网安综合课程设计

实验报告

（57117105 杨哲君）

***VPN Tunneling Lab***

**实验环境：**

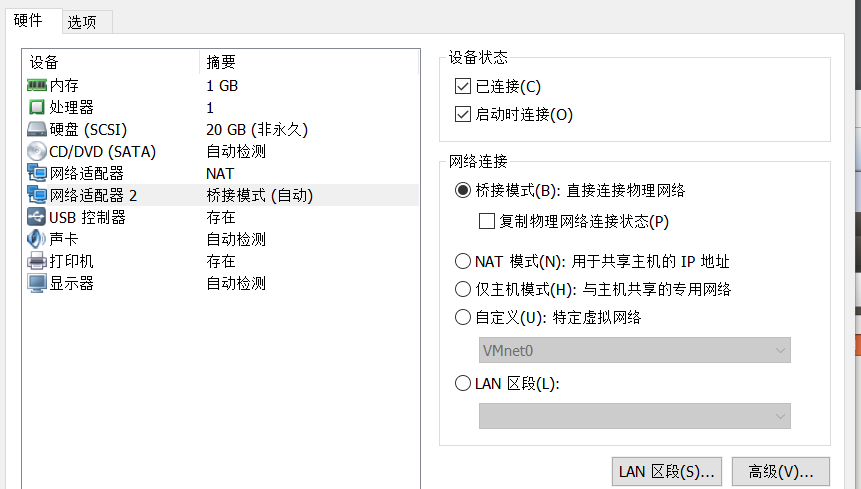
**Host U 192.168.80.137**

**VPN Server 192.168.80.138 10.0.0.1**

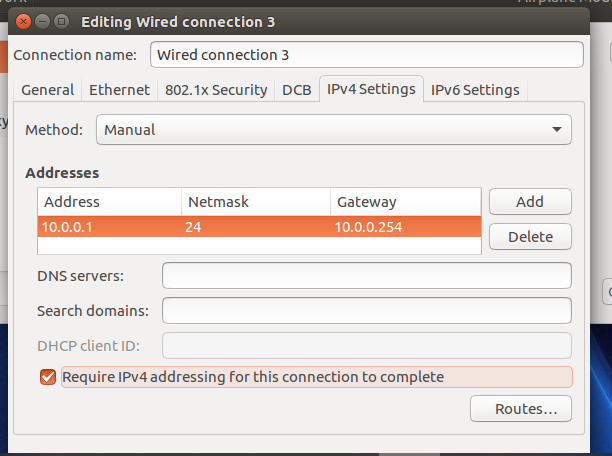
**Host V 10.0.0.2**

**Task 1: Network Setup**

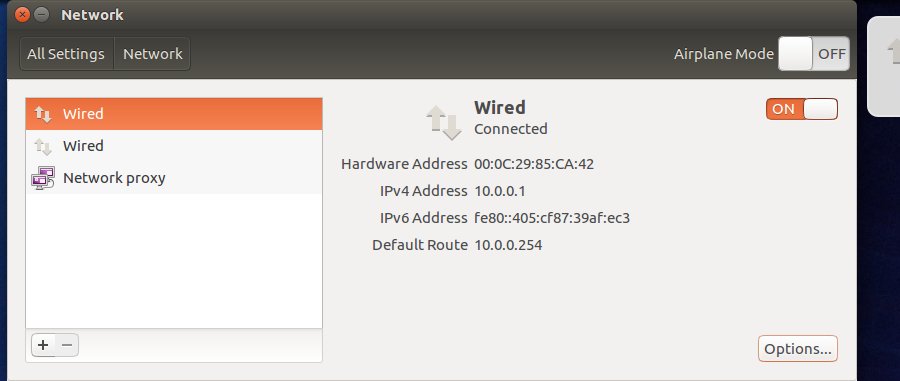
下面先搭建环境，为VPN Server添加网络适配器，设置连接模式为桥接模式。

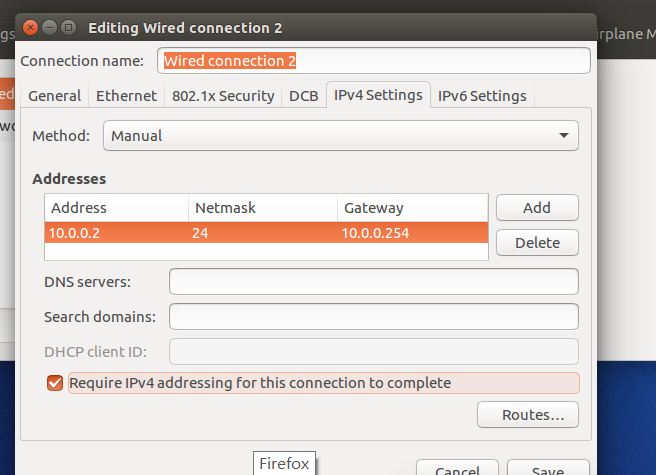


开启新的网卡后，打开系统设置中的Network，在options中设置其ip信息，如下图所示：



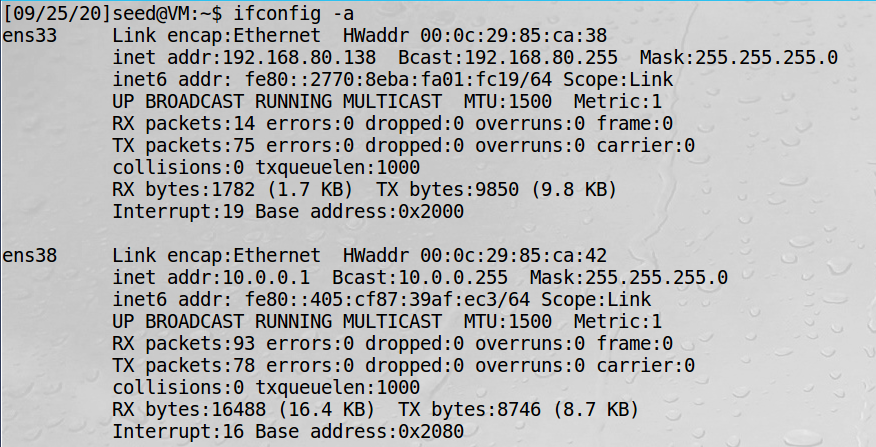
设置好之后是这样的：

在Host V中，设置网卡的连接模式为桥接模式，同样在Network中设置其ip信息：

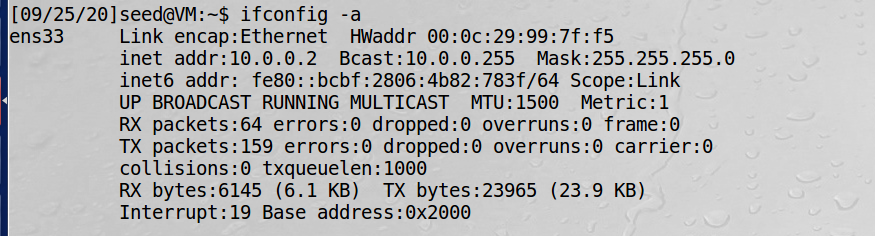


重启VPN Server和Host V。

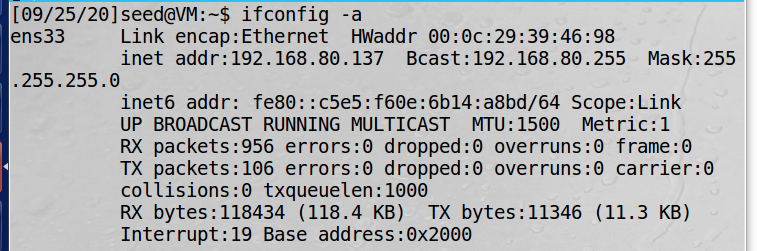
查看VPN Server的IP：



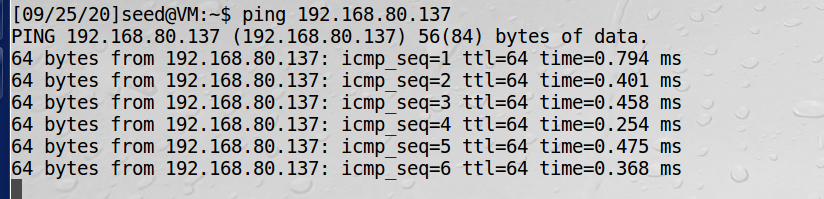
查看Host V的IP：

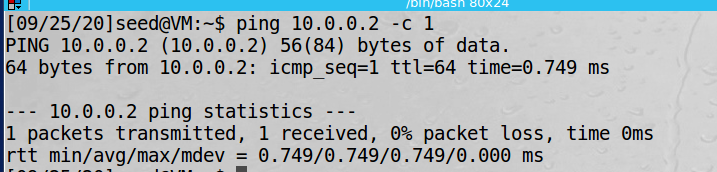


查看Host U的IP：

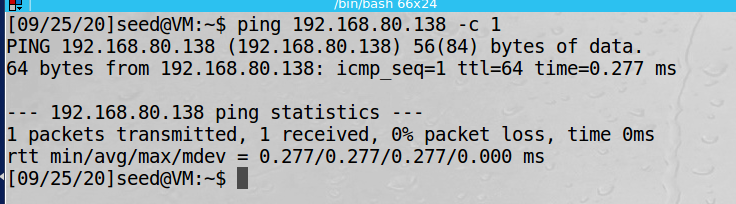


然后分别在VPN Server上ping Host U和Host V，发现都可以ping通。

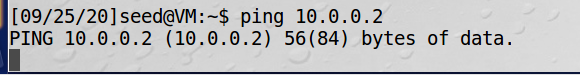




在Host U上ping VPN Server，可以ping通。



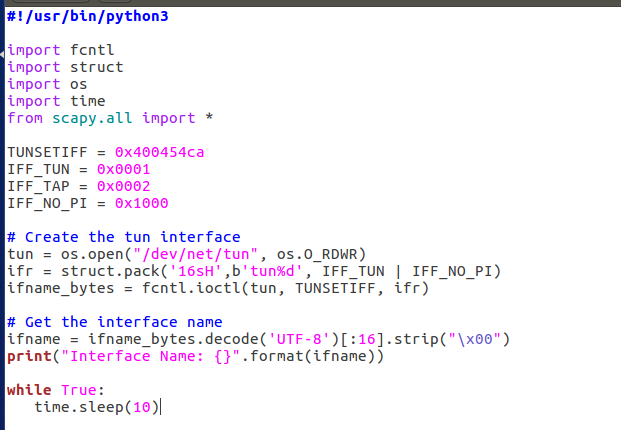
在Host U上ping Host V，发现ping不通。



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

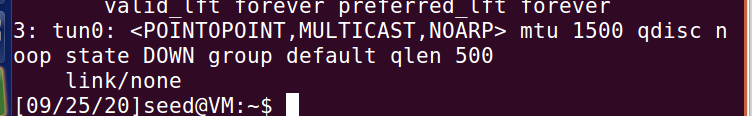
**Task 2.a: Name of the Interface**

在Host U中执行下面的Python程序：

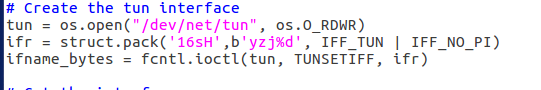




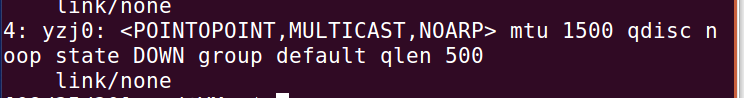
在另一个终端里使用命令ip address来查看各个网卡的IP地址。发现多了一个tun0网卡。



按实验手册的代码修改tun.py：

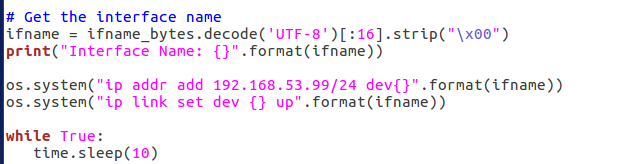


再次执行，增加的网卡名字变成了yzj0。

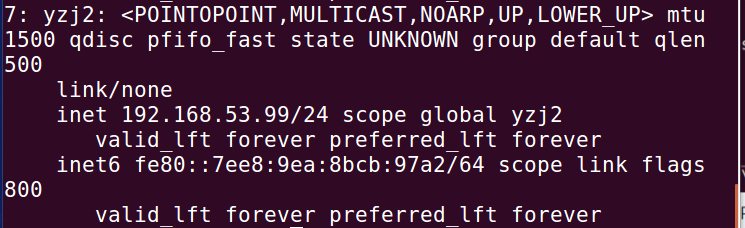


**Task 2.b: Set up the TUN Interface**

修改tun.py，为新建的tun网卡分配IP，启动网卡。

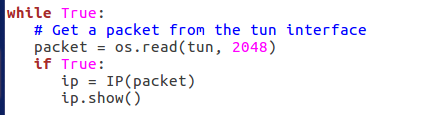


再次执行ip address，发现新建的网卡已经获得了ip地址：192.168.53.99.

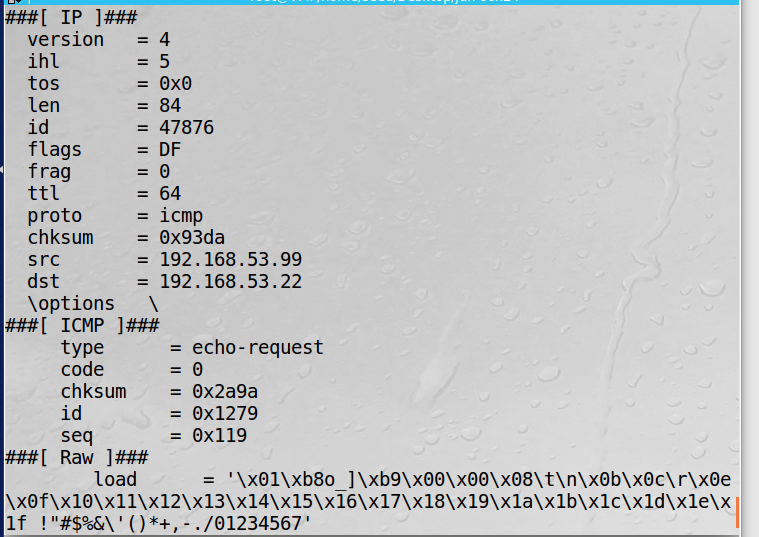


**Task 2.c: Read from the TUN Interface**

修改脚本，增加读取报文的功能。



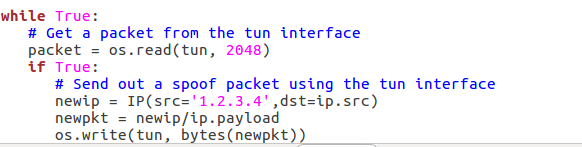
运行tun.py，在另一个终端ping192.168.53.22，发现tun网卡正在监听。打印出了src 为 192.168.53.99，dst 为 192.168.53.22的数据包。



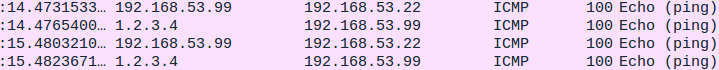
用Host U ping10.0.0.2，tun.py无反应，是因为此时发往10.0.0.2的报文不会经过yzj0虚拟网卡。

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

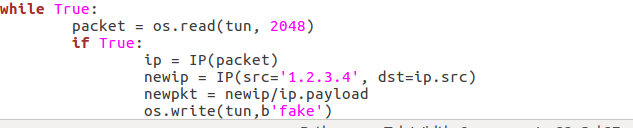
修改脚本。在结尾处增加伪造数据包的代码。



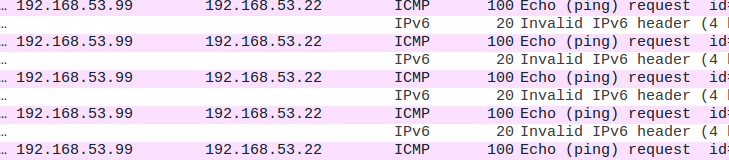
运行tun.py后，向192.168.53.22发送报文，用wireshark观察，发现抓到了源地址为1.2.3.4的伪造数据包。说明脚本可以实现伪造并发送报文的功能。



将发送到网卡的数据改为fake，意为伪造数据，重复上述操作。

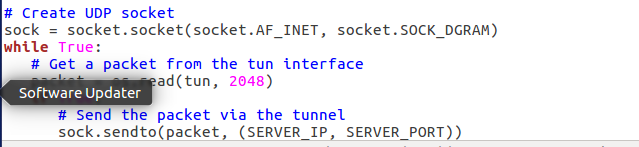


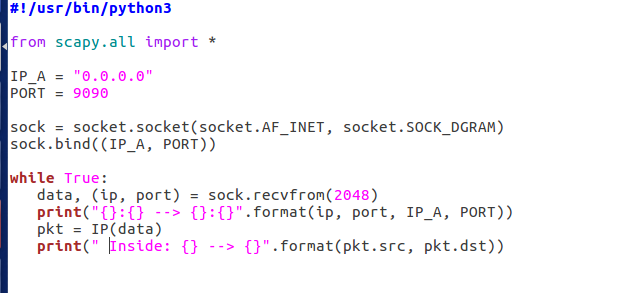
发现wireshark抓到一些没有意义的包，因为这些包的格式有问题，无法分析。



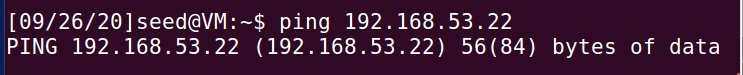
**Task 3: Send the IP Packet to VPN Server Through a Tunnel**

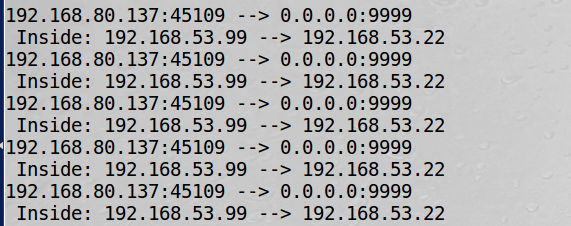
编写tun\_client.py和tun\_server.py程序：





从Host U上ping192.168.53.22，查看VPN Server的输出：



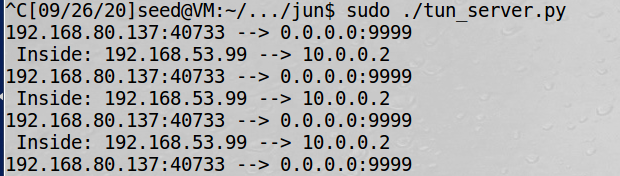


可以Ping通，因为目的地址属于192.168.53.0/24，所以系统的路由表会将ICMP数据包传递给yzj0网卡，客户端程序可以从/dev/net/tun中读取ICMP数据包，然后封装发往Server，实现了隧道的功能。

在Host U中添加路由规则，实现将发往10.0.0.2/24网段的数据包从yzj0发出。

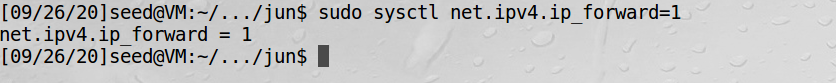


再次ping10.0.0.2，现在VPN Server可以监听到Host U发往V的报文了。

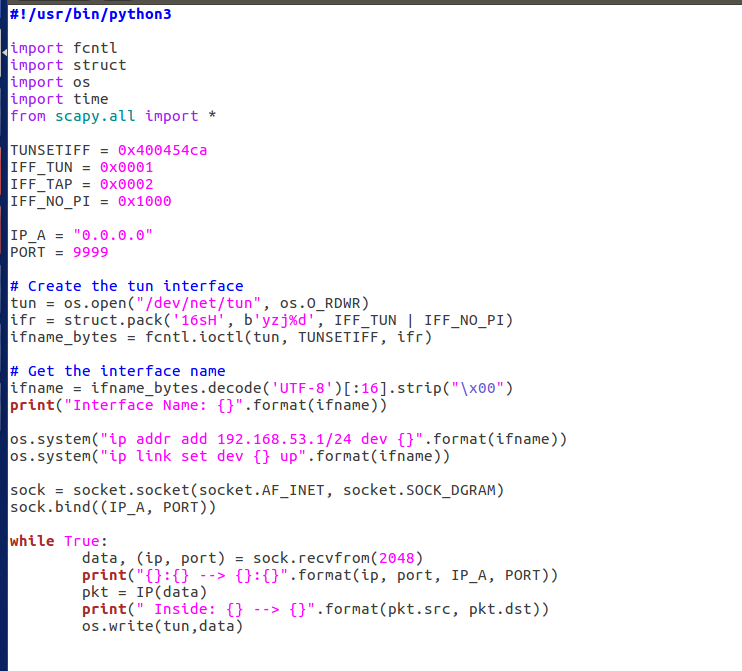


**Task 4: Set Up the VPN Server**

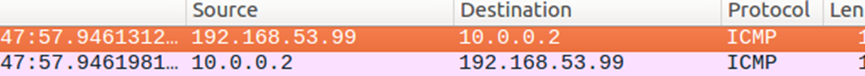
首先设置VPN Server的路由转发功能成开启状态。



编写下面的tun\_server.py程序，并执行。



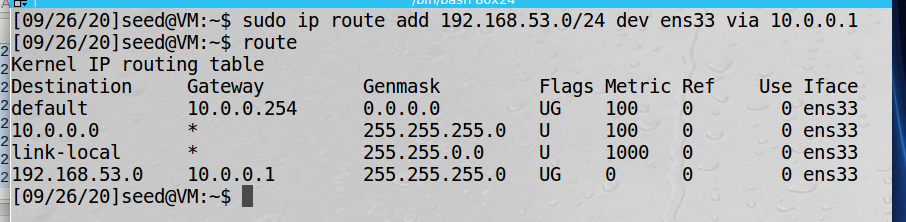
用Host V 中的wireshark监听，在Host U中ping Host V，抓到由Host U通过tun隧道发送到本来无法联通的Host V上的数据包。



**Task5: Handling Traffc in Both Directions**

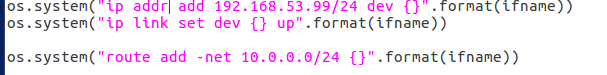
**Step1：**

在Host V上添加一条路由，可以使发往192.168.53.0/24的流量转向10.0.0.1。



**Step2.修改tun\_client.py**

在tun\_client.py中，添加一条代码，设置其虚拟网卡的IP和路由，让发往10.0.0.0/24的数据经过tun。



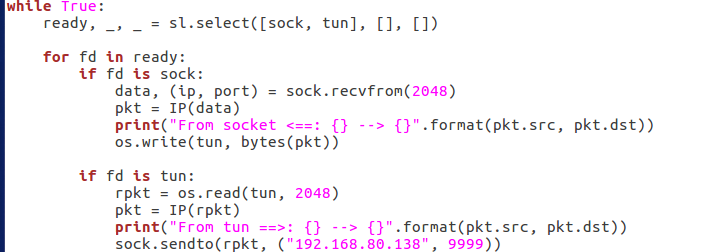
使用系统调用的select。



参考实验手册的代码，进行改写：

sock 里的消息是 VPN 服务端发送的报文，UDP 报文的数据即为真正的报文，将其交给tun，由于此时报文的目的 IP 即为 tun 的 IP，不会继续转发；

tun 里的消息是从客户端发出的消息，需要将其封装在 UDP 报文里，发送给 VPN 服务器。

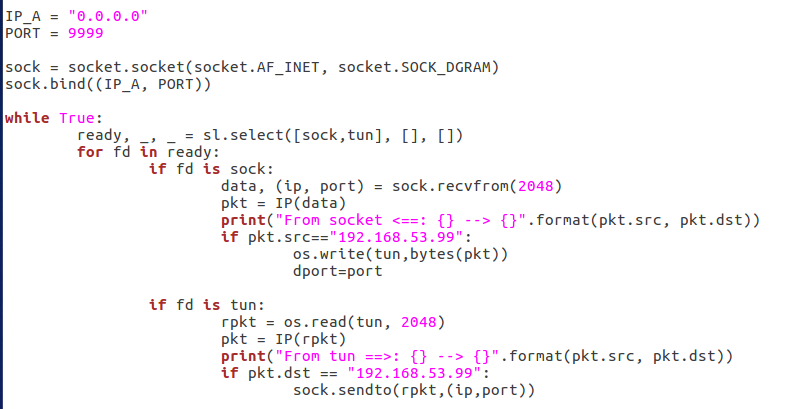


**Step3：修改tun\_server.py**

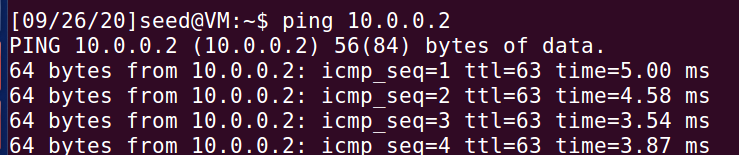
设置网卡并启动，开启UDP服务。

提取 sock 的消息的数据部分并封装成IP 报文，如果源地址是客户端虚拟网卡的地址，则将其交给自身的虚拟网卡 tun，tun 会根据目的地址转发给 V并记录客户端的端口号。

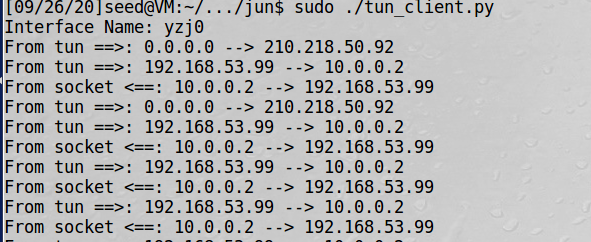
如果虚拟网卡 tun 收到的消息是来自 V 的，需要将其封装到 UDP 报文中，并发送给客户端。



用Host U打开另一个终端 ping Host V，发现可以ping通。



Host U中的输出：

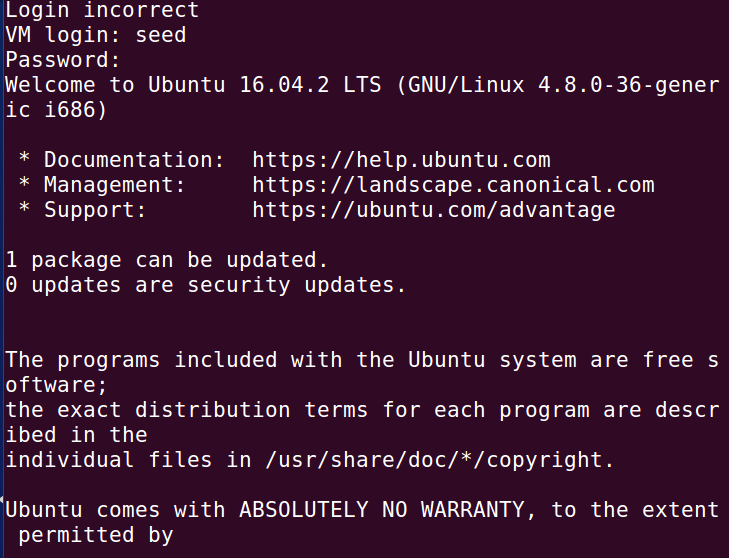


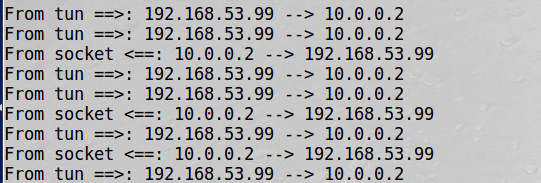
用wireshark可以观察到由Host U的虚拟端口发往Host V的数据包和回复的数据包。



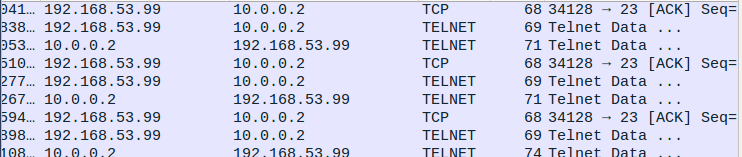
在Host U中的另一个终端对Host V发起Telnet请求，成功连接。





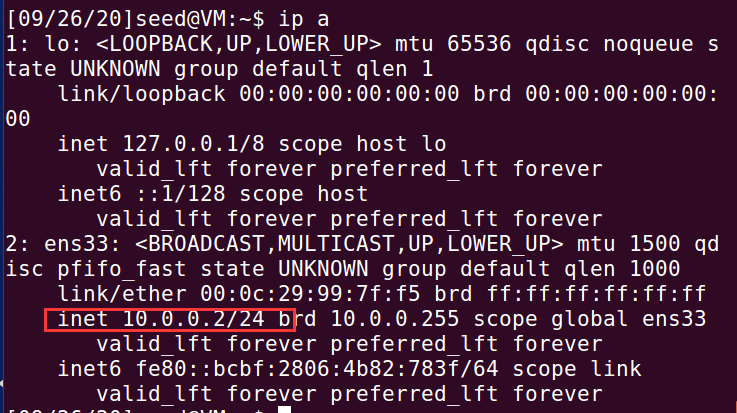


Wireshark中可以观察到Telnet数据包信息。

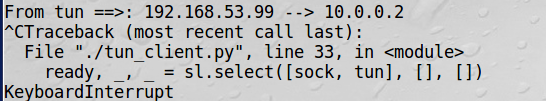


**Task6: Tunning-Breaking Experiment**

首先让Host U和Host V建立Telnet连接。此时输入命令一切正常。



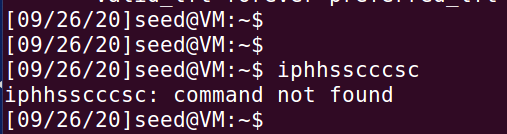
然后关闭VPN Server的程序。



此时，在Telnet界面输入任何命令都没有反应。



重新建立连接，Telnet界面出现了刚刚输入过的命令，并且此时可以正常输入命令，说明服务器在断开连接，键入的数据只能存在缓冲区，当重新建立连接后，若再输入数据，会将之前的数据也一起进行重传。



**Task7: Routing Experiment on Host V**

在Task5中已经添加过此条命令。



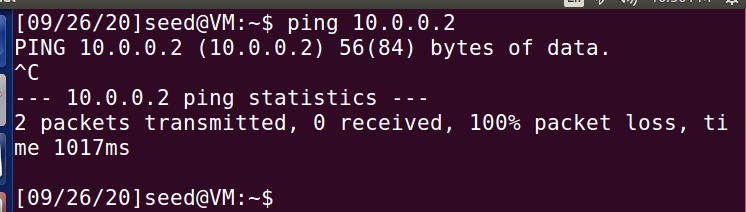
结果与task5，task6一致。

**Task8: Experiment with the TUN IP Address**

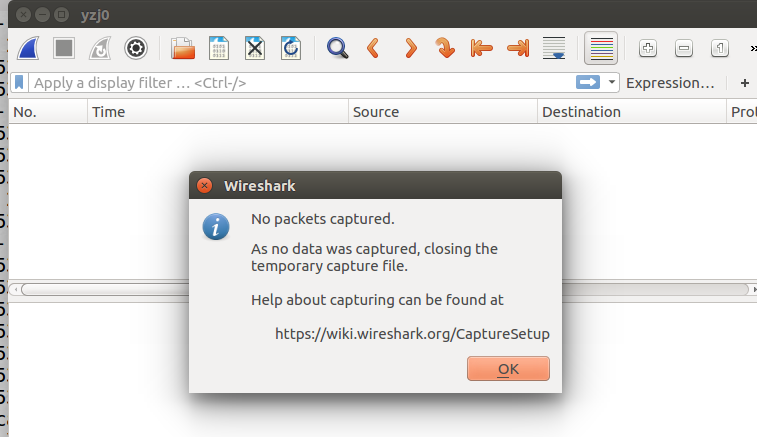
修改Host U的tun网卡的IP



运行程序，用Host U对Host V发起ping请求，此时ping不通。



监听VPN Server的yzj0网卡，没有收到数据包。

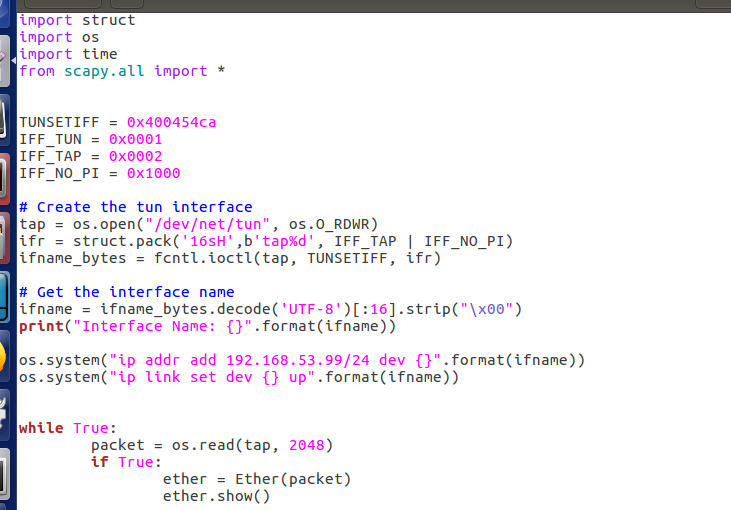


但是监听VPN Server的ens33网卡，却收到了来自Host U的UDP数据包，说明此时VPN Server的yzj0网卡还不能处理UDP里的IP报文。

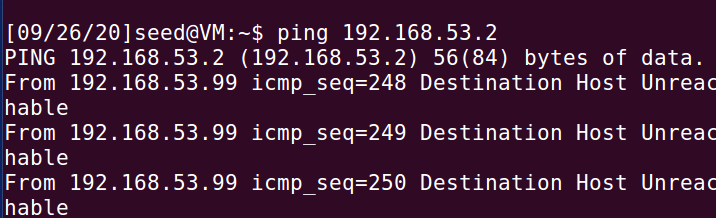


**Task9: Experiment with the TAP Interface**

TUN网卡只能捕捉 IP 层的数据，而 TAP 可以扩展到数据链路层。下面我们将 IFF\_TUN 改成 IFF\_TAP 即可创建 TAP 虚拟网卡。



Ping 192.168.53.2

发现tap网卡捕获到了数据链路层的数据。

