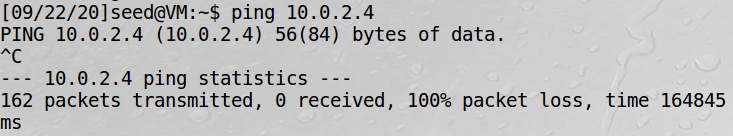
成功。

V ping VPN Server：



成功。

V ping U：



无法ping通，成功。

* **实验结论**

所有配置都已完成。

**Task 2: Create and Configure TUN Interface**

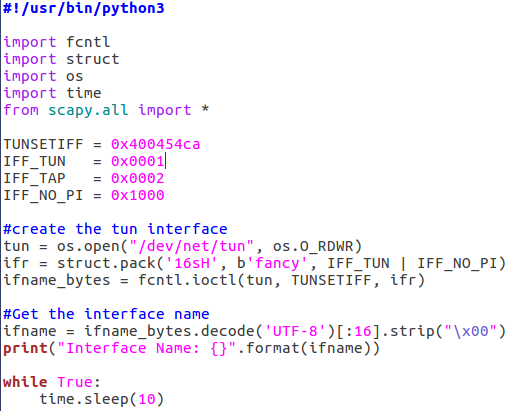
* **实验目的**

创建并配置TUN虚拟端口。

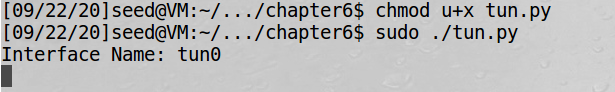
* **实验过程**

1. 在U上运行tun.py程序添加一个TUN虚拟接口。

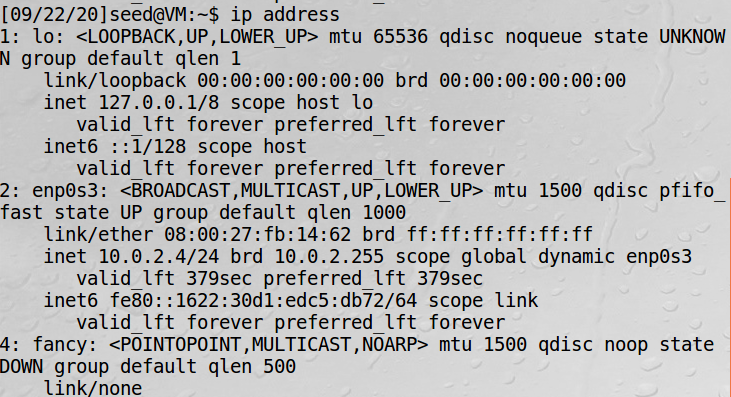
编写tun.py：



运行：



再打开一个terminal观察设备端口信息：



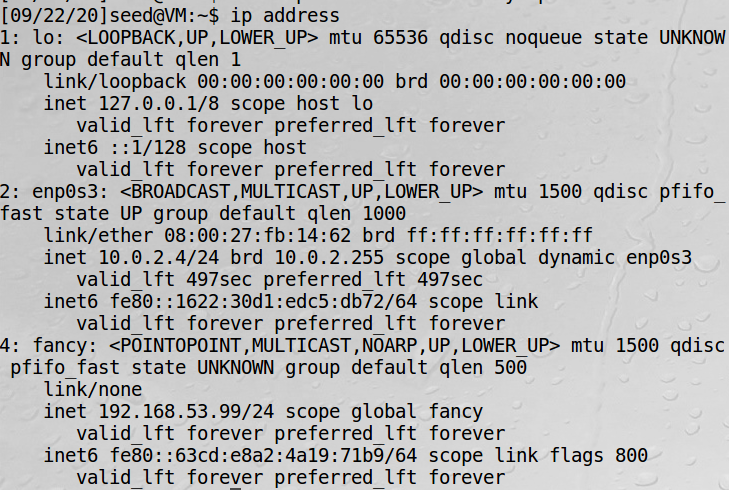
可以看到此时多出一个虚拟端口fancy。

1. 配置该虚拟接口。

为接口分配IP地址并启用：



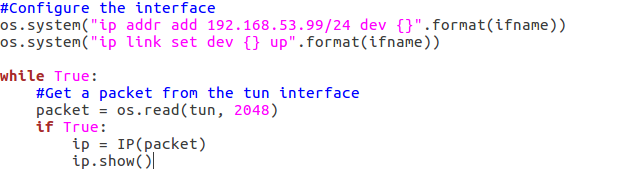
查看端口信息：



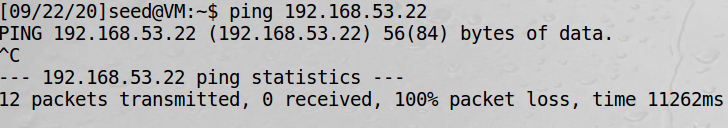
可以看到此时fancy端口拥有了IP地址，且状态变为了打开。

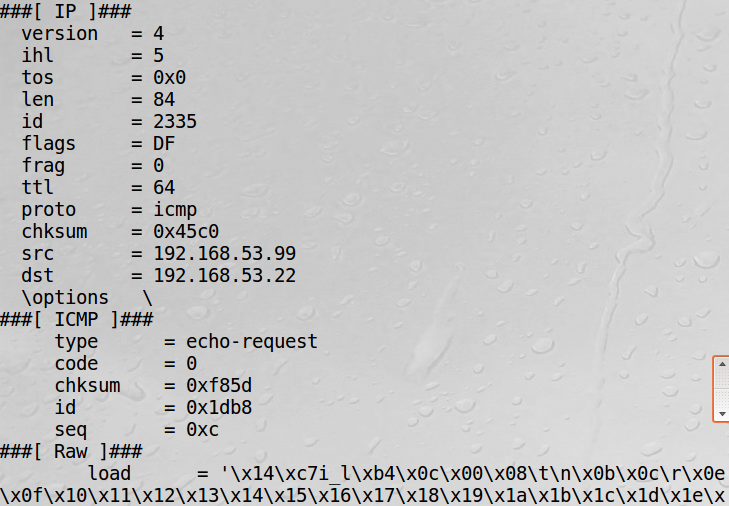
1. 从TUN端口读取数据。

添加代码：



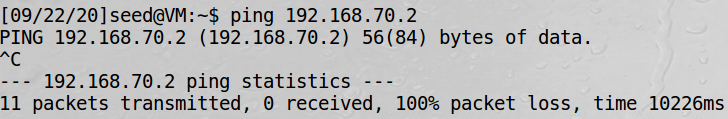
运行程序，ping 192.168.53.0/24网内地址进行测试：

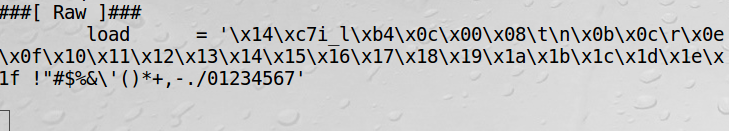




可以看到tun.py程序打印出了发往192.168.53.22的数据包，也就是说数据包在路由选择时被发给了我们的fancy虚拟接口。需要注意的是，数据包的源地址被修改成了fancy端口的IP地址，而不再是enp0s3端口的10.0.2.4。

ping 192.168.70.0/24网内地址进行测试：

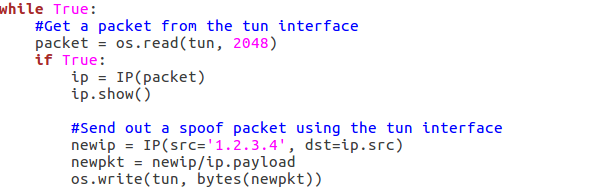




无事发生，可见此时192.168.70.0/24这一网络还不在fancy的转发范围内。

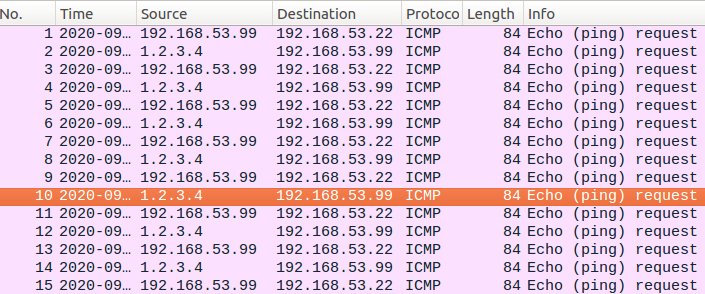
1. 向TUN端口写入数据。

添加代码：



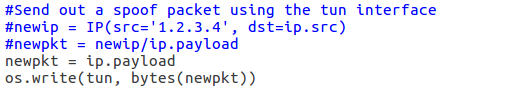
运行程序，观察wireshark：



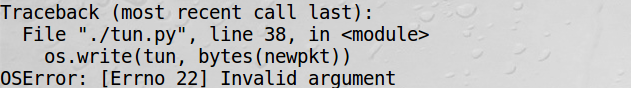


可以看到既有从192.168.53.99发往192.168.53.22的ICMP请求报文，又有从1.2.3.4发往192.168.53.99的ICMP请求报文，TUN运行成功。

随意写入数据：



运行程序：



可以看到产生了“参数不合法”的错误，因此可知只能向TUN中写入IP数据报。

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

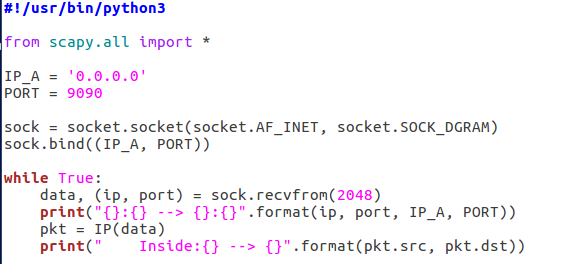
* **实验目的**

从主机U上使用TUNNEL向VPN服务器发送IP数据报。

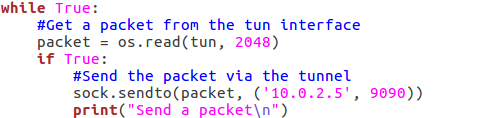
* **实验过程**

1. 编写VPN服务器上的监听程序。

在VPN服务器上运行tun\_server.py程序，该程序只是一个简单的UDP服务器程序，监听9090端口并打印出所有收到的数据：



1. 编写主机U上的隧道传输程序。

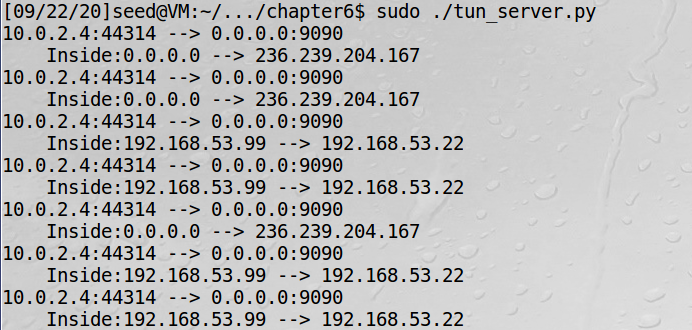


1. 运行程序。

ping 192.168.53.0/24网内任意IP地址：



观察VPN服务端输出：

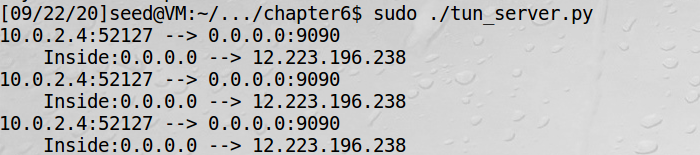


可以看到VPN服务器能够收到数据报，外层为10.0.2.4:44314发往自身9090端口的IP数据报，内层为192.168.53.99发往192.168.53.22的IP数据报，可见内层数据报通过IP隧道在主机U与VPN服务器间实现了通信。

ping 192.168.70.2内部主机V：



观察VPN服务端输出：



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，而没有从192.168.53.99发往192.168.70.2的数据报，可见目的地址为192.168.70.0/24的数据包不会被路由器交给TUN接口处理。

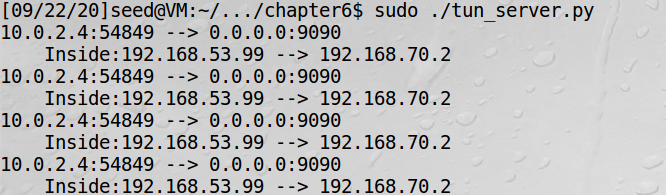
1. 配置路由。

在主机U上配置路由信息，将192.168.70.0/24网段的出口设为虚拟接口fancy：



再次测试：





可以看到此时服务器端可以收到从主机U上发往192.168.70.2的报文。

* **实验结论**

可以看到通过TUN客户端程序，可以把自身端口管辖范围内的数据包作为负载重新封装到UDP或TCP中，再从协议栈中发送出去，服务端可以接受到封装后的UDP数据包，去掉头部后得到原本的IP数据报。这里可以看出，当主机上有多个端口多个IP地址时，数据包从哪个端口发送就使用该端口的IP地址。

**Task 4: Set Up the VPN Server**

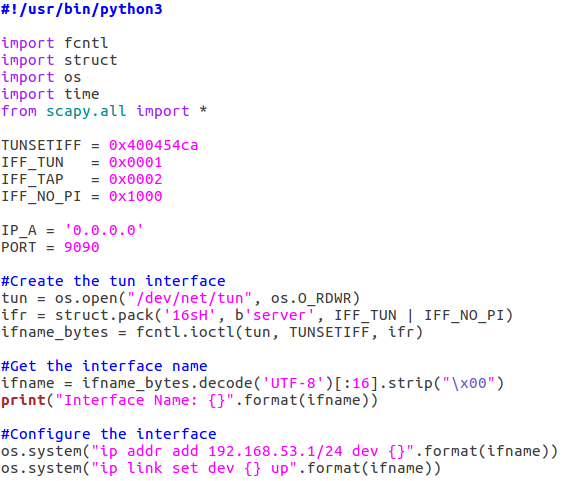
* **实验目的**

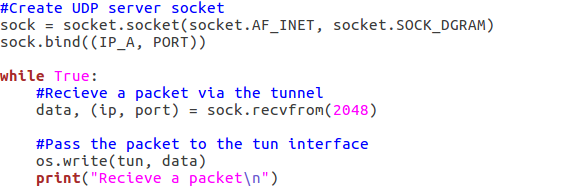
配置VPN服务器。

* **实验过程**

1. 修改tun\_server.py，使其成为TUN服务端程序。

编写tun\_server.py代码：



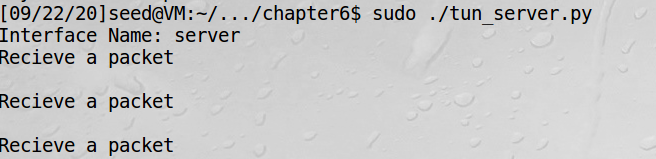


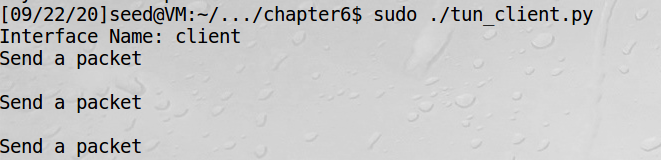
1. 打开VPN服务端的路由选项。



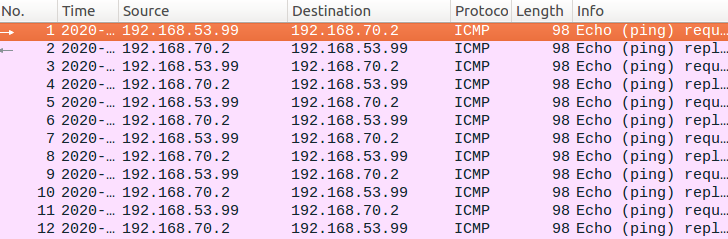
1. 运行程序。







观察主机V上wireshark：



有大量发给自身（192.168.70.2）的ICMP请求报文，来自192.168.53.99。

* **实验结论**

可以看到配置后VPN服务器成功将主机U发来的报文处理后转发给了主机V。这里主机U在路由表中将V所处网络设置为交付TUN端口转发，这样TUN接手报文后交给自身程序的socket套接字，再将IP数据报从协议栈中封装一遍，此时外层已经是物理端口源IP和VPN服务器物理端口目的IP了。因此该封装后的报文能够顺利通过网络由VPN服务器接收，报文经过协议栈处理交付服务端TUN进程的socket套接字，套接字再将数据部分交给TUN接口直接传递到路由选择处。路由选择获得是内部原IP数据报，即可按照路由表做出判断，将报文从enp0s8接口发给主机V，完成单向的IP隧道通信。

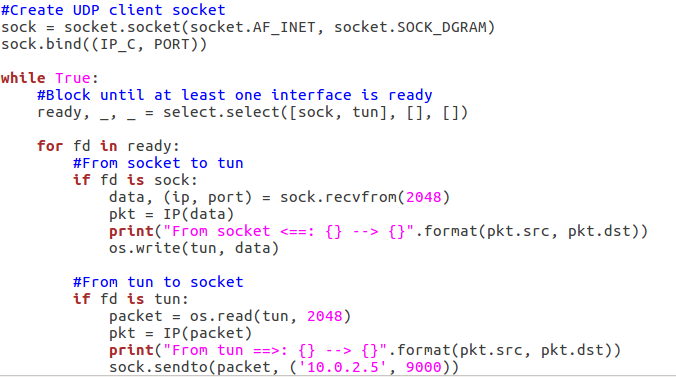
**Task 5: Handling Traffic in Both Directions**

* **实验目的**

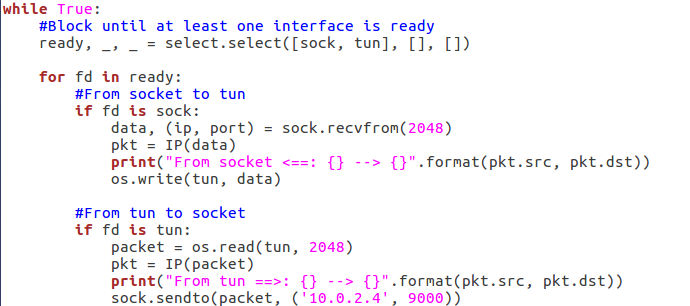
进行双向IP隧道通信。

* **实验过程**

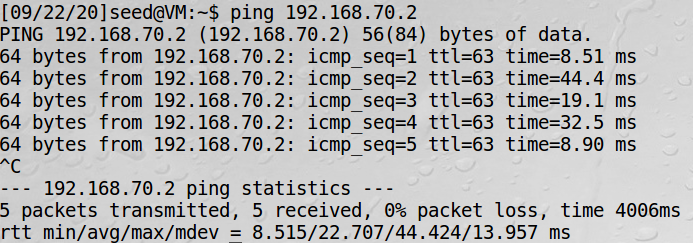
1. 修改客户端程序。



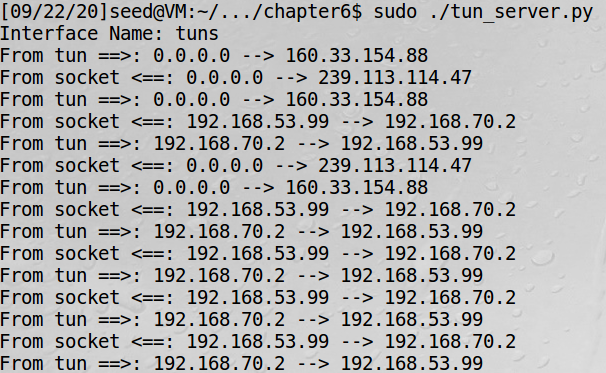
1. 修改服务端程序。



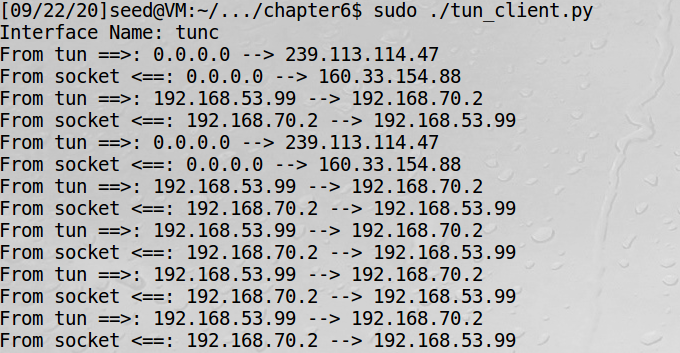
1. 运行并测试。



可以看到此时能够ping通内网192.168.70.2。



服务端能够看到192.168.53.99与192.168.70.2的通信交互过程。



客户端能看到类似的情况。

* **实验结论**

在TUN程序中同时控制TUN虚拟网络设备接口和socket套接字接口，可以实现数据包的再封装，以及IP隧道技术。

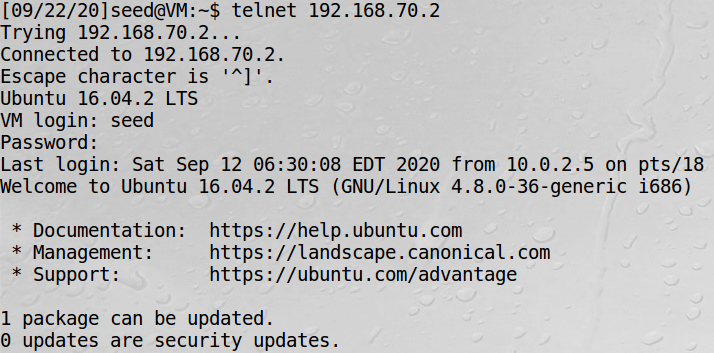
**Task 6: Tunnel-Breaking Experiment**

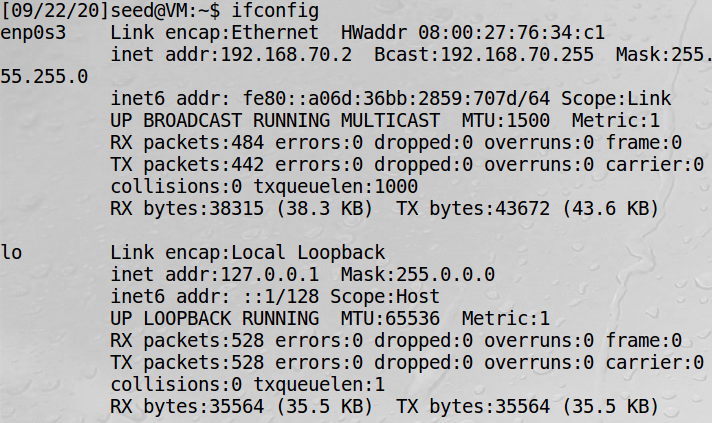
* **实验目的**

使用IP隧道进行telnet通信过程中，通过关闭tun\_client.py或tun\_server.py程序终止隧道，观察此时的telnet情况，然后再重新打开进行观察。

* **实验过程**

1. 打开隧道并进行telnet通信。





此时能够正常通信。

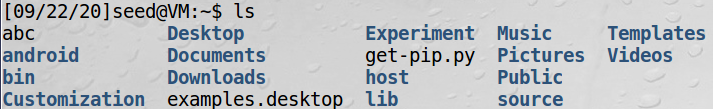
1. 关闭隧道。

关闭tun\_server.py：

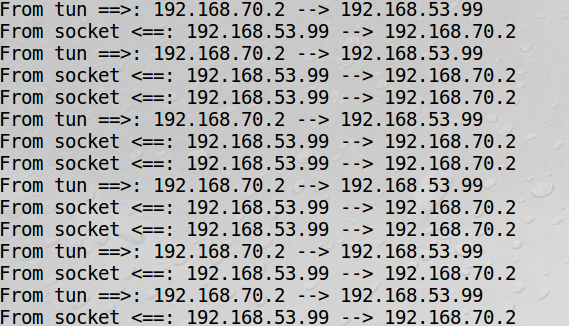


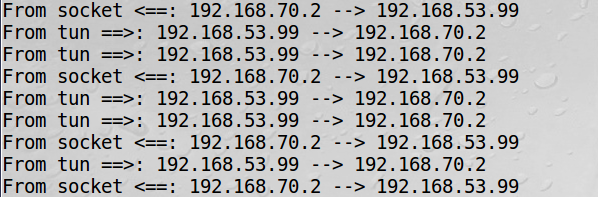
无法再输入命令，且客户端与服务端不再有报文传输。

1. 重新打开。

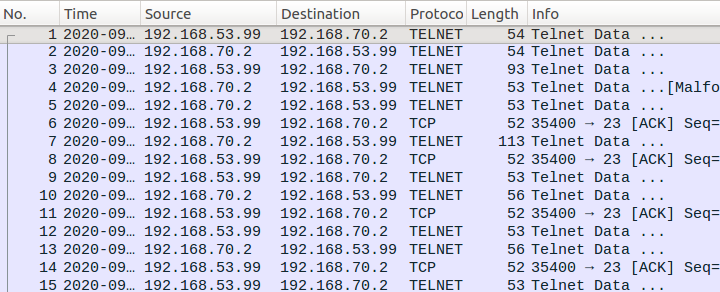


等待一段时间后重新连接上。





客户端和服务端都突然传送大量报文。



观察wireshark，可以看到双方重新连接了telnet服务。

* **实验结论**

可以看到，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。

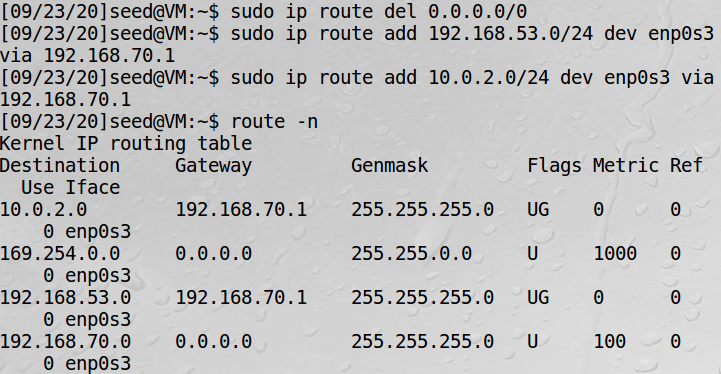
**Task 7: Routing Experiment on Host V**

* **实验目的**

在主机V上进行路由配置实验。

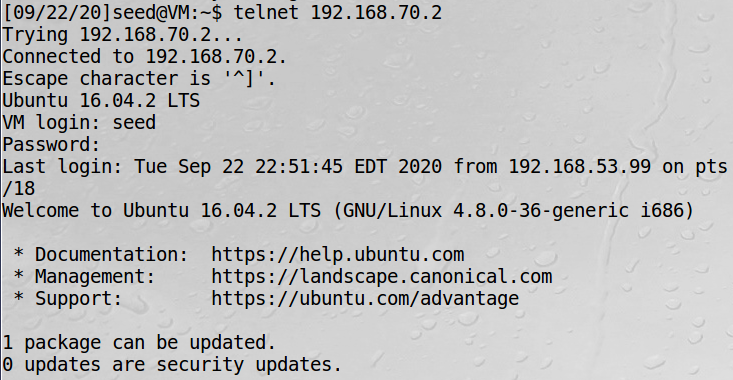
* **实验过程**

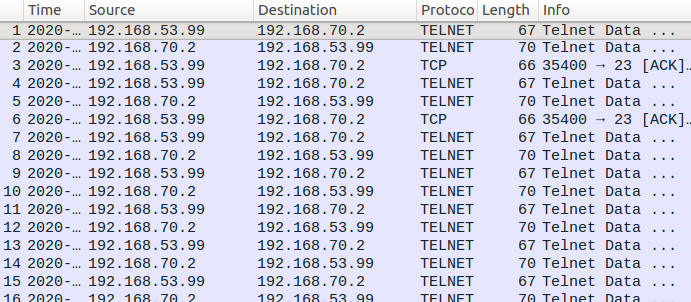
1. 配置路由。



1. 打开tunnel进行测试。

主机U向主机V发起telnet：





可以成功通信。

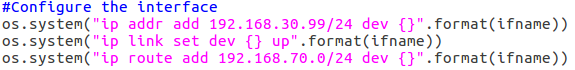
**Task 8: Experiment with the TUN IP Address**

* **实验目的**

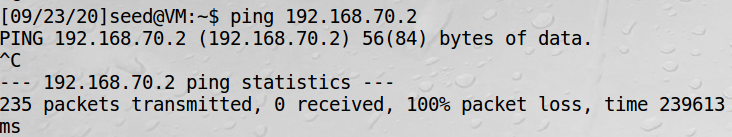
修改TUN接口IP，使客户端与服务器不处于同一网段再进行测试与配置。

* **实验过程**

1. 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。

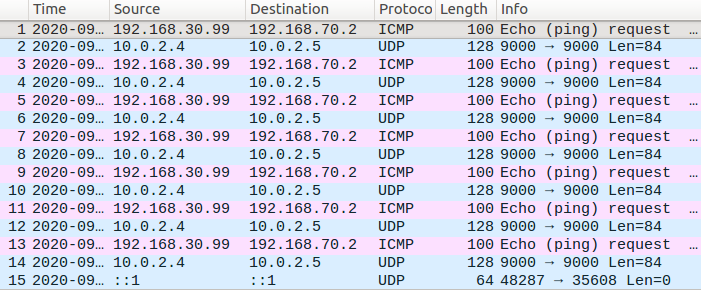


1. 打开tunnel进行测试。



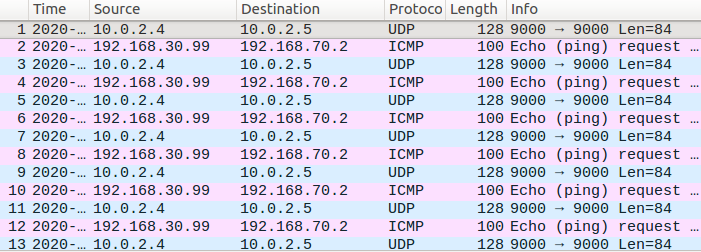
可以看到此时无法ping通192.168.70.2。

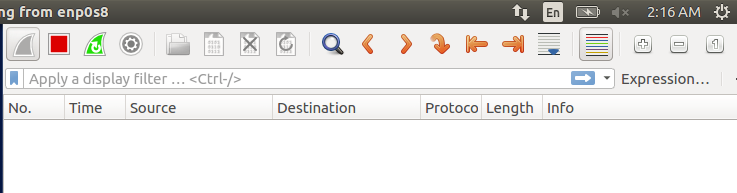
观察主机U上wireshark：



可以看到主机U上既有从enp0s3发出的10.0.2.4->10.0.2.5的UDP数据报，也有从tunc发出的192.168.30.99->192.168.70.2的ICMP数据报。

观察VPN服务器上wireshark：



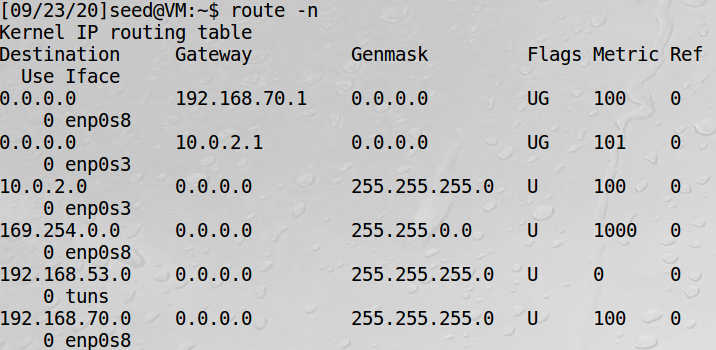


可以看到enp0s3和tuns端口都分别有收到UDP和ICMP报文，但enp0s8端口却没有报文发出，也就是说VPN服务器在内层的ICMP报文进行路由选择时，并未将其从enp0s8端口转发出去。

1. 解释与修正。

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服务器上路由信息：



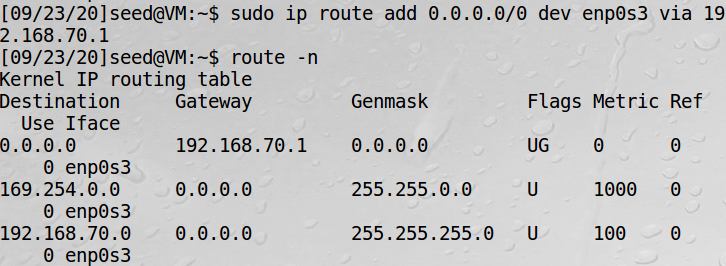
在本实验中，主机U上修改后的TUN端口IP地址为192.168.30.99，来源于192.168.30.0/24网络，经VPN服务器的路由查找，其应通过默认路由项0.0.0.0/0转发，即通过enp0s3端口，但实际中ICMP报文却从tuns端口而来，因此服务端会认为这是一个伪造报文从而丢弃。

我们需要在VPN服务器上添加192.168.30.0/24网络与tuns端口关联的路由表项：

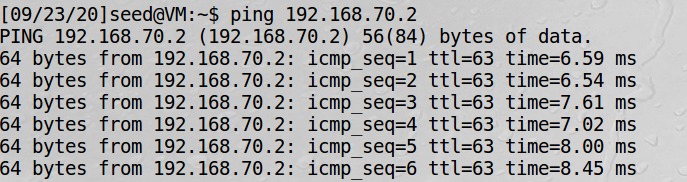


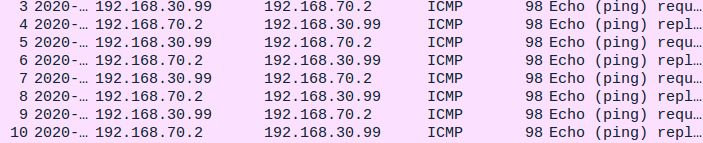
实际上，即使没有RPC机制，如果不添加上述表项，虽然主机V能够收到ICMP请求报文，但返回的应答报文在经过VPN服务器时会被发往默认网关（192.168.70.1或10.0.2.1），并不能够达到主机U。

而在主机V上仍使用默认路由设置：



现在再运行tunnel进行通信：





可以看到此时能够成功ping通192.168.70.2了。

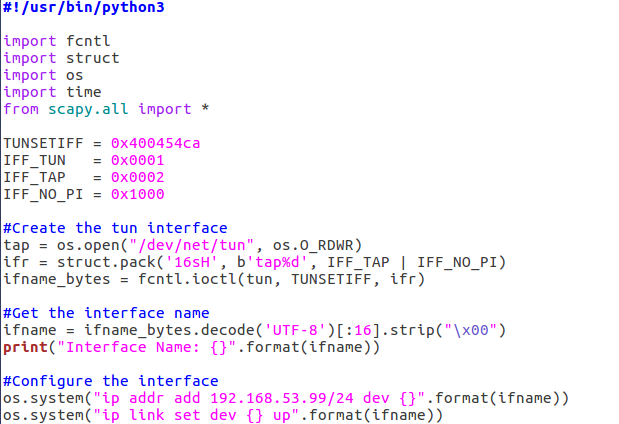
**Task 9: Experiment with the TAP Interface**

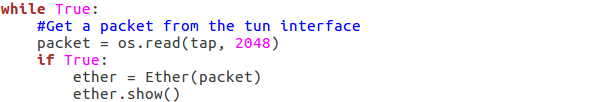
* **实验目的**

搭建基于TAP端口的IP隧道进行通信。

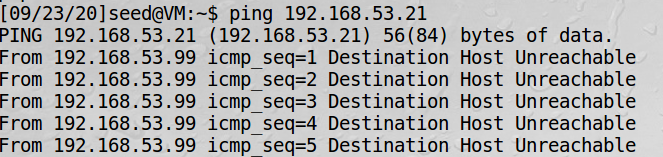
* **实验过程**

1. 编写代码。

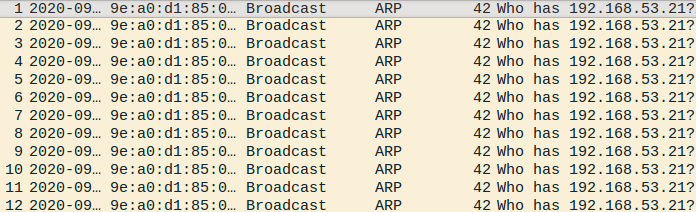




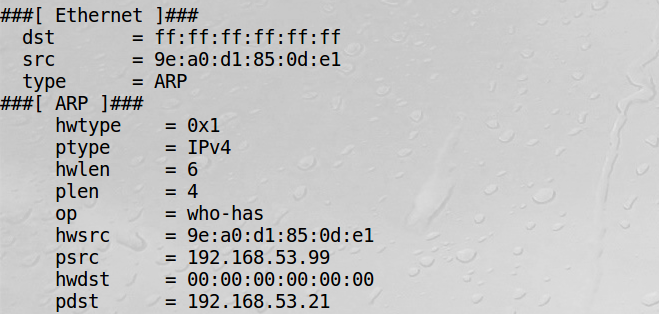
1. 测试。



观察wireshark：



tap0端口在向外发送ARP请求，查询192.168.53.21的MAC地址，而enp0s3则没有任何信息发送。可见TAP在MAC层就接管即将发送的VPN数据包，需要填写目的IP 192.168.53.21的MAC地址。该网络在路由表中是自己的直连网络，所以就发送ARP请求查询MAC地址，但由于这本身是虚拟网络，并不存在，所以ARP并不能收到响应，也就导致ICMP数据包滞留在TUN端口中不能通过socket发送出去。



我们可以观察到此时TUN接口抓到的包，是ARP请求报文，请求的是192.168.53.21的MAC地址。

由此可见TAP对MAC层的处理更加复杂，可能需要使用TCP进行通信。