《计算机网络》
实验教材

[第 2.01 版]

庄元　许鼎鼎　编写

夏耐　顾庆　李文中　汇总

南京大学计算机科学与技术系

2017 年 2 月

目录

实验一 基本网络工具集使用和协议数据单元（PDU）观测

实验任务

1. 利用 VMWare 搭建一个由 5 台虚拟机组成的随机拓扑网络。要求该网络中至少有 2 个子网，两个路由器，本章末尾给出实验参考拓扑。
2. 利用 Wireshark 观测 PDU：

（1） 安装并打开 Wireshark，确保本机能否正常上网。

（2） 在本机 ping 系主页 cs.nju.edu.cn，用 Wireshark 记录本机和 cs.nju.edu.cn 相互通信的所有数据包，截图，简要分析数据包的各字段。

（3） 在本机打开浏览器，访问 www.nju.edu.cn，用 Wireshark 记录本机和 www.nju.edu.cn 相互通信的所有数据包，截图，简要分析数据包的各字段。

实验报告格式

按要求完成实验，并写出实验报告。实验报告格式如下，同学可以根据内容进行修改。说明是为了更详细的解释各个项目，提交的报告中无需包含。

实验目的	
网络拓扑配置	说明：填写附表 1，并绘图说明
路由规则配置	说明：写明输入的命令
数据包截图	说明：用 wireshark 抓包，并截图
协议报文分析	说明：对抓取的数据包进行字段分析

附表 1：

节点名	虚拟设备名	ip	netmask
Router0		eth0:	
		eth1:	
Router1		eth0:	
		eth1:	
PC1			
PC2			
PC3			
PC4			
PC5			

背景知识

本系列实验选用 Linux 操作系统作为主要实验平台。通过虚拟机软件在一台计算机实体

上模拟出一个小型网络环境，完成实验。

1 Linux

Linux 是由 Linus Torvalds 最初开发的操作系统内部核心程序，即内核（kernel）的标识。而所有的基于 Linux kernel 的类 UINX 操作系统也都被称作 Linux。相较于 Windows 操作系统在图形用户界面上的巨大优势，Linux 操作系统在网络应用上有很好的表现。这也是我们选用 Linux 操作系统作为主要实验平台的原因之一。

2 Linux 下的常用网络命令有

ifconfig 、ping、ip、netstat 、netconfig、nc、tcpdump、telnet、ftp 、route、rlogin 、rcp 等，我们简要介绍其中比较基础的 ifconfig 和 ping 命令。

ifconfig 和 MS-DOS 下的 ipconfig 命令相似，用以显示和设置网络设备。

语　法：

ifconfig [网络设备][down up -allmulti -arp -promisc][add<地址>][del<地址>][<硬件地址>][media<网络媒介类型>][mem_start<内存地址>][metric<数目>][mtu<字节>][netmask<子网掩码>][tunnel<地址>][-broadcast<地址>] [-pointopoint<地址>]

参　数：

add<地址> 设置网络设备 IPv6 的 IP 地址。

del<地址> 删除网络设备 IPv6 的 IP 地址。

down 关闭指定的网络设备。

<硬件地址> 设置网络设备的类型与硬件地址。

io_addr 设置网络设备的 I/O 地址。

irq 设置网络设备的 IRQ。

media<网络媒介类型> 设置网络设备的媒介类型。

mem_start<内存地址> 设置网络设备在主内存所占用的起始地址。

metric<数目> 指定在计算数据包的转送次数时，所要加上的数目。

mtu<字节> 设置网络设备的 MTU。

netmask<子网掩码> 设置网络设备的子网掩码。

tunnel<地址> 建立 IPv4 与 IPv6 之间的隧道通信地址。

up 启动指定的网络设备。

-broadcast<地址> 将要送往指定地址的数据包当成广播数据包来处理。

-pointopoint<地址> 与指定地址的网络设备建立直接连线，此模式具有保密功能。

-promisc 关闭或启动指定网络设备的 promiscuous 模式。

指定网络设备的 IP 地址。

[网络设备] 指定网络设备的名称。

ping 命令是一个检查网络联通状况的常用命令，它发送一个回送信号请求给网络主机。

语　法：

ping [-d] [-D] [-n] [-q] [-r] [-v] [\ -R] [-a addr_family] [-c Count] [-w timeout] [-f | -i \ Wait] [-l Preload] [-p Pattern] [-s PacketSize] [-S hostname/IP addr] \ [-L] [- I a.b.c.d.] [-o interface] [-T ttl] Host [PacketSize] \ [Count]

参　数：

-c Count 指定要被发送（或接收）的回送信号请求的数目，由 Count 变量指出。

-w timeout 这个选项仅和 -c 选项一起才能起作用。它使 ping 命令以最长的超时时间去等待应答（发送最后一个信息包后）。

-d 开始套接字级别的调试。

-D 这个选项引起 ICMP ECHO_REPLY 信息包向标准输出的十六进制转储。

-f 指定 flood-ping 选项。-f 标志“倾倒”或输出信息包，在它们回来时或每秒 100 次，选择较快一个。每一次发送 ECHO_REQUEST，都打印一个句号，而每接收到一个 ECHO_REPLY 信号，就打印一个退格。这就提供了一种对多少信息包被丢弃的信息的快速显示。仅仅 root 用户可以使用这个选项。

-I a.b.c.d 指定被 a.b.c.d 标明的接口将被用于向外的 IPv4 多点广播。-I 标志是大写的 i 。

-o interface 指出 interface 将被用于向外的 IPv6 多点广播。接口以 “en0”，“tr0”等的形式指定。

-i Wait 在每个信息包发送之间等待被 Wait 变量指定的时间（秒数）。缺省值是在每个信息包发送之间等待 1 秒。这个选项与 -f 标志不兼容。

-L 对多点广播 ping 命令禁用本地回送。

-l Preload 在进入正常行为模式(每秒 1 个)前尽快发送 Preload 变量指定数量的信息包。-l 标志是小写的 L。

-n 指定仅输出数字。不企图去查寻主机地址的符号名。

-p Pattern 指定用多达 16 个“填充”字节去填充你发送的信息包。这有利于诊断网络上依赖数据的问题。例如，-p ff 全部用 1 填充信息包。

-q 指定静默输出。除了在启动和结束时显示总结行外什么也不显示。

-r 忽略路由表直接送到连接的网络上的主机上。如果 主机 不在一个直接连接的网络上，ping 命令将产生一个错误消息。这个选项可以被用来通过一个不再有路由经过的接口去 ping 一个本地主机。

-R 指定记录路由选项。-R 标志包括 ECHO_REQUEST 信息包中的 RECORD_ROUTE 选项，并且显示返回信息包上的路由缓冲。

-a addr_family 映射 ICMP 信息包的目的地址到 IPv6 格式，如果 addr_family 等于“inet6”的话。

-s PacketSize 指定要发送数据的字节数。缺省值是 56，当和 8 字节的 ICMP 头数据合并时被转换成 64 字节的 ICMP 数据。

-S hostname/IP addr 将 IP 地址用作发出的 ping 信息包中的源地址。在具有不止一个 IP 地址的主机上，可以使用 -S 标志来强制源地址为除了软件包在其上发送的接口的 IP 地址外的任何地址。如果 IP 地址不是以下机器接口地址之一，则返回错误并且不进行任何发送。

-T ttl 指定多点广播信息包的生存时间为 ttl 秒。

-v 请求详细输出，其中列出了除回送信号响应外接收到的 ICMP 信息。

更多命令信息请各位自行使用 man 命令查询，例：man ip。

3 虚拟机

我们采用虚拟机（Virtual Machine）软件来模拟一个网络环境进行实验，这类软件的主要功能是利用软件来模拟出具有完整硬件系统功能的且运行在隔离环境中的完整计算机系统。这样我们可以在一台物理计算机即宿主机器（Host Machine）上模拟出一台或多台虚拟的计算机。这些虚拟机能够像真正的计算机那样进行工作，我们可以在其上安装全新的操作系统和应用软件。通过虚拟机软件中的虚连接设备将各个虚拟机连接起来，我们就可以搭建出实验所需的网络环境。

实验方法

本实验的主要目的是让学生了解在一个常见的 UNIX/Linux 系统中，熟悉系统最基本的网络工具集合（包括 ifconfig、route、wireshark 等）的使用，并能够熟练观察和初步分析协议 PDU 的内容，为进一步的实验打下基础。

部署，操作/设计步骤（虚拟机器的配置）

1 虚拟机及网络配置

虚拟机软件以 VMWare 为例，操作系统选用 Ubuntu 12.04。

1.1 首先按照提示安装/拷贝多个虚拟机

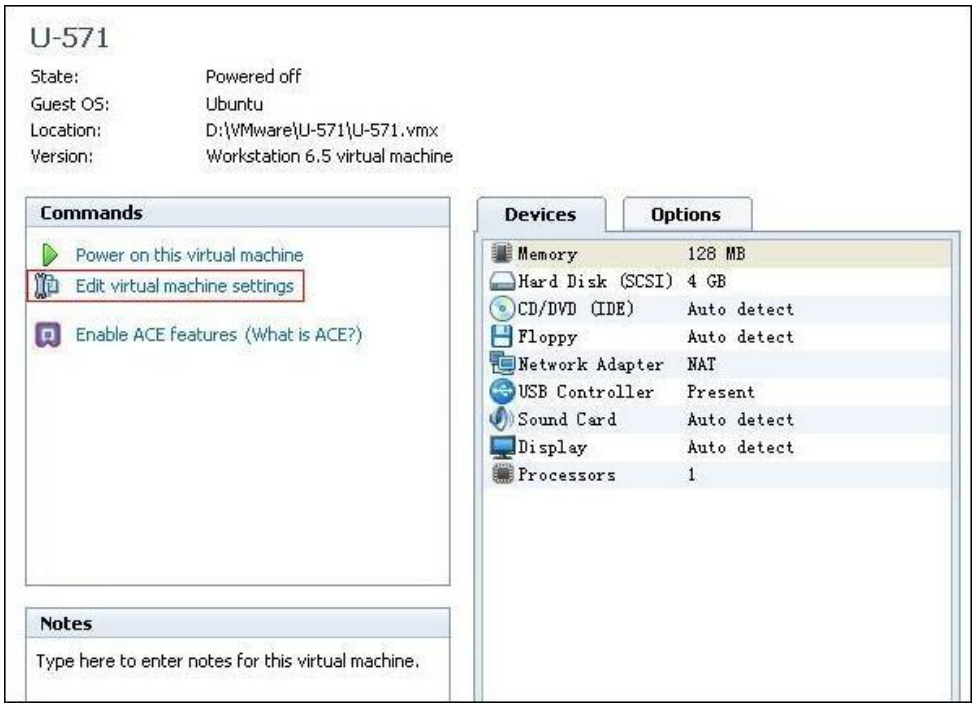
除需运行图形界面软件的虚拟机外，其它默认采用字符界面。在字符界面下可以选用普通用户或根用户(root)登录。用普通用户登录时，命令提示符为$，在执行需要 root 用户权限的命令时可通过在命令前加 sudo，或用 su 命令提升为 root 用户完成。用 root 用户登录时，命令提示符变为#。

从字符界面启动图形界面时用命令 startx，为使图形界面运行正常，请先确保虚拟机内存达到 256M。

1.2 连接已安装好的多个虚拟机

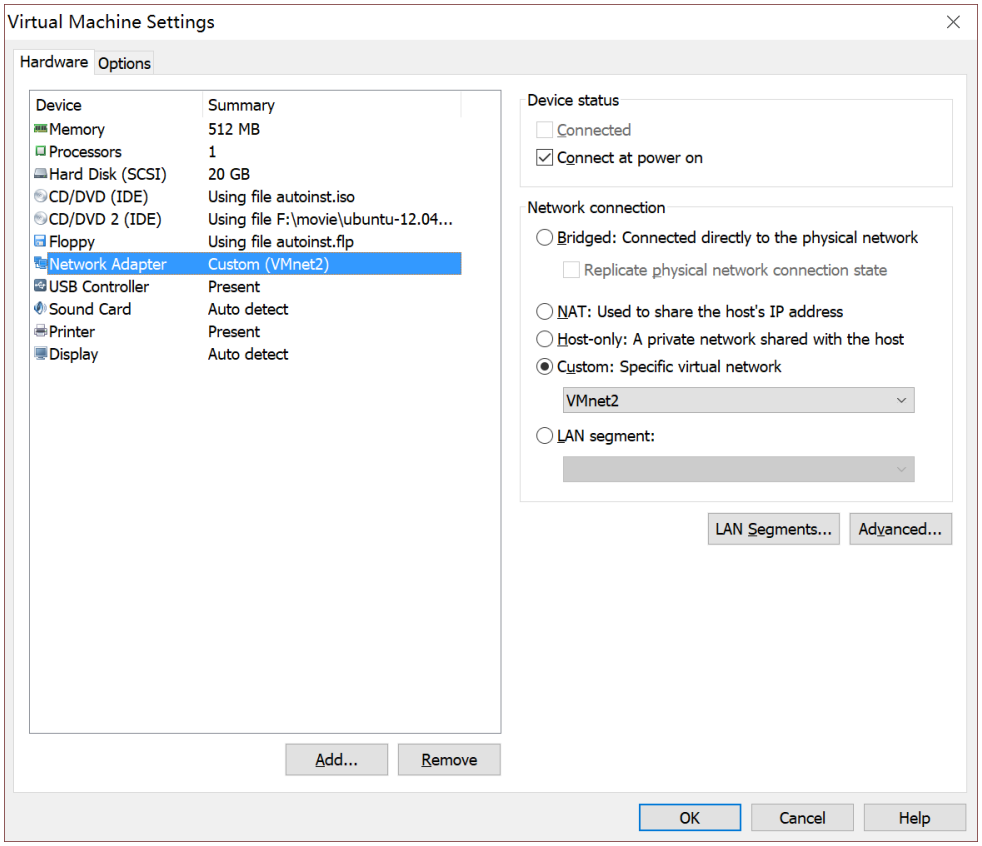
VMWare 提供了十个虚拟交换机 VMnet0—VMnet9。其中 VMnet0、VMnet1 和 VMnet8 为专用设备，分别以 default Bridged、Host-only 和 NAT 三种方式为虚拟机提供宿主机器原网络服务。另外七个虚拟交换机未被定义，可以用它们进行连接，配制虚拟网络。

如图为虚拟机启动前状态：



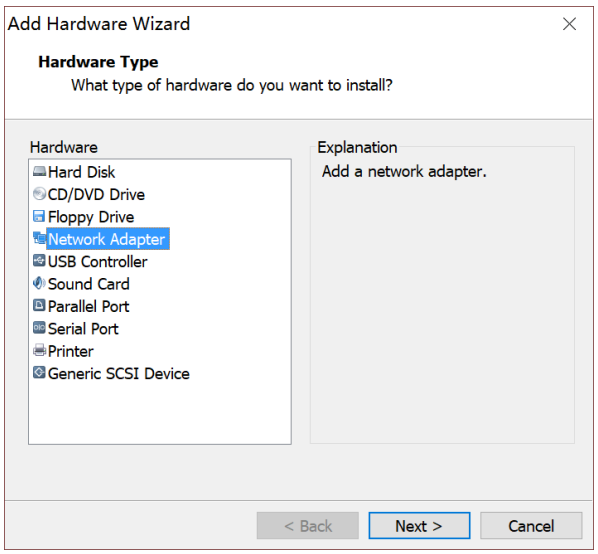
选择 *Commands* 条目下 *Edit virtual machine settings* 选项，即红框中部分。可对虚拟机的虚拟硬件设备进行调配。

设置界面如下：

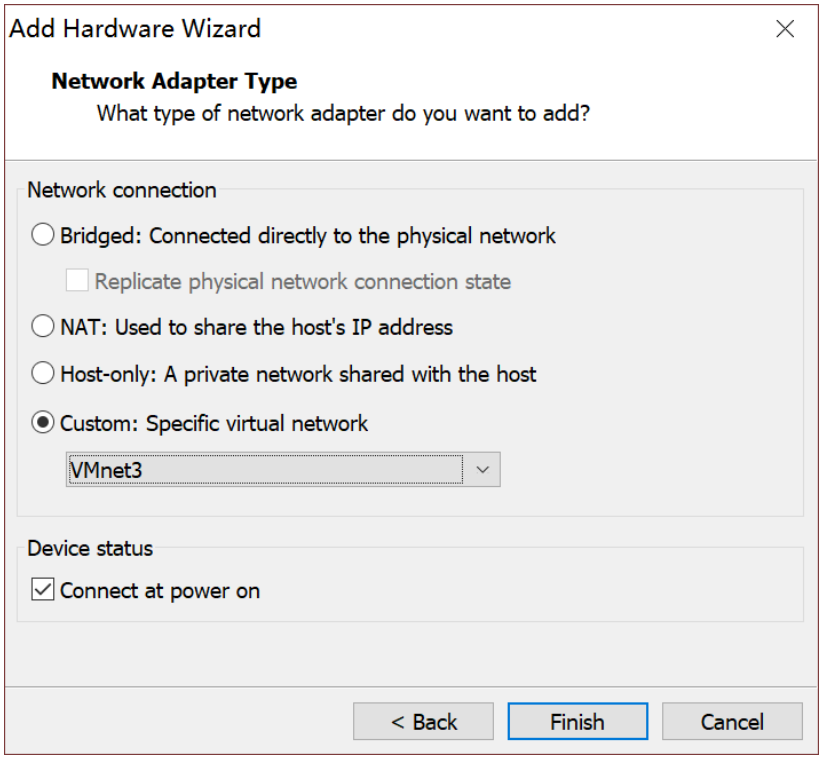


选择 *Network Adapter*,在右侧的选项框中，用 *Custom:Specific virtual network* 条目设置虚拟机与虚拟交换机的连接。如图虚拟机 U-571 与虚拟交换机 VMnet2 连接。

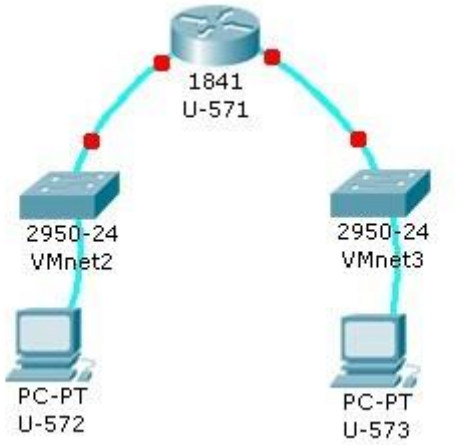
VMWare 没有提供虚拟路由，我们需要用虚拟机来模拟出一个路由器，这样用来模拟路由器的虚拟机至少需要两张网卡，同样通过 *Virtual Machine Settings*，可以为虚拟机添加多张网卡。选择上图左边框下方的 *Add* 按钮



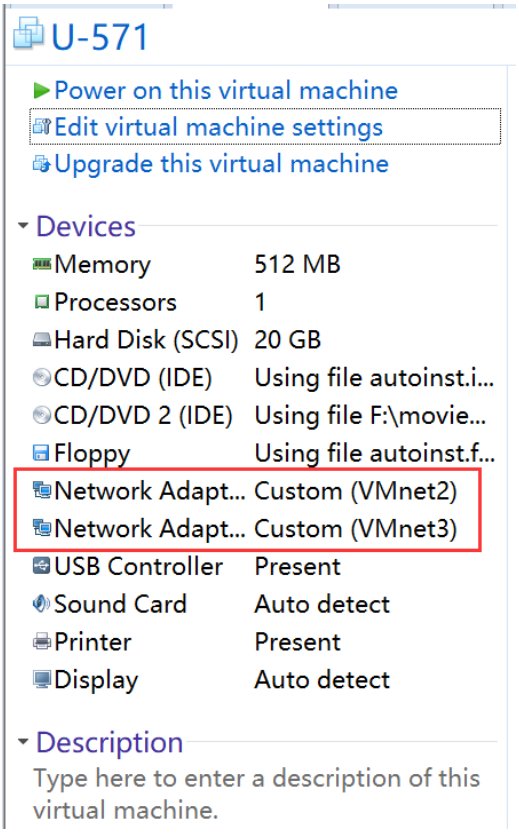
点选 *Network Adapter*

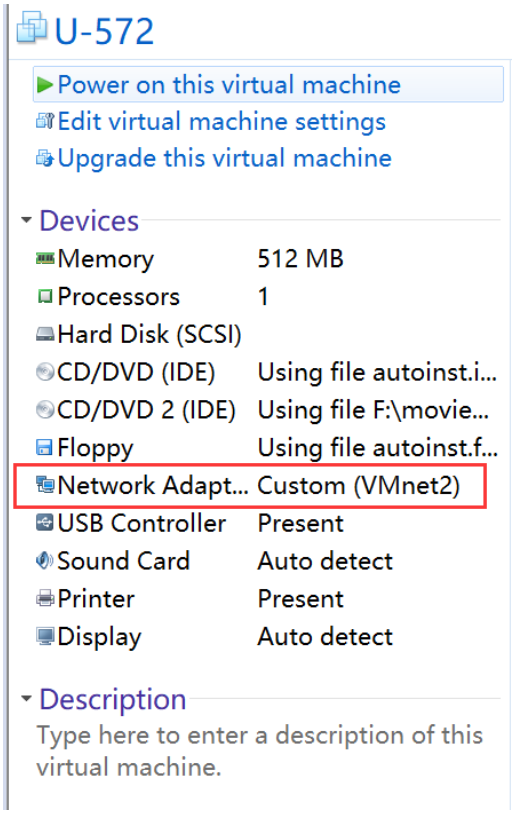


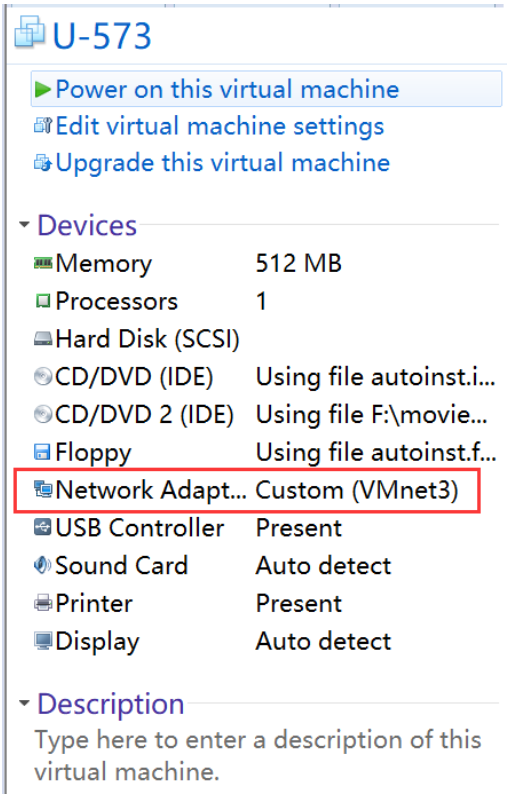
同样选择 *Custom:Specific virtual network*，这张虚拟网卡与虚拟交换机 VMnet3 连接。连接一个简单的网络如下图所示时



虚拟机的配置图如下：







2 设置 IP 与路由规则

IP 地址是计算机进行网络通讯的基础，每一台联网计算机都至少具有一个 IP 地址。在日常使用中，我们通常能自动获取 IP，这是由于 DHCP 协议的作用。在本次实验中我们需

要手动为配置好的虚拟网络分配 IP 地址。

首先使用 ifconfig 命令查看网络配置，以虚拟机 U-571 为例，键入命令

ifconfig -a | less

用"q"键退出。

此时，虚拟机还没有 ipv4 地址。

```
eth0      Link encap:以太网  硬件地址 00:0c:29:5c:fb:25
          inet6 地址: fe80::20c:29ff:fe5c:fb25/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  跃点数:1
          接收数据包:0 错误:0 丢弃:0 过载:0 帧数:0
          发送数据包:11 错误:0 丢弃:0 过载:0 载波:0
          碰撞:0 发送队列长度:1000
          接收字节:0 (0.0 B)  发送字节:2178 (2.1 KB)
          中断:19 基本地址:0x2000

eth1      Link encap:以太网  硬件地址 00:0c:29:5c:fb:2f
          inet6 地址: fe80::20c:29ff:fe5c:fb2f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  跃点数:1
          接收数据包:0 错误:0 丢弃:0 过载:0 帧数:0
          发送数据包:10 错误:0 丢弃:0 过载:0 载波:0
          碰撞:0 发送队列长度:1000
          接收字节:0 (0.0 B)  发送字节:1836 (1.8 KB)
          中断:16 基本地址:0x2080
```

然后再使用 ifconfig 命令分别为两个网络设备 eth0、eth1 设置 IP（设置 IP 地址时，子网号最好和虚拟交换机 VMnet*保持一致，便于调试），命令形式如下：

sudo ifconfig eth0 192.168.2.1 netmask 255.255.255.0

配置好后再用 ifconfig -a 查看

```
eth0      Link encap:以太网  硬件地址 00:0c:29:5c:fb:25
          inet 地址:192.168.2.1  广播:192.168.2.255  掩码:255.255.255.0
          inet6 地址: fe80::20c:29ff:fe5c:fb25/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  跃点数:1
          接收数据包:0 错误:0 丢弃:0 过载:0 帧数:0
          发送数据包:22 错误:0 丢弃:0 过载:0 载波:0
          碰撞:0 发送队列长度:1000
          接收字节:0 (0.0 B)  发送字节:4813 (4.8 KB)
          中断:19 基本地址:0x2000

eth1      Link encap:以太网  硬件地址 00:0c:29:5c:fb:2f
          inet 地址:192.168.3.1  广播:192.168.3.255  掩码:255.255.255.0
          inet6 地址: fe80::20c:29ff:fe5c:fb2f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  跃点数:1
          接收数据包:0 错误:0 丢弃:0 过载:0 帧数:0
          发送数据包:19 错误:0 丢弃:0 过载:0 载波:0
          碰撞:0 发送队列长度:1000
          接收字节:0 (0.0 B)  发送字节:3921 (3.9 KB)
          中断:16 基本地址:0x2080
```

对于端系统和大多数情况下的路由器还需要设置网关（路由器中也称为转发规则）（由于例子中的拓扑只有一个路由器，因此跨路由器的数据报均由该路由器转发，因此路由器 U-571 不需要设置转发规则）

以虚拟机 U-572 为例，设置 U-571 的 eth0 为其默认网关（一般情况下，一台主机只有一个默认网关，当终端只有一个网络适配器时，可以用设置默认网关这种操作方式），使用

route 命令，命令形式如下：

sudo route add default gw 192.168.2.1

用命令 route 查看结果

```
内核 IP 路由表
目标            网关            子网掩码        标志  跃点   引用  使用 接口
default         192.168.2.1     0.0.0.0         UG    0      0        0 eth0
```

IP 设置好后，就可以根据 IP 在路由器上设置路由规则。（刚才已经说过，因为拓扑中只有一个路由器，其实添加路由规则是不必要的，但是为了给大家展示命令的使用，下面给出了添加路由规则的命令）。

如为使虚拟机 U-572 和 U-573 之间能够进行通讯，在 U-571 上添加路由规则，命令形如：

sudo ip route add 192.168.2.0/24 via 192.168.2.1

sudo ip route add 192.168.3.0/24 via 192.168.3.1

其中 ip route add 192.168.2.0/24 via 192.168.2.1 命令添加的规则，告诉路由目的 IP 在 192.168.2.0/24(192.168.2.1~192.168.2.255)网段内的数据报经由 IP 地址为 192.168.2.1 的设备转发出去，即下一跳的 IP 为 192.168.2.1。而 192.168.2.0/24 是 Linux 中常用的掩码表示方式。24 表示掩码字长为 24 即掩码为 255.255.255.0，192.168.2 为网络号，1~254 为网络中的主机号。此外还有其他形式用于添加路由规则的命令。

当输入上述命令后，Linux 会提示 file exists 的错误信息，说明 Linux 在分配 IP 地址的时候，会默认帮我们生成一部分路由信息，默认生成的一般是正确的，因此可直接执行下一步操作。但是有两个以上的路由器该怎么设置 IP 信息呢？当有超过两个路由器时，添加路由规则的正确与否在于 via 后面的“下一跳”IP 地址，请大家自行理解“下一跳”的含义，探索如何正确设置路由器的路由规则。

最后我们要让虚拟路由允许转发，置虚拟机 U-571 的 ip_forward 标志为 1。这里我们需要把/proc/sys/net/ipv4/目录下的文件 ip_forward 值置为 1。使用命令 echo，形如：

echo 1 > /proc/sys/net/ipv4/ip_forward

注意，在执行命令前，请先输入：

sudo su

进入 root 用户，再对 ip_forward 的值进行修改。否则会报“permission deny”的错误。更改完成后，请输入 exit 返回到普通用户，因为 root 用户权限太高，对 Linux 所有文件拥有修改权限，如果不小心改动了不该更改的文件，会对 Linux 造成不可逆的损害。

最后，如果 U571、U572 和 U573 能够相互 ping 通，证明上述实验成功完成。

3 抓取协议数据单元

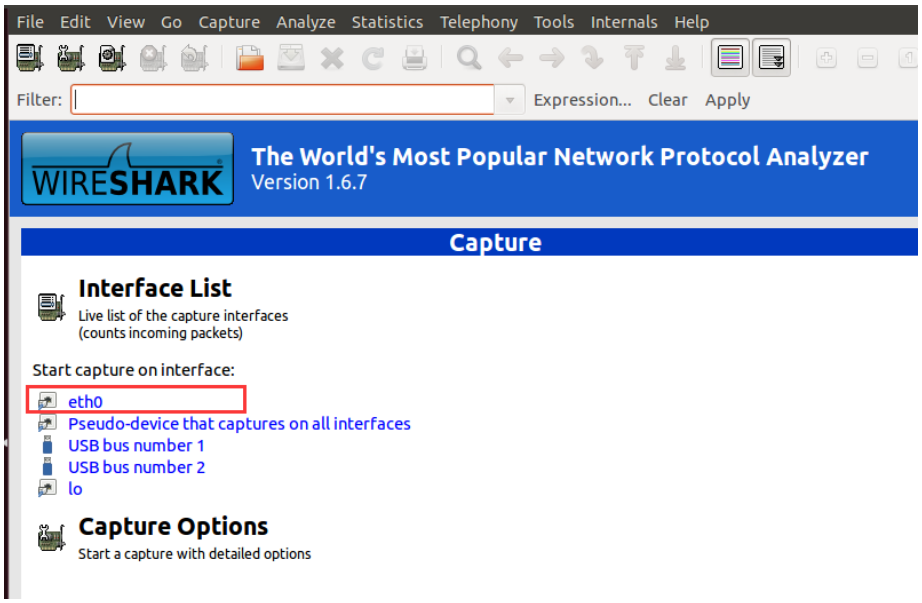
在网络配置完成的基础上，使用网络抓包分析工具 wireshark 抓取协议数据单元进行分析。例如在虚拟机 U-572 上使用 ping 命令测试与虚拟机 U-573（设其 ip 为 192.168.3.2）是否连通。使用命令

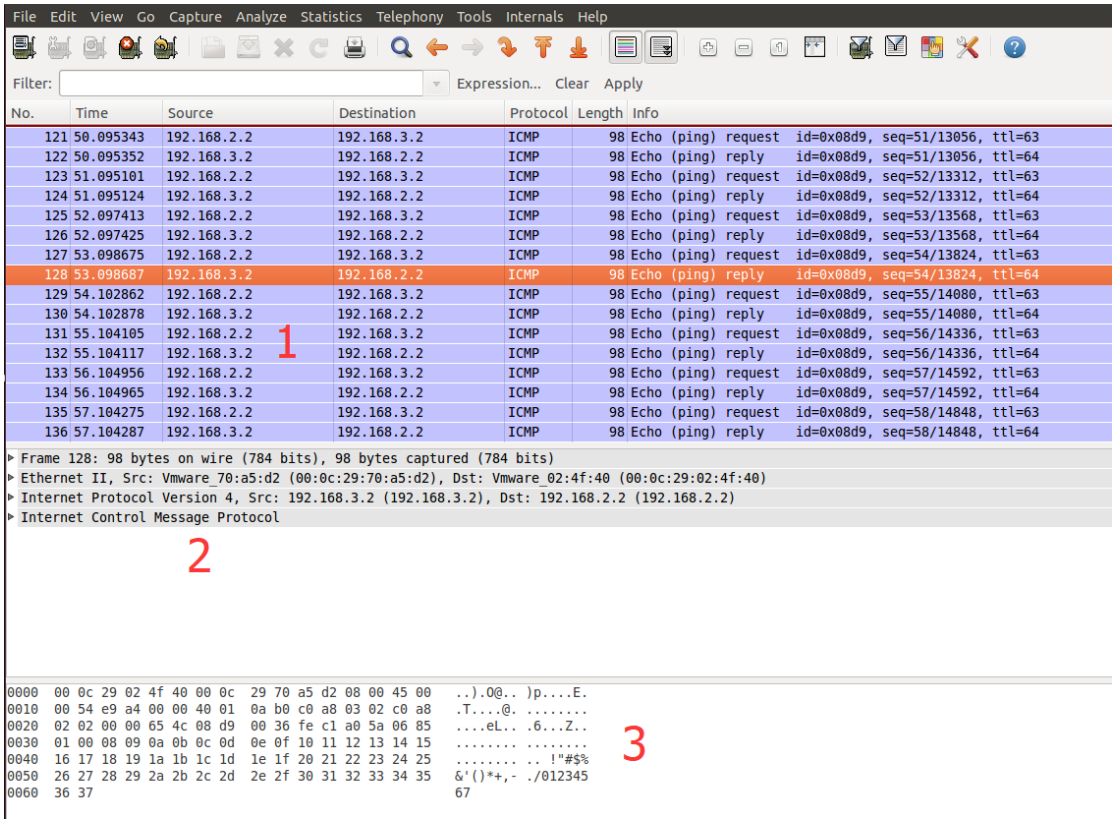
ping 192.168.3.2

在图形界面下使用 wireshark 可以抓取数据包，运行 wireshark 时需要 sudo 权限，否则无法进行数据报的抓取。

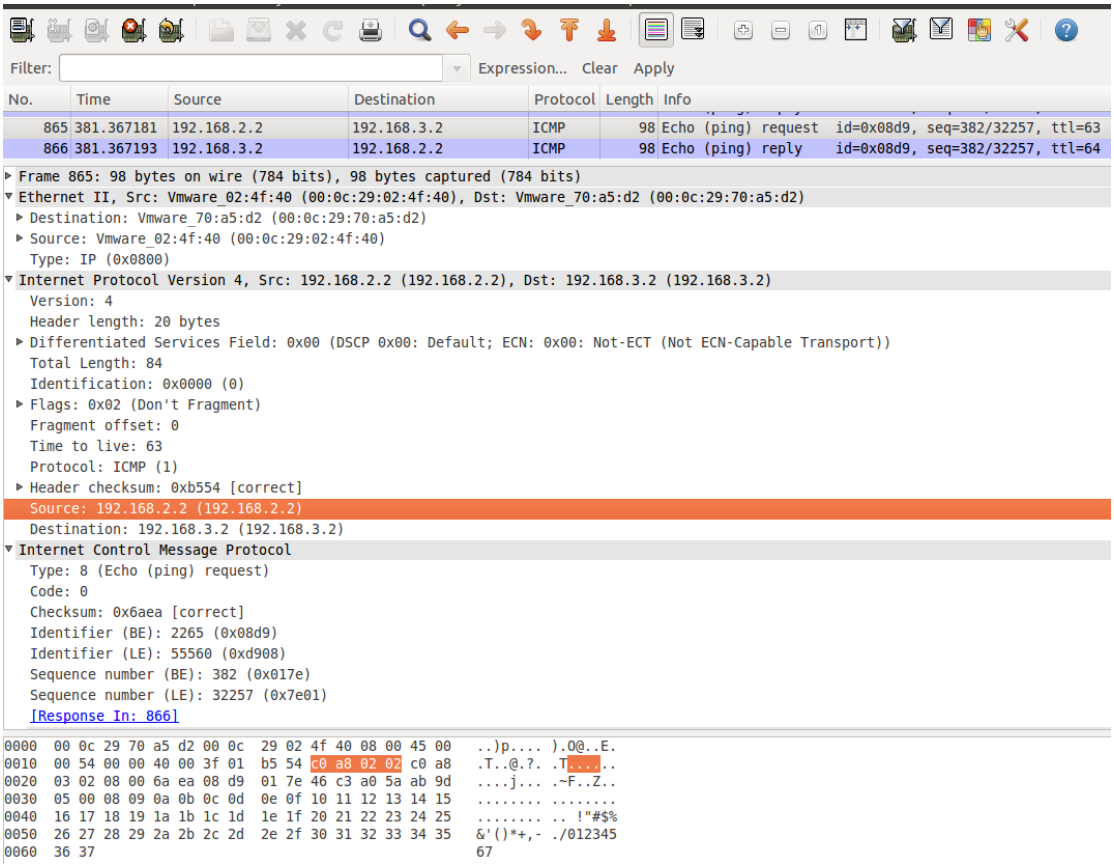
sudo wireshark

在下图中 Interface list 中选择要监听的设备，单击即开始监听，运行情况如下：



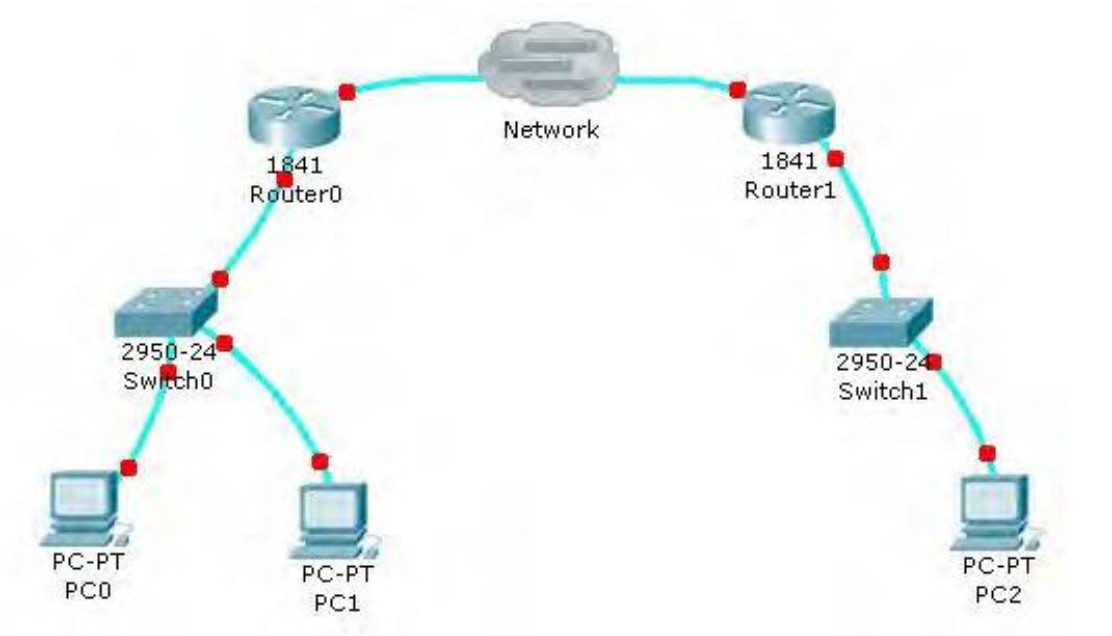


wireshark 能够深入地解析每个分组。图示中第一个区域是列表框，记录了数据报的基本信息（序号、时间戳、源地址、目的地址、协议、数据报长度、大致信息），第二个区域为协议框，wireshark 已经帮我们分析好了各个协议并予以显示，在此区域可以看到数据报的字段信息和每个字段的含义，第三个区域为原始框，上面给出了数据报的 16 进制数据信息和 ASCII 表示的信息。在列表框中选中一个分组，协议框和原始框中会显示该分组的详细信息。如图：



协议框中显示所选分组的各层协议：物理层帧、以太网帧、IP 协议，Internet 控制报文协议。原始框中则显示分组中包含的数据的每个字节。从中可以观察到原始数据，其中左边显示的是十六进制的数据，右边则是 ASCII 码。在协议框中选中一个条目，在原始框中会标记出对应的原始数据，反之在原始框中选中也一样。因此，我们可以利用 wireshark 轻松的分析协议的各个字段的含义及属性。

下图为实验任务的参考拓扑：



常见问题

问题 1：主机无法 Ping 通与其直接相连的虚拟路由器？
解答：a. 检查主机和虚拟路由器的网卡 Network Adapter 是否为同一类型的虚拟交换机 VMnet*（在虚拟机的设置中，网卡顺序和 ifconfig -a 看到的网卡顺序是相同的）；

b. 检查主机网关是否为虚拟路由器对应的网卡 IP 地址；

c. 确认 IP 设置无误，注意请先置 IP 再置网关（在设置 IP 地址时，请确保设置 IP 的网络适配器的正确性）。

问题 2：主机可以 ping 通网关，但是无法 ping 通主机所在局域网外的路由器或主机？
解答：a. 检查虚拟路由器是否允许转发，确保 ip_forward 为 1，即允许转发；

b. 检查虚拟路由器转发规则是否正确，确保转发端口（下一跳）和当前主机不在同一个局域网内。

c. 当有一段很长的链路需要进行联通(ping)测试时，可以采取分段分析的策略，通过分段分析可以准确定位

问题 3：为什么打开 572-573 或者 575-578 的时候，需要定位 571 或者 574 的虚拟机文件？
解答：在创建虚拟机的时候，为了节省计算机资源，572-573、575-578 均为链接复制，因此在打开时需要定位主文件的位置。（有兴趣的同学可以自己创建多个虚拟机）

问题 4：为什么我设置了 IP 地址，过一会儿之后 IP 地址自动消失了？
解答：实验室的图形界面虚拟机默认开启了 DHCP 模式，会自动发现并分配 IP 地址，但是局域网内并没有 DHCP 服务器，因此该服务启动时，会抵消掉原来设置好的 IP 地址。因此在使用图形界面时，可以先用下述命令：

sudo service network-manager stop

关闭网络服务，再手动设置 IP 地址。

顺便一提，如果你觉得每次开机都要设置 IP 地址很麻烦，可以在/etc/network/interfaces 文件中写入你想设置的 IP 地址。（/etc/network/interfaces 的格式请自行查阅资料）

问题 5：在添加路由规则的时候出现了“no such process” ，该如何处理？
解答：假设添加路由规则的命令为：ip route add 192.168.a.0/24 via 192.168.c,d
1. 原理上来说，路由规则如果能正确生效，192.168.c.d 必须是该路由器能够“通信”的 IP 地址，在我们实验中，可以理解为 192.168.c.d 和路由器的出端口 IP 地址在同一个子网下，例如路由器的出端口是 eth1，IP 地址为 192.168.3.1，那 via 后面的 IP 地址 192.168.c.d，必须是 192.168.3.x 和 eth1 的 IP 地址在同一子网内。如果不在同一子网内会报“no such process”的错误，输入正确的路由规则即可。
2. 还是上述路由器，如果添加路由规则时，在 via 后面先添加了自己出端口的 IP 地址，ip route add 192.168.a.0/24 via 192.168.3.1，那么把这条路由规则删除，再添加相关的路由规则时，比如 ip route add 192.168.a.0/24 via 192.168.3.2，也会报“no such process”的错误。如果是这种情况，先利用命令 ifconfig eth1 down 使该网口失效，然后重新设置 IP 地址和路由规则即可。

问题 6：为什么 sudo echo 1 > /proc/sys/net/ipv4/ip_forward 会报错？

解答：sudo 的作用是将命令以 root 权限执行，仅作用于后面跟的那一条命令。
> file 表示将标准输出重定向到 file 文件内，这一步是由 shell 来执行的，因此受制于当前 shell 的权限。
shell 将 sudo echo 1 的输出写入到/proc/sys/net/ipv4/ip_forward 时，由于没有对应的写入权限（需要 root），因此会报错。

实验二 RAW SOCKET 编程与以太网帧分析基础

实验任务

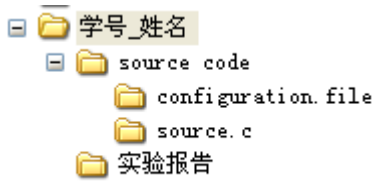
1. 编写自己的抓包程序。在本机分别运行 ping 程序和用浏览器浏览网页，使用该抓包程序记录相应的数据包，截图并做简要分析。
2. 编写自己的 ping 程序。利用 Raw Socket 封装和发送以太网帧的功能，实现 ICMP 包的发送和接收，并输出结果。分别运行你的 ping 和系统的 ping 程序，截图并作简单比较。

提交材料：实验报告及程序源代码。

实验报告

由于本次实验着重程序的设计，试验报告中将不要求统一的配置性细节，大家可以自行设置实验环境。

实验者需提交源代码以及实验报告。提交文件布局如下：



其中 source.c 为 C 的源文件，configuration.file 为相关配置文件。建议同学可以在 source.c 中的重要代码或者函数入口处添加注释，不做定性要求。

实验报告最好提交 word 文档或 pdf 文档，应包含如下表所示的内容，同学可以根据内容进行修改。说明是为了更详细的解释各个项目，提交的实验报告中无需包含。

实验目的	

数据结构说明	说明：此处需要解释各自的源代码中的数据结构，以及其用途。
	示例：
	struct xiaomaolv{
	int id;
	floag weight;
	int age;
	};
	该结构是一个描述 xiaomaolv 的结构，对应于现实生活农场中养的毛驴，其中 id 是其在农场中的编号，weight,age 分别代表了该毛驴的质量和年龄。

配置文件说明（非必须）	说明：此处需要给出配置文件的格式以及读取方式，如程序中没有用到配置文件，则该项可省略
程序设计的思路以及运行流程	说明：此处需要给出程序的运行流程或者思路。请给出如下两种格式之一： 1. 流程图 2. 流程说明 示例： 1. 程序开始 2. 读取配置 3. 检测数据 4. 处理数据 4.1 正确 goto 3 4.2 错误 goto 5 5. 程序结束
运行结果截图	说明：请给出你的运行结果截图
相关参考资料	说明：请给出你完成该实验的参考书目或者网页
对比样例程序	说明：请给出你参考样例程序的部分，假如没有参考点，填无
代码个人创新以及思考	说明：请给出你认为你的源代码中的亮点，比如，针对某个细节的处理或者算法的优化
该程序的应用场景创新（非必须）	说明：请思考一下，该类程序除了在背景中的应用之外，是否还有其他可能的应用场景。

背景知识

1 基础背景

在计算机网络中，socket 是一个很重要的概念。因为，在网络通讯之中，我们可以把其理解为两个进程的通信，通信的话就要描述通信链路两端的信息。而套接字就是使用操作系统中的文件描述符和系统进程进行通信的一种手段。Socket 有时也被看作是一个网络编程接口集，应用程序，或者 TCPIP 协议栈。

目前，有许多类型的 socket 可用，其中包括数据流套接字，数据报套接字，原始套接字等。其中在一般的应用程序之中，前二者用的比较多，但是，在一些安全程序，网络管理程序之中，对于数据链路层整体包的捕获要求，它们并不能很好的满足。那么，这里就要用 raw socket, 也就是原始套接字。

原始套接字可以使得应用程序直接收发在网络上传输的包，那么我们就可以直接对原始数据包的内容进行修改和检测，这对特定环境下的应用非常必要。比如，在以太网环境下，网络是共享的，所有的包传输都是通过同过如下方式进行传播的：发送主机要向另外一个主机发包，那么当这个包离开发送主机在网络上传输的时候，其他所有的主机都可以收到这个包，但是，在操作系统的内核协议栈上，对目标地址与自己无关的包都被丢弃了。那么对于网络管理员来说，可以对整个网络的数据进行监控和控制。

嗅探，就是网络管理员进行网络监控的一种途径。所谓嗅探，就是充分利用了以太网的特点，通过将网卡设置成混杂模式，将网络上传输的包进行捕获，并显示出来。通过这种方法，可以检测出网络的流量状况，ARP 欺骗等一系列问题。

2 Raw socket 编程基础

Raw socket 的编程源自于 socket 编程，只是参数和实际收发包的整体内容有所区别。同 socket 一样，要收发包首先应该先建立一个 socket。socket 的函数原型如下：

```
int socket(int domain,int type,int protocol);
```

其中 domain 标识一个通信域，一般来说，通信域决定了真正用于通信的协议族。

就目前来说，支持的域包括 UNIX, INET, INET6, IPX 等连接服务。其中支持 TCP/IP 协议的套接口类型是 INET。在 Linux 的 INET 套接口支持 SOCK_STREAM, SOCK_DGRAM, SOCK_RAW 等套接口类型。下面是一个创建普通套接字类型的例子。

```
int sockfd;
sockfd = socket(AF_INET, SOCK_RAW, protocol);
```

这是一个很普通的创建 raw socket 的操作，但是他所真正控制的仅仅是一个 IP 包，假如我们要对数据链路层的数据进行操作，我们需要按照如下方式进行创建

```
sockfd  = socket(PF_PACKET,SOCK_RAW,htons(ETH_P_IP);
```

创建完 socket, 可以自定义 socket 的行为, 对应的函数原型是

```
setsockopt(int sockfd,int level,int optname,const void * optval,socklen_t optlen);
```

其中参数的含义分别是：sockfd 指明要设置的套接字描述符，level 指明定义的层次，一般可用的选项有 SOL_SOCKET, IPPROTO_IP, IPPROTO_IPV6, IPPROTO_TCP。例如在 socket API 的层次来定义选项，可以将 level 设置成 SOL_SOCKET。optname 指明要设置的选项 ID，而 optval 则是存放选项值的地址，optlen 指明选项值的长度。假设我们要在发送 UDP 包的时候使其进行广播，有

```
int bBroadcast=1;
setsockopt(s,SOL_SOCKET,SO_BROADCAST,(const char*)&bBroadcast,sizeof(int));
```

创建完一个 socket 之后, 这个 socketfd 对应的 socket 就可以收包了

recvfrom(int sockfd,void* buf,size_t len,int flags,struct sockaddr * src_addr,socklen_t * addrlen);

其中 buf 是收到的包所存放的位置, 这里假如我们创建 socket 的给出的参数是接收数据链路层上的包, 那么我们可以获得的数据包就是以太帧。

在 Linux 中，调用 man 可以查看相关命令的调用细节，如 man socket. 此处应当注意，man 手册共有 8 个 sections，每个对应于不同的区域。一般而言，对于程序设计者来说，会用 man 2，这里用的环境是 ubuntu 9.04，man 的默认 section 为 7。

```
SOCKET(7)                    Linux Programmer's Manual                    SOCKET(7)

NAME
       socket - Linux socket interface

SYNOPSIS
       #include <sys/socket.h>

       sockfd = socket(int socket_family, int socket_type, int protocol);

DESCRIPTION
       This  manual  page describes the Linux networking socket layer user interface.  The BSD
       compatible sockets are the uniform interface between the user process and  the  network
       protocol stacks in the kernel.  The protocol modules are grouped into protocol families
       like AF_INET, AF_IPX, AF_PACKET and socket types like SOCK_STREAM or  SOCK_DGRAM.   See
       socket(2) for more information on families and types.

   Socket Layer Functions
       These functions are used by the user process to send or receive packets and to do other
       socket operations.  For more information see their respective manual pages.

       socket(2) creates a socket, connect(2) connects a socket to a  remote  socket  address,
       the  bind(2)  function  binds  a  socket to a local socket address, listen(2) tells the
       socket that new connections shall be accepted, and accept(2)  is  used  to  get  a  new
```

图 1

3 以太帧解析

所谓以太帧，就是在以太网上传输数据的时候用的一个单位。在以太网上跑的基本是 MAC 包头的包，此时就有如下的 MAC 包，

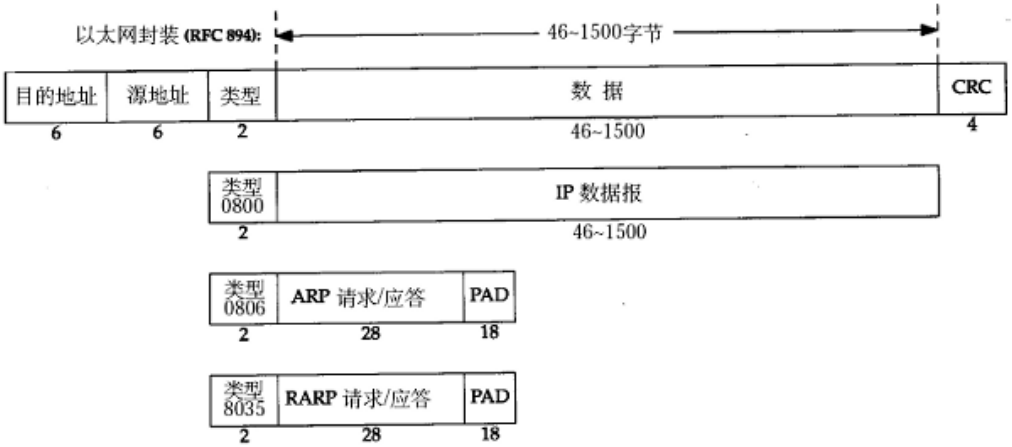


图 2

根据 MAC 头部的类型信息，对应的数据部分有多种可能。叫常用的有 0800, 0806, 8035。分别对应于 IP 数据包，ARP 请求/应答，RARP 请求应答。

其中 IP 数据包的格式如下：

图 3

下面是一个用 Wireshark 抓取的 ARP 请求包

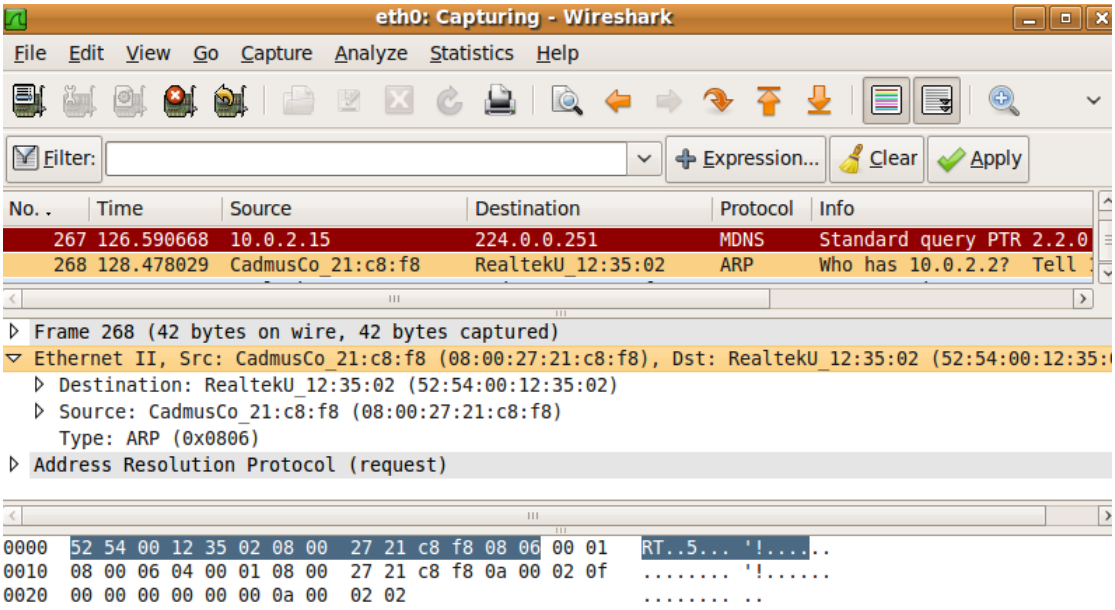


图 4

在这个图中，我们可以看到一个以太帧的完整格式，包括整个以太帧的 16 进制表示。

一些相关的资料链接

[1] http://en.wikipedia.org/wiki/Raw_socket

[2] http://bbs3.chinaunix.net/viewthread.php?tid=876233

实验目标

本实验主要目的是让学生熟悉 Linux 环境下基本的 raw socket 编程，对以太网帧进行初步分析，可通过程序完成对不同层次 PDU 的字段分析，修改和重新发送。

部署，操作/设计步骤 （虚拟机器的配置和拓扑）

我们要做的事情就是运行一个我们用 raw socket 实现的程序，该程序用来捕获主机之间互相发送的包，并可以修改相关字段并重新发送。

首先，我们应当明白，用 raw socket 捕获的数据包内容是存放在一个缓冲区内，一般情况下，是一个指针指向的内存区域。内存区域的不同字段所代表的含义是网络协议事先规定好的。修改相关字段也仅仅是修改对应内存上的数据而已。下面，给出实验的步骤。

1 虚拟机配置

我们需要的网络拓扑如下：

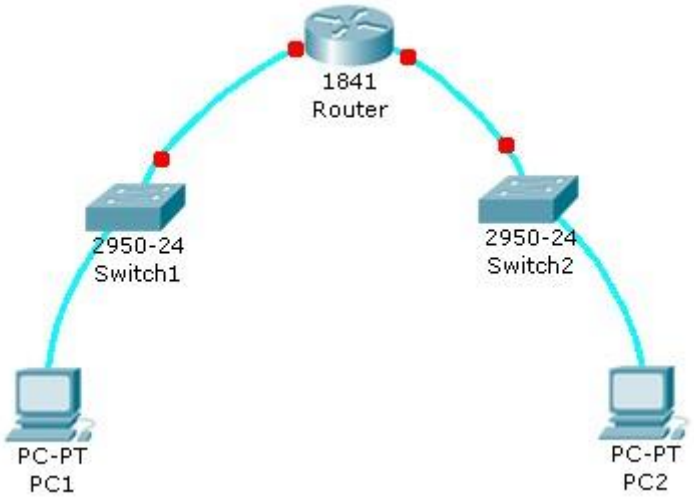


图 5

在 VMWare 设置如下，设置两台主机和一台路由，配置其为 PC1，PC2，Router。

在这里，我们用一台 Linux 主机来模拟 Router。

分别打开 shell, 输入如下命令

PC 1# **ifconfig eth0 192.168.0.2 netmask 255.255.255.0**

PC 1# **route add default gw 192.168.0.1**

PC 1# **sudo /etc/init.d/networking restart**
PC 2# **ifconfig eth0 192.168.1.2 netmask 255.255.255.0**
PC 2# **route add default gw 192.168.1.1**
PC 2# **sudo /etc/init.d/networking restart**
配置 Router
Router# **ifconfig eth0 192.168.0.1 netmask 255.255.255.0**
Router# **ifconfig eth1 192.168.1.1 netmask 255.255.255.0**
并开启 Router 的路由转发功能
Router# **vim /etc/sysctl.conf**
修改 net.ipv4.ip_forward = 1。
Router# **sudo /etc/init.d/networking restart**
测试
PC 1# **ping 192.168.1.2**
如果能 Ping 通说明连接已经建立，下面对抓包程序进行说明。

2 代码实现与检测

下面给出一个简单的抓包代码

```
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <linux/if_ether.h>
#include <linux/in.h>
#define BUFFER_MAX 2048
int main(int argc,char* argv[]){
   int sock_fd;
   int proto;
   int n_read;
   char buffer[BUFFER_MAX];
   char *eth_head;
   char *ip_head;
   char *tcp_head;
   char *udp_head;
   char *icmp_head;
   unsigned char *p;
   if((sock_fd=socket(PF_PACKET,SOCK_RAW,htons(ETH_P_IP)))<0){
      printf("error create raw socket\n");
      return -1;
   }
   while(1){
      n_read = recvfrom(sock_fd,buffer,2048,0,NULL,NULL);
      if(n_read < 42)
      {
         printf("error when recv msg \n");
```

```
            return -1;
        }
        eth_head = buffer;
        p = eth_head;
        printf("MAC address: %.2x:%02x:%02x:%02x:%02x:%02x
            ==> %.2x:%02x:%02x:%02x:%02x:%02x\n",
            p[6],p[7],p[8],p[9],p[10],p[11],
            p[0],p[1],p[2],p[3],p[4],p[5]);
        ip_head = eth_head+14;
        p = ip_head+12;
        printf("IP:%d.%d.%d.%d==> %d.%d.%d.%d\n",
            p[0],p[1],p[2],p[3],p[4],p[5],p[6],p[7]);
        proto = (ip_head + 9)[0];
        p = ip_head +12;
        printf("Protocol:");
        switch(proto){
        case IPPROTO_ICMP:printf("icmp\n");break;
        case IPPROTO_IGMP:printf("igmp\n");break;
        case IPPROTO_IPIP:printf("ipip\n");break;
        case IPPROTO_TCP:printf("tcp\n");break;
        case IPPROTO_UDP:printf("udp\n");break;
        default:printf("Pls query yourself\n");
        }
    }
    return -1;
}
```

程序的思路很简单，就是建立一个简单的链路层 socket, 然后不断的收包，显示包的部分内容，并根据包头类型域的值来显示包的类型。

在任何一台机器上编译该程序

PC 1# **gcc raw_socket.c –o raw_socket**

编译成功之后运行(运行时需要 root 用户下)

PC 1# **su yourpassword**

PC 1# **./raw_socket**

会收到相应的包并有结果输出

在笔者的机器上，输出如下：

PC 1 Ping PC 2 的情况 （框中内容为完整起见列出，实际不显示）

MAC address: PC 1's mac address ===> Router's mac address
IP:192.168.0.2 ==> 192.168.1.2
Protocol:icmp
MAC address: Router's mac address ===> PC 1's mac address
IP: 192.168.1.2 ==> 192.168.0.2
Protocol:icmp

实验者需修改程序，使其对更多的网络协议进行支持，比如 ARP 协议。并最好可以在终端下模仿 Wireshark, TCPDump, OmniPeek 等主流抓包工具的输出方式。同时应该尝试对数据段进行更改，比如可以自己调用 Raw Socket API 来发送一个 ICMP 请求包，而不是通过调用 Ping 命令来发送，等待和分析目的主机的回应。关于修改数据段的操作，可以尝试使用 memset 函数。

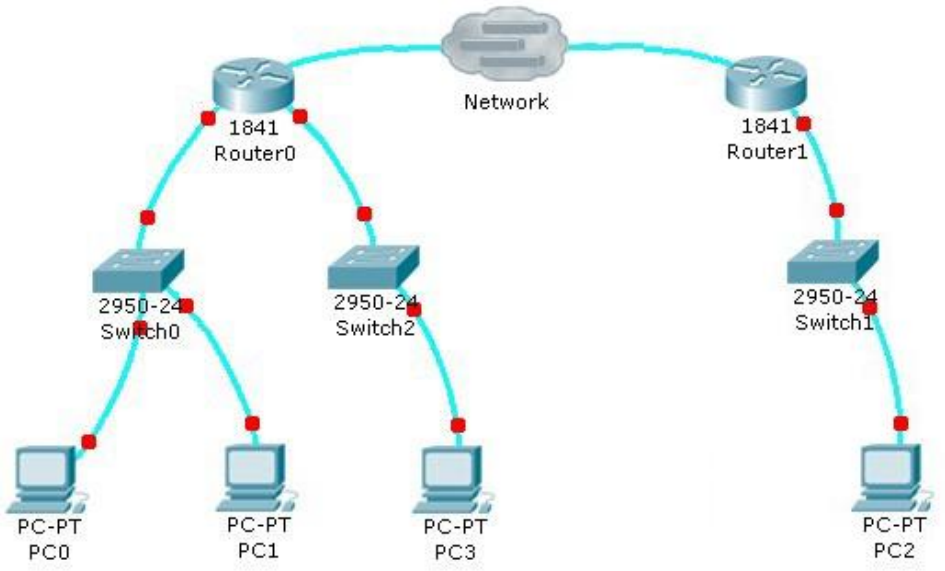
补充内容

sock_raw 编程可以接收到本机网卡上的数据帧或包，对于监听网络流量和分析是很有作用的。一共可以有 3 种方式创建：

- socket(AF_INET, SOCK_RAW, IPPROTO_TCP | IPPROTO_UDP | IPPROTO_ICMP)，发送和接收 IP 数据包；
- socket(PF_PACKET, SOCK_RAW, htons(ETH_P_IP |ETH_P_ARP |ETH_P_ALL))，发送和接收以太网数据帧
- socket(AF_INET, SOCK_PACKET, htons(ETH_P_IP|ETH_P_ARP|ETH_P_ALL))，老的用法，不推荐。

实验三 子网划分和 NAT 配置

实验任务



1. 按照如图所示的网络拓扑，配置网络，形成三个子网，子网 1（PC0，PC1），2（PC3），3（PC2），在 router0 上设置路由规则，使得子网 1 和子网 2 变成内网，子网 3 和 router0 的一个端口处于公网。
   设置完成之后，让 PC0，PC2，PC3 两两互 ping，并把截图记录下来。

2. 设置 iptables 规则，内网设备可以通过 NAT 跟公网设备通信。
   设置完成之后，让 PC0，PC2，PC3 两两互 ping，并把截图记录下来。
3. 比较 1 和 2 的结果，并解释 NAT 所起到的作用。

实验报告

按要求完成实验，并写出实验报告。实验报告格式如下，同学可以根据内容进行修改。说明是为了更详细的解释各个项目，提交的报告中无需包含。

实验目的	
网络拓扑配置	说明：填写附表 1，并绘图说明
路由规则配置	说明：写明输入的命令
NAT 设置命令	说明：写出所使用的命令
数据包截图	说明：用 wireshark 抓包截图
协议报文分析	说明：对抓取的数据包进行字段分析

附表 1：

节点名	虚拟设备名	ip	netmask
Router0		eth0:	
		eth1:	
		eth2:	
Router1		eth0:	
		eth1:	
PC0			
PC1			
PC2			
PC3			

背景知识

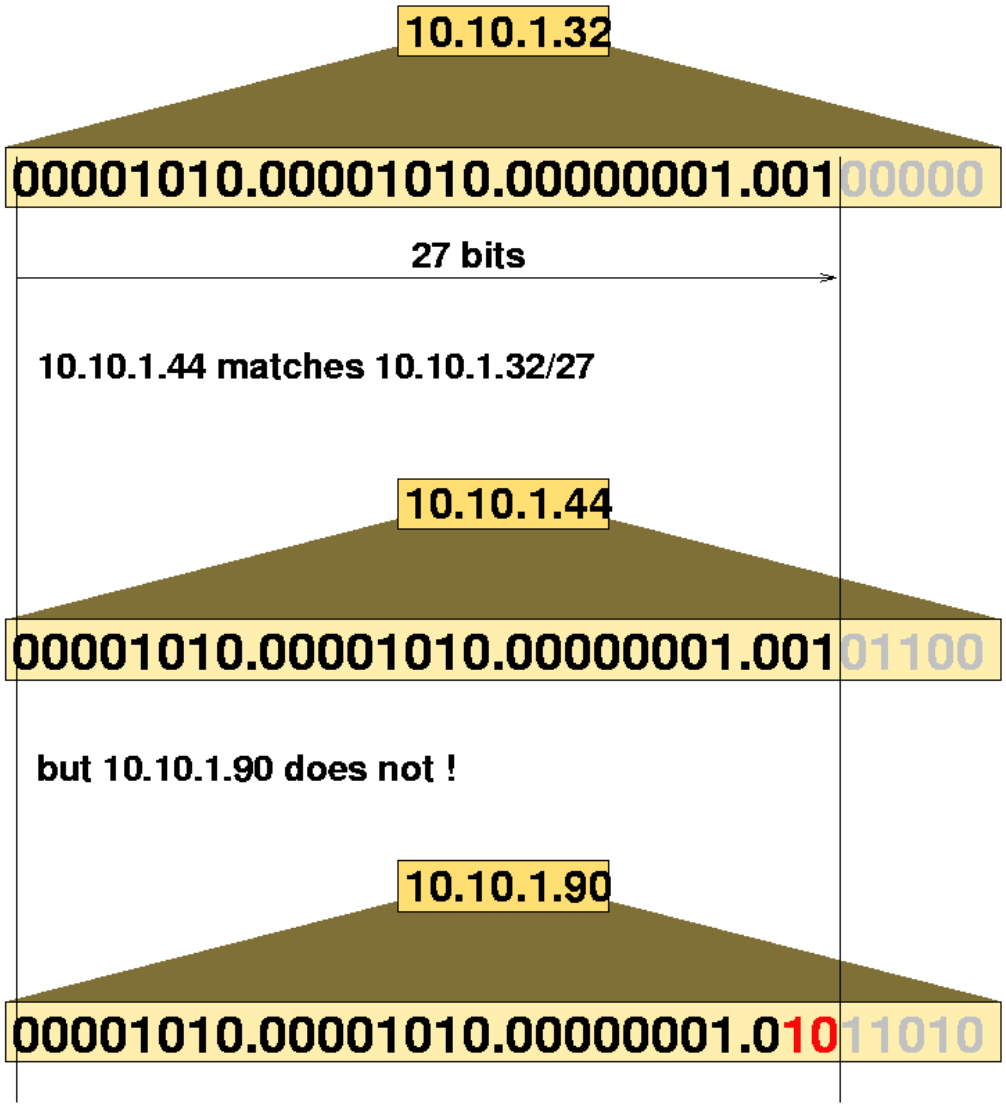
1 子网划分

以太网交换机在数据链路层上基于端口进行了数据使得冲突域被缩小到交换机的每一个端口，有效地提高了网络系统的利用率。但随着网络规模的增大，网络内主机数量急剧增加。当这些主机都同属一个局域网即同一广播域时，网络中任一主机发送的广播报文将被转发给广播域中的全部主机，造成网络利用率大幅下降。为避免这种情况，我们将大的广播域隔离成多个较小的广播域，即进行子网划分。

子网的划分，实际上就是设计子网掩码的过程。子网掩码主要是用来区分 IP 地址中的网络 ID 和主机 ID，它用来屏蔽 IP 地址的一部分，从 IP 地址中分离出网络 ID 和主机 ID。最初 IP 地址空间被分为 A 类，B 类，C 类三个分类网络，IP 地址的分配把 IP 地址的 32 位按每 8 位为一段分开。这使得前缀必须为 8，16 或者 24 位。因此，可分配的最小的地址块有 256（24 位前缀，8 位主机地址，28=256）个地址，而这对大多数企业来说太少了。大一点的地址块包含 65536（16 位前缀，16 位主机，216=65536）个地址，而这对大公司来说都太多了。这导致不能充分使用 IP 地址和在路由上的不便，因为大量的需要单独路由的小型

网络（C 类网络）因在地域上分得很开而很难进行聚合路由，于是给路由设备增加了很多负担。为了解决这个问题，而引入了无类别域间路由(CIDR)也即可变长子网掩码(VLSM)。

例如 IP 地址段：192.168.0.1-192.168.0.254，其中 192.168.0 这个属于网络号码，而 1～254 表示这个网段中最大能容纳 254 台主机。192.168.0.1-192.168.0.254 默认使用的子网掩码为 255.255.255.0，其中的 0 在 2 进制中表示为 8 个 0。因此有 8 个位置没有被网络号码给占用，2 的 8 次方就是表示有 256 个地址，去掉一个头（网络地址）和一个尾（主机地址），表示有 254 个主机地址，因此我们想要对这 254 来划分的话，就是占用最后 8 个 0 中的某几位。假如占用第一个 0，那么 2 进制表示的子网掩码为 11111111.11111111.11111111.10000000。转换为 10 进制就为 255.255.255.128，些时主机数即为 2 的 7 次方(不再是原来的 2 的 8 次方了)，2 的 7 次方=128，因此假如子网掩码为 255.255.255.128 的话，这个地址段可以被区分为 2 个网络，每个网络中最多有 128 台主机。 192.168.0.1-192.168.0.127 为一个，192.168.0.128-192.168.0.255 为第二个。关于 CDIR 块的匹配如图所示：



2 NAT

NAT（Network Address Translation）网络地址转换。NAT 的出现是为了解决 IP 日益短缺的问题，使用 NAT 技术可以在多重 Internet 子网中使用相同的 IP,从而解决了 IP 地址不足的问题，而且还能够有效地避免来自网络外部的攻击，隐藏并保护网络内部的计算机。NAT 的运作机制是自动修改 IP 报文的源 IP 地址和目的 IP 地址，IP 地址校验则在 NAT 处理过程中自动完成。NAT 的实现方式有三种，即静态转换 Static Nat、动态转换 Dynamic Nat 和 端口多路复用 OverLoad。

静态转换是指将内部网络的私有 IP 地址转换为公有 IP 地址，IP 地址对是一对一的，是一成不变的，某个私有 IP 地址只转换为某个公有 IP 地址。借助于静态转换，可以实现外部网络对内部网络中某些特定设备(如服务器)的访问。

动态转换是指将内部网络的私有 IP 地址转换为公用 IP 地址时，IP 地址对是不确定的，而是随机的，所有被授权访问上 Internet 的私有 IP 地址可随机转换为任何指定的合法 IP 地址。也就是说，只要指定哪些内部地址可以进行转换，以及用哪些合法地址作为外部地址时，就可以进行动态转换。动态转换可以使用多个合法外部地址集。当 ISP 提供的合法 IP 地址略少于网络内部的计算机数量时。可以采用动态转换的方式。

端口多路复用(Port address Translation,PAT)是指改变外出数据包的源端口并进行端口转换，即端口地址转换.采用端口多路复用方式。内部网络的所有主机均可共享一个合法外部 IP 地址实现对 Internet 的访问，从而可以最大限度地节约 IP 地址资源。同时，又可隐藏网络内部的所有主机，有效避免来自 Internet 的攻击。因此，目前网络中应用最多的就是端口多路复用方式。

3 iptables

iptables 是建立在 netfilter 架构基础上的一个包过滤管理工具，最主要的作用是用来做防火墙或透明代理。iptables 从 ipchains 发展而来，它的功能更为强大。iptables 提供以下三种功能：包过滤、NAT（网络地址转换）和通用的 pre-route packet mangling。包过滤：用来过滤包，但是不修改包的内容。iptables 在包过滤方面相对于 ipchians 的主要优点是速度更快，使用更方便。NAT：NAT 可以分为源地址 NAT 和目的地址 NAT。

iptables 通过 iptables 命令设置规则，并将其添加到内核空间的过滤表内的链中。iptables 命令的语法规则如下：

iptables [-t table] command [match] [target]

系统根据链中的规则进行过滤。iptables 包含三个表，filter 管理本机进出、nat 管理后端主机、mangle 管理特殊标志使用。此外还可以自订额外的链。实验中将用到的是表 nat，用作进行来源与目的的替换。其中链 PREROUTING 中为进行路由判断之前所要进行的规则；链 POSTROUTING 中为进行路由判断之后所要进行的规则；链 OUTPUT 中为与发送出去的数据包相关的规则。

NAT 工作的原理是修改 IP，对于一次通讯中 IP 转换的完整过程通过两条链完成，POSTROUTING 链修改来源 IP（SNAT），PREROUTING 链修改目的 IP（DNAT）。内部主机访问外部网络时，SNAT 起作用，替换掉 PDU 中不合法的内部源 IP 地址，外部响应到达

时，DNAT 起作用，替换 PDU 中的目的 IP 地址为内网 IP,再由路由转发给内部主机。不过 iptables 运行时维护有一个表记录了数据包中 IP 的转换，实际中只用使用 POSTROUTING 链即可。

实验目标

本实验的主要目的是让学生能熟练地按照需求配置一个静态的包含多个子网的网络环境，并学会 NAT 的组网方式，为以后的实验的过程中对组网的要求的打下基础。

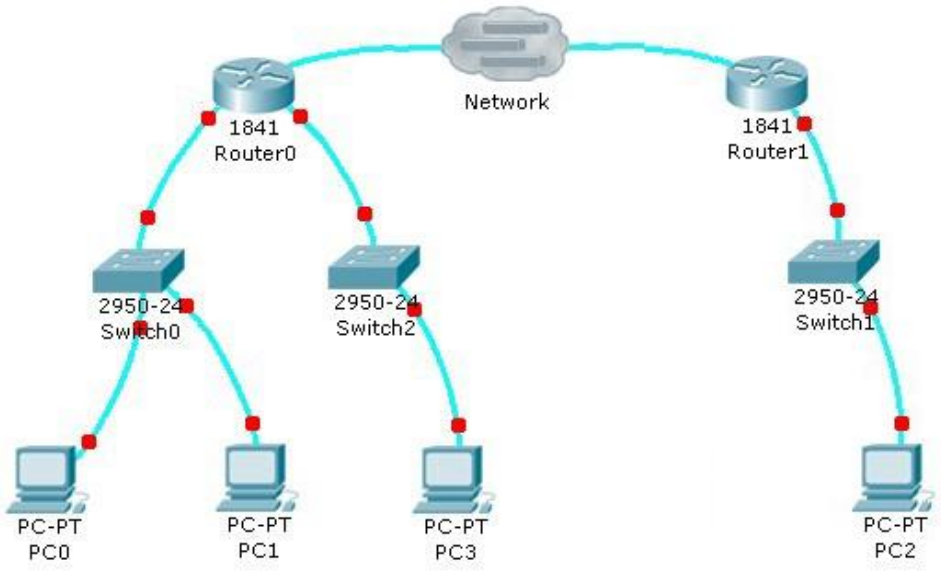
部署，操作/设计步骤（虚拟机器的配置和拓扑）

1 虚拟网络配置

1.1 按照提示安装/拷贝多个虚拟机

1.2 根据拓扑图连接网络

拓扑图如下：



2 设置 IP 与路由规则

令 router0 的网段为 192.168.2.0/24，假设 swich0 连接有 80 台主机，swich2 连接有 20 台主机。请按要求自行划分子网，并为各主机设置 IP，然后根据 IP 添加路由规则。所用到的命令同前一个实验。设置完成后用 ping 命令检测是否联通。

3 用 iptables 进行网络地址转换

在模拟 router0 的虚拟机上使用 iptables 进行网络地址转换。实验中采用静态 NAT。假设 router0 具有外部 IP：210.28.130.166 并用 eth0 与 router1 连接，然后 router0 通过 iptables

进行映射，需要添加链 **POSTROUTING**。使用如下命令设置：

sudo iptables -t nat -A POSTROUTING -o eth0 -s 192.168.2.0/24 -j SNAT --to 210.28.130.166

实验要求完成出网地址映射操作后，在 PC0 上分别 ping PC1，PC2，PC3，用 wireshark 抓包，记录抓取的 PDU，并作简单分析。

注意事项

1、为保证有效的完成实验，可以先在手册中为各个节点分配好 ip 地址和子网掩码，然后再上机实现；

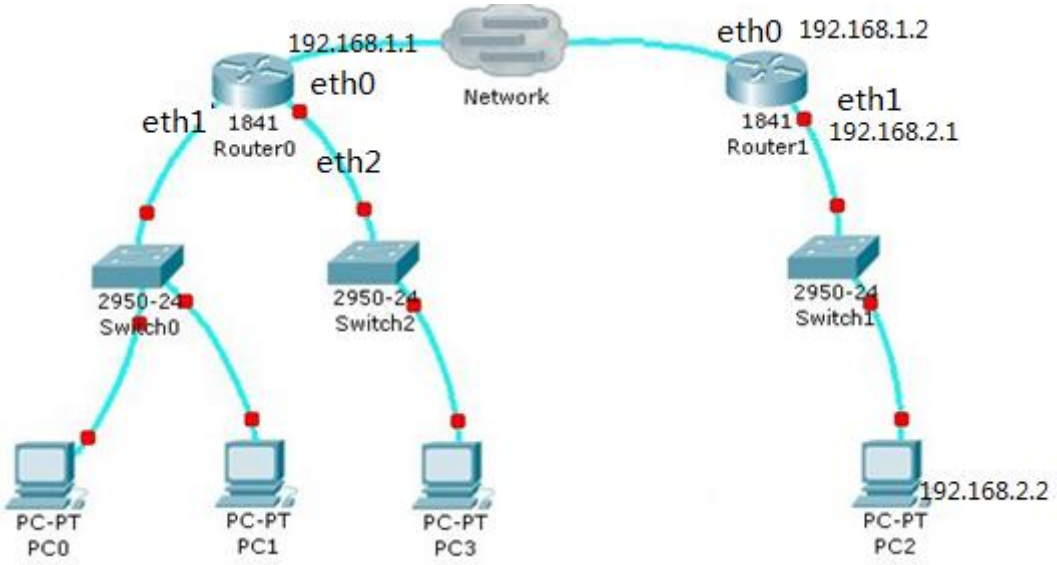
2、建议 VMware 中启动的虚拟机的名称与手册中的各对应节点的名称相同；

3、连接网络过程中，优先配置两台 router 的地址，要注意 router 上有多个网卡，各个网卡对应不同的 VMnet，不要出错，可以先不在 router 上填写路由表，但要确认有默认路由表，设置 ip_forward 的值为 1（见实验一）；

4、配置 PC，注意 PC0，PC1，PC3 的子网掩码为 25 位，并注意填写 PC 默认网关；

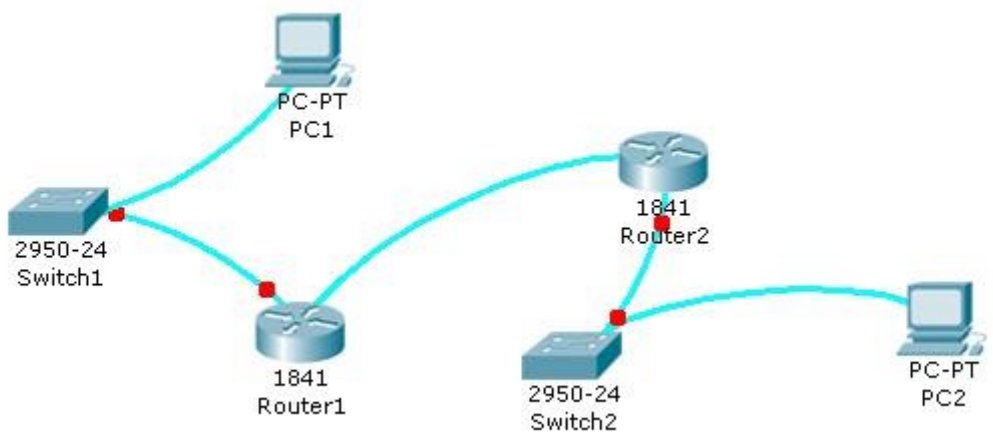
5、完成以上步骤后，图的左部分可以相互通信。右部分可以相互通信，但是图的左半边和右半边不能通信；

6、在 router0 添加路由规则，使得所有目的节点的网络地址为 PC2 所在的网络地址的包通过 router1 的某一端转发，但注意所选取的这一端一定是与 router0 处在同一网段上（如下图 router0 上添加的这条命令为 **sudo ip route add 192.168.2.0/24 via 192.168.1.2**），如果前几步都对，则 router0 ping PC2 成功；



实验四 静态路由编程实现

实验任务



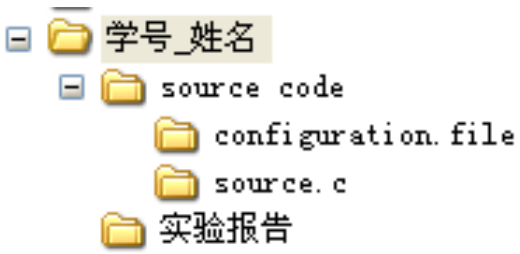
1. 按照上图的网络拓扑，搭建网络，注意，所有 PC 和路由器不要设置 IP 地址。
2. 在 PC1 和 PC2 上建立自己的 ARP 表和 IP 表，用 Raw Socket 实现收发 ICMP 包的程序。
3. 在 Router1 和 Router2 上建立自己的 ARP 表、IP 表和路由表，用 Raw Socket 实现 IP 包的静态路由转发程序。
4. 在每个 PC 和 router 上运行 ifconfig，确保 IP 地址为空，截屏。
5. 用第 2 步的收发包程序，从 PC1 ping PC2，把运行结果以及 Wireshark 的抓包结果进行截屏和分析。

提交材料：实验报告+程序源代码

实验报告

由于本次实验着重程序的设计，试验报告中将不要求统一的配置性细节，大家可以自行设置实验环境。

实验者需提交源代码以及实验报告。提交文件布局如下：



其中 source.c 为 C 的源文件，configuration.file 为相关配置文件。建议同学可以在

source.c 中的重要代码或者函数入口处添加注释，不做定性要求。

实验报告最好提交成 word 文档或 pdf 文档，格式应该包含下表内容，同学可以根据内容进行修改，但是各个项目不可省略。说明是为了更详细的解释各个项目，提交的实验报告中无需包含。

实验目的	

数据结构说明	说明：此处需要解释各自的源代码中的数据结构，以及其用途。
	示例：
	struct xiaomaolv{
	int id;
	floag weight;
	int age;
	};
	该结构是一个描述 xiaomaolv 的结构，对应于现实生活农场中养的毛驴，其中 id 是其在农场中的编号，weight,age 分别代表了该毛驴的质量和年龄。

配置文件说明（非必须）	说明：此处需要给出配置文件的格式以及读取方式，如程序中没有用到配置文件，则该项可省略
程序设计的思路以及运行流程	说明：此处需要给出程序的运行流程或者思路。请给出如下两种格式之一： 1. 流程图 2. 流程说明 示例： 1. 程序开始 2.读取配置 3. 检测数据 4 处理数据 4.1 正确 goto 3 4.2 错误 goto 5 5.程序结束
运行结果截图	说明：请给出你的运行结果截图
相关参考资料	说明：请给出你完成该实验的参考书目或者网页
对比样例程序	说明：请给出你参考样例程序的部分，假如没有参考点，填无
代码个人创新以及思考	说明：请给出你认为你的源代码中的亮点，比如，针对某个细节的处理或者算法的优化
该程序的应用场景创新（非必须）	说明：请思考一下，该类程序除了在背景中的应用之外，是否还有其他可能的应用场景。

背景知识

路由

路由是一种把信息从源穿过网络传递到目的地的行为，在路上，至少遇到一个中间节点。完成路由工作的核心是路由器，路由器是工作在第三层上的设备。路由器一般都包含路由和交换功能，在这里，仅仅讨论路由功能。

路由整体上是由两个部分构成的，路径选择和数据交换。其中路径选择主要是基于路由算法来确定的，不同的路由算法可能会得出不同的路径选择方式。至于数据交换，仅仅是在 2 层上的数据传输，这个交换的前提是按照之前确定的算法进行的。

静态路由

在众多的路由算法分类之中，有一种是静态路由和动态路由的划分。所谓静态路由，就是在网路工作前，由网络管理员事先配置好的网络表映射来确定包的路由方式。而动态路由是一种可以随着网络改变而动态改变路由表的方式。这里要求实现一个静态路由，所以不深入讨论动态路由的实现。

系统实现

在操作系统中，是自动拥有路由表的。

在 shell 下输入 route 命令，可以查看当前的路由表信息：

```
july@july-laptop:~$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.2.0        *               255.255.255.0   U     1      0        0 eth0
link-local      *               255.255.0.0     U     1000   0        0 eth0
default         10.0.2.2        0.0.0.0         UG    0      0        0 eth0
```

图 1

在这张路由表中，Destination, Gateway, Genmask, Iface 分别对应于目的网络，网关，子网掩码，对应的网络接口。

通过 ARP 命令查看系统的 ARP 缓存：

```
july@july-laptop:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.2                 ether   52:54:00:12:35:02   C                     eth0
```

图 2

Address，HWtype, HWaddress, Iface 分别对应于 IP 地址，硬件类型，硬件地址，网络接口。

在一个真正的系统当中，对于一个包的路由过程可以看成如下：首先，一个包要发往一个地址，那么根据该地址在路由表中进行查询，找到对应的网关，然后将包转发到网关上，而转发到网关的过程中，会用到 ARP 缓存来确定以太帧的 MAC 头信息。此时，我们需要动态的更新查找 ARP 缓存来填充头部信息。然后发送。

实验目标

本实验主要目的设计和实现一个简单的静态路由机制，用以取代 Linux 实现的静态路由方式，进而加深对二三层协议衔接及静态路由的理解。

部署，操作/设计步骤

数据结构以及流程实现

回顾在背景知识中介绍的路由过程以及系统实现中的数据结构，我们确定了我们自己的简单路由协议至少要具备如下数据结构：

```
//the information of the static routing table
struct route_item{
    char destination[16];
    char gateway[16];
    char netmask[16];
    char interface[16];
}route_info[MAX_ROUTE_INFO];
// the sum of the items in the route table
int route_item_index=0;
```

一个简单的静态路由表，该表是符合于静态路由协议的，那么也就是说路由路径的选择是通过该静态路由表来确定的，而且该路由表除了手动修改之外在程序运行过程之中不会改变。

```
//the informaiton of the " my arp cache"
struct arp_table_item{
    char ip_addr[16];
    char mac_addr[18];
}arp_table[MAX_ARP_SIZE];
// the sum of the items in the arp cache
int arp_item_index =0;
```

一个可以动态改变的 ARP 缓存

该 ARP 缓存可以在运行的过程中动态的添加删除 arp 表项。除此之外，为了方便，还增加了一个模仿系统网络配置的数据结构。

```
// the storage of the device ,  got information  from configuration file : if.info
struct device_item{
    char interface[14];
    char mac_addr[18];
}device[MAX_DEVICE];
// the sum of the interface
```

int device_index=0;

定义完数据结构之后，我们应该定义一下这个静态路由程序的工作流程，我们需要回顾一下，一个路由器的工作方式。

路由器可以看作是一个重复工作的循环机器，当它启动的时候，管理员会设置它的启动选项，并对它的路由表，路由策略进行选择，他自己也要初始化自己的一些其他类似于操作系统的信息，它监听设备上的许多硬件接口，查看着每一个它收到的包，当一个数据包符合它路由表的条件的时候，它就会根据路由表的信息对其进行转发，在转发的过程由于是通过其他的接口的转发，必然要用到 arp 缓存中的数据项，当一个数据包符合在网络上准确传输的时候，它就会将这个经过它重重修改的包发送到对应的接口上去，这个数据包未来的发展就与它无关了，它所要做的就是继续监听，并重复之前做过的事情。

根据一个路由器的工作流程，我们也可以模仿出我们静态路由程序的工作流程。

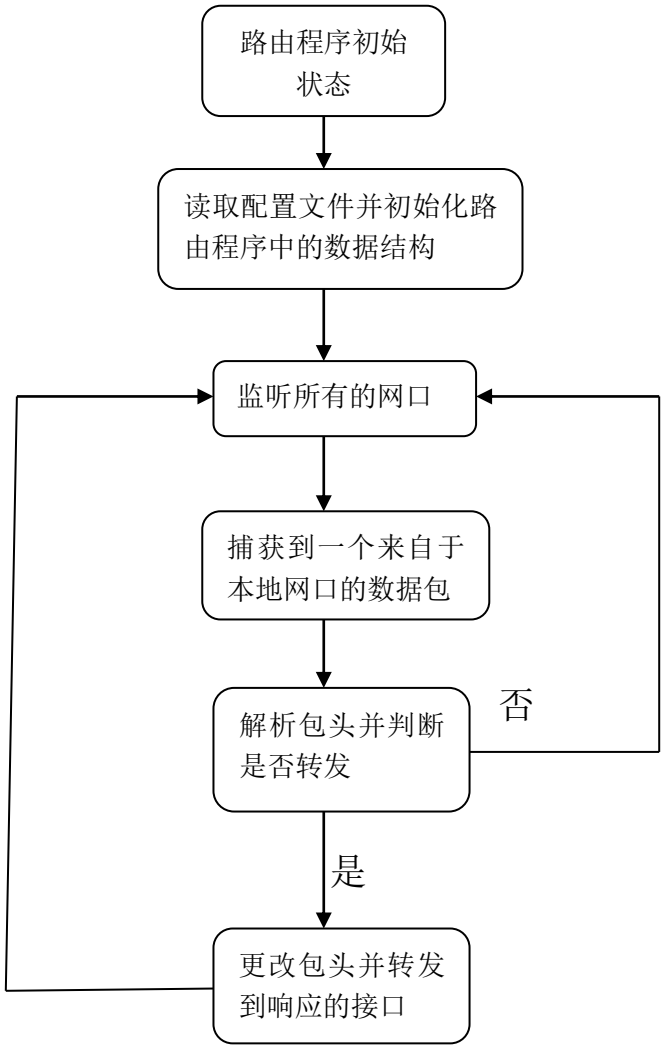


图 5

其中监听网口和我们可以看作是创建一个 socket，并 recvfrom 在这个端口之上，假如收到包的话，就算是捕获到一个来自本地网口的数据包。

实验五 动态路由协议 RIP，OSPF 和 BGP 观察

背景知识

1 自治系统

自治系统（AS，Autonomous System），是一个处于一个或多个管理机构控制之下的路由器和网络群组。一个自治系统中的所有路由器必须相互连接，运行相同的路由协议，所使用的路由协议由系统自主决定，因此有时也被称为是一个路由选择域（routing domain）。自治系统自主决定用于进行网络内部路由信息通信的协议称为内部网关协议（IGP，Interior Gateway Protocols），而各个自治系统网络之间则是通过边界网关协议（BGP，Border Gateway Protocol）来共享路由信息。每个自治系统都会被分配一个全局的唯一的号码，自治系统号（ASN）。这个号码用于标识出一个自治系统，以支持 BGP。

2 内部网关协议

内部网关协议（IGP）是一种专用于一个自治系统中网关间交换数据流转通道信息的协议。网络 IP 协议或者其他的网络协议常常通过这些通道信息来决断怎样传送数据流。目前的内部网关协议有 RIP、OSPF、IGRP、EIGRP、IS-IS 等协议。其中最常用的两种内部网关协议分别是：路由信息协议（RIP）和最短路径优先路由协议（OSPF）。

2.1 路由信息协议

路由信息协议（RIP，Routing Information Protocol）是应用较早、使用较普遍的内部网关协议，它采用距离向量算法，适用于小型网络。在默认情况下，RIP 使用跳跃计数(hop count)作为来衡量路由距离，跳跃计数是一个包到达目标所必须经过的路由器的数目。如果到相同目标有二个不等速或不同带宽的路由器，但跳跃计数相同，则 RIP 认为两个路由是等距离的。跳跃计数取值为 1~15，数值 16 表示无穷大。RIP 使用 UDP 的 520 端口来发送和接收 RIP 报文。RIP 报文每隔 30s 以广播的形式发送一次，为了防止出现“广播风暴”，其后续的的报文将做随机延时后发送。在 RIP 中，如果一个路由在 180s 内未被刷新，则相应的距离就被设定成无穷大，并从路由表中删除该表项。RIP 报文分为两种：请求报文和响应报文。

2.2 开放式最短路径优先

开放式最短路径优先（OSPF，Open Shortest Path First）是另一种被广泛使用的内部网关协议。OSPF 根据域中的链路状态来决策路由，计算出最短路径树，通常多用于较大型的网络。OSPF 协议同时使用单播（unicast）和多播（multicast）来发送 Hello 包和连接状态更新（link state updates），使用的多播地址为 224.0.0.5 和 224.0.0.6。不同于 RIP 的是，OSPF 协议直接使用 IP 协议，并将链路状态广播数据包传送给在某一区域内的所有路由器，而不是将部分或全部的路由表传递给与其相邻的路由器。

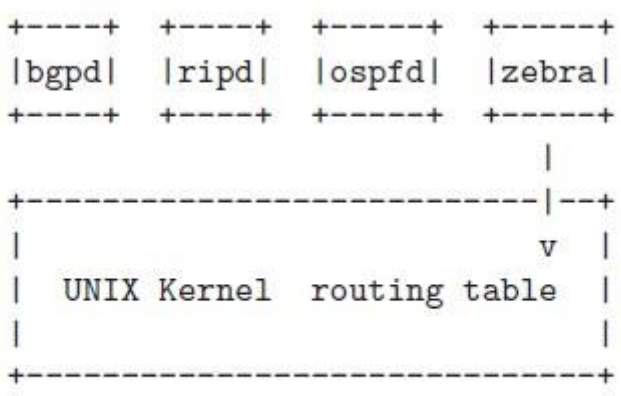
3 边际网关协议

边界网关协议（BGP，Border Gateway Protocol）是互联网的核心路由协议。它通过维护路由表来实现自治系统之间的可达性，属于向量路由协议。BGP 不使用传统域内路由协议的距离度量，而是基于路径、网络策略和规则集来决定路由。BGP 通过在路由器上手工设置来使用 TCP 的 179 号端口来发送和接收报文。BGP 路由器会周期地发送 19 字节的保持

存活报文来维护连接（默认周期为 60 秒）。当 BGP 在一个自治系统内部运行时，它被称作内部边界网关协议（iBGP，Interior Border Gateway Protocol）；当 BGP 在 AS 之间运行时，它被称作外部边界网关协议（eBGP，Exterior Border Gateway Protocol）。

4 Quagga

Quagga 是一个提供了基于 TCP/IP 协议的路由服务的软件包。Quagga 工作在 Uinx 平台上，是 GNU Zebra 的分支之一。除可提供静态路由服务外，Quagga 还支持 RIPv1，RIPv2，RIPng，OSPFv3，BGP-4，BGP-4+等动态路由协议。Quagga 由多个守护进程组成，结构如图所示：



Quagga System Architecture

其中核心的是 zebra，它可以读取和更新内核路由表，为其它守护进程提供操作 Unix 和 TCP 数据流的 API。Ospfd 实现了 OSPFv2 协议；ripd 实现了 RIPv1 和 RIPv2 协议；ospf6d 实现了 OSPFv3 协议；ripngd 实现了 RIPng 协议；bgpd 实现了 BGPv4 和 BGPv4+协议。

实验目标

理解自治系统（AS），观察 RIP，OSPF 以及 BGP 动态路由协议的实际运行过程。在网络拓扑结构变更的情况下观察路由表的动态变更，通过实验理解路由选择算法。

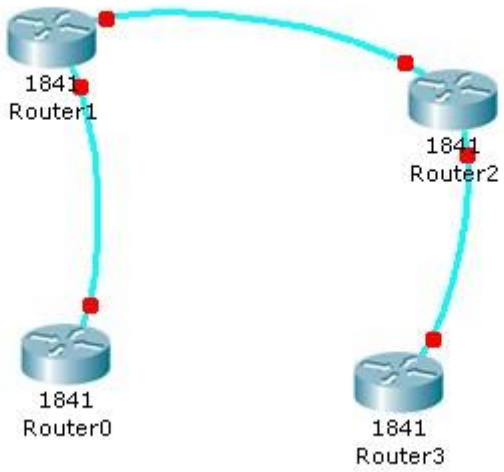
实验任务

1 RIP 协议观察

1.1 按照提示安装/拷贝多个虚拟机

1.2 根据拓扑图连接一个简单链状网络

拓扑图如下：



注意：请为 router0 预留一个网口，为 router3 预留两个网口

1.3 给每个连接在网络上的网卡设置 ip
使用 ifconfig 命令进行设置，或通过 zebra.conf 文件进行设置（见 1.4）

1.4 运行 RIP 协议
1.4.1 修改/etc/quagga/目录下的 daemons 文件，以启用 RIP。
将 daemons 文件中的"zebra=no"和"ripd=no"改为"zebra=yes"和"ripd=yes"

1.4.2 在/etc/quagga/目录下添加两个配制文件 zebra.conf 和 ripd.conf。
1)本次实验中我们只需进行简单的配制，若已通过 ifconfig 命令为网卡设置过 ip，则可以直接使用 quagga 的配置示例，复制示例中的配制文件，使用命令如下：

sudo cp /usr/share/doc/quagga/examples/zebra.conf.sample /etc/quagga/zebra.conf

2)若未使用 ifconfig 命令。可通过使用 zebra.conf 文件设置，内容如下：

```
!-*-zebra-*-
hostname router
password zebra
enable password zebra
log stdout
!
interface network
   description Interface to External Network
   ip address a.b.c.d/m
!
interface network
   description Interface to Internal Network
   ip address a.b.c.d/m
!
```

注意，请根据具体情况选择使用"External Network" 连接外部网络，"Internal Network" 连接

内部网络；"ip address *a.b.c.d/m*"中 *a.b.c.d/m* 表示网络设备的 ip 地址和子网掩码，如 10.0.0.1/8。

建立 rip 配置文件 ripd.conf，内容如下：

```
!-*-rip-*-
hostname ripd
password zebra
router rip
   network network
log stdout
!
```

注意，若有多个网络设备要运行 RIP 协议，则要在配置文件中写入多条"network *network*"，*network* 表示网络设备，如 eth0。

1.4.3 启动 wireshark 准备抓取报文，然后启动 zebra，ripd 两个进程，使用命令如下 ：

sudo /etc/init.d/quagga restart

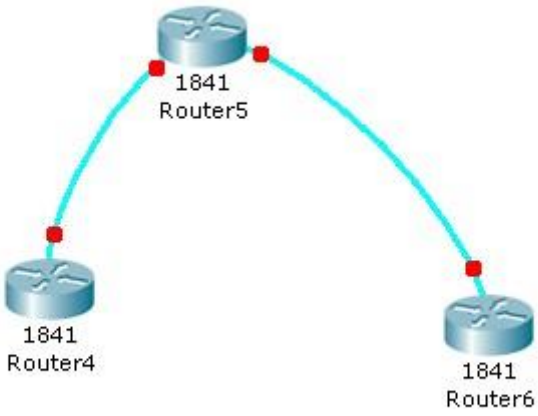
1.5 观察 RIP 报文

2 OSPF 协议观察

2.1 按照提示安装/拷贝多个虚拟机

2.2 根据拓扑图连接一个简单链状网络

拓扑图如下：



注意：请为 router4 预留一个网口

2.3 给每个连接在网络上的网卡设置 ip

设置方式同 1.

2.4 运行 OSPF 协议

2.4.1 修改/etc/quagga/目录下的 daemons 文件，以启用 OSPF。
将 daemons 文件中的"zebra=no"和"ospfd=no"改为"zebra=yes"和"ospfd=yes"

2.4.2 在/etc/quagga/目录下添加两个配制文件 zebra.conf 和 ospfd.conf。
设置方式同 1。

建立 ospf 配置文件 ospfd.conf，内容如下：

```
!-*-ospf-*-
hostname ospfd
password zebra
router ospf
   network a.b.c.d/m area 0
log stdout
!
```

注意，若有多个网络设备要运行 OSPF 协议，则要在配置文件中写入多条"network *a.b.c.d/m* area 0"，*a.b.c.d/m* 表示网络设备所处的网络，如 192.168.1.0/24。

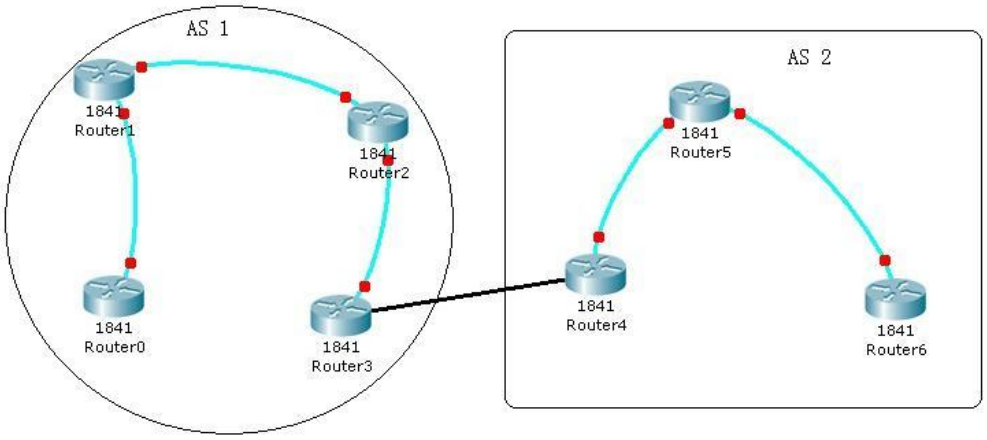
2.4.3 启动 wireshark 准备抓取报文，然后启动 zebra，ospfd 两个进程，使用命令如下：

sudo /etc/init.d/quagga restart

2.5 观察 OSPF 报文

3 BGP 协议观察

3.1 根据拓扑图连接两个 AS
拓扑图如下：



3.2 给两个新连接的网卡设置 ip
设置方式同 1。

3.3 运行 BGP 协议

3.3.1 分别修改 router3 和 router4 中/etc/quagga/目录下的 daemons 文件，以启用 BGP。将 daemons 文件中的"bgpd=no"改为"bgpd=yes"

3.3.2 在/etc/quagga/目录下添加配制文件 bgpd.conf。

bgpd 配置文件 bgpd.conf 内容如下：

```
!-*-bgp-*-
hostname bgpd
password zebra
router bgp asn
   bgp router-id a.b.c.d
   network a.b.c.d/m
   neighbor a.b.c.d remote-as asn
log stdout
!
```

其中"router bgp *asn*"中的 *asn* 为 AS 号，例如 100。"bgp router-id *a.b.c.d*"中 *a.b.c.d* 为网络设备的 IP。"network *a.b.c.d/m*"中 *a.b.c.d/m* 为 AS 内部网络地址的集。"neighbor *a.b.c.d* remote-as *asn*"中 *a.b.c.d* 和 *asn* 分别为相邻 AS 的网络设备 IP 和 AS 号。

3.3.3 启动 wireshark 准备抓取报文，然后启动 bgpd 进程，使用命令如下：

sudo /etc/init.d/quagga restart

3.4 观察 BGP 报文

4 观察路由表动态变更

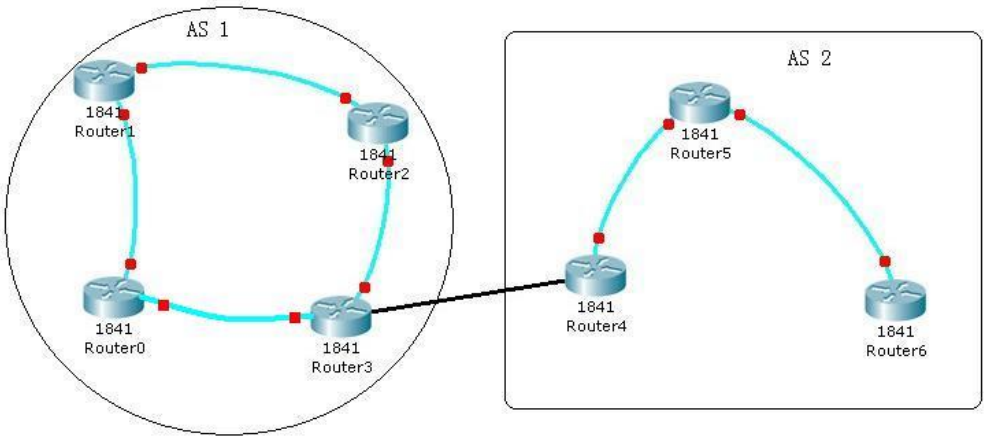
4.1 观察初始网络情况

观察 router0 的路由表，并追踪 router0 到 router3 的包传输路径，使用命令如下：

tracepath *ip*

4.2 添加一条连接，变更原来的网络结构

连接 router0 和 router3，拓扑图如下：



4.3 观察变更后网络情况

4.3.1 给两个新连接的网卡设置 ip，设置方式同 1。

4.3.2 在 router0，router3 中的 ripd.conf 文件中加入"network *network*"，重启 quagga。

4.3.3 观察 router0 的路由表，并追踪 router0 到 router3 的包传输路径，使用命令同 4.2

实验报告

按要求完成实验，并写出实验报告。实验报告格式如下，同学可以根据内容进行修改。说明是为了更详细的解释各个项目，提交的报告中无需包含。

实验目的	
网络拓扑配置	说明：填写附表 1，也可绘图说明
路由配置文件	说明：即 zebra.conf，ripd.conf，ospfd.conf，bgpd.conf。根据实际配置给出实验结束时 router0，router3，router4 和 router6 中的配置文件即可
数据包截图	说明：用 wireshark 抓包截图，RIP，OSPF，BGP 报文各一，需要标明抓包的路由器和端口
协议报文分析	说明：分析抓取的报文
观察动态路由	说明：比较网络变更前后 router0 的路由表

附表 1：

节点名	虚拟设备名	ip	netmask
Router0		eth0:	
		eth1:	
Router1		eth0:	
		eth1:	
Router2		eth0:	
		eth1:	
Router3		eth0:	
		eth1:	
		eth2:	
Router4		eth0:	
		eth1:	
Router5		eth0:	
		eth1:	
Router6		eth0:	

实验六 VPN 设计、实现与分析

背景知识

VPN，Virtual Private Network，也就是虚拟专用网。是一种通过一个公用网络建立一个临时的，安全的连接，是一条穿过混乱的公用网络的安全，稳定的隧道。常用的虚拟专用网协议有 IPSec，PPTP，L2F，L2TP 以及 GRE。IPSec 是 IP Security 的缩写，是保护 IP 协议安全通信的标准，它主要对 IP 协议分组进行加密和认证。PPTP 的全称是 Point to Point Tunneling Protocol 点到点隧道协议。L2F 是 Layer 2 Forwarding,也就是第二层转发协议的缩写。L2TP 是 Layer 2 Tunneling Protocol 第二层隧道协议的缩写。GRE 是 VPN 的第三层隧道协议。

当前 VPN 的需求很大，原因是随着集团公司规模的扩大以及其他的一些专用需求，特定的私有网络得到很大的青睐，但是单独建设私有网络或者租用私有网络的成本很高，那么此时通过 VPN 实现的，建立在普通互联网技术之上的私有网络技术得到很多 IT 部门的认可。

下面给出一些参考资料，大家可以查看他们来对 VPN 获得更加深入的认识。
http://en.wikipedia.org/wiki/Virtual_private_network
http://computer.howstuffworks.com/vpn.htm
同时大家如果有兴趣，可以参加开源 VPN 软件 OpenVPN 的开发
www.openvpn.net
另外还有一些有关 VPN 实现的 RFC 文档
http://www.ietf.org/rfc/rfc2637.txt

实验目标

本实验主要目的是设计和实现一个简单的虚拟专用网络的机制， 并与已有的标准实现（如 PPTP）进行比较， 进而让学生进一步理解 VPN 的工作原理和内部实现细节。

实验任务

1 实验环境搭建

首先明确我们的目的，我们是要实现一个简易的 VPN，从 VPN 的定义来看，我们首先需要一个单独的网络来模拟因特网，其次对于每个连接 VPN 的主机，都需要一个 VPN 入口，也就是一个 VPN 接入服务器，通过该服务器来实现主机的 VPN 接入，该服务器负责将主机的发送包进行重封装并传递到目标 VPN 主机，并接受传递到自身下属主机的

VPN 包，并跟据 VPN 解包规范进行包的解析，并传递要对应的 VPN 主机。

整个网络的拓扑如下：

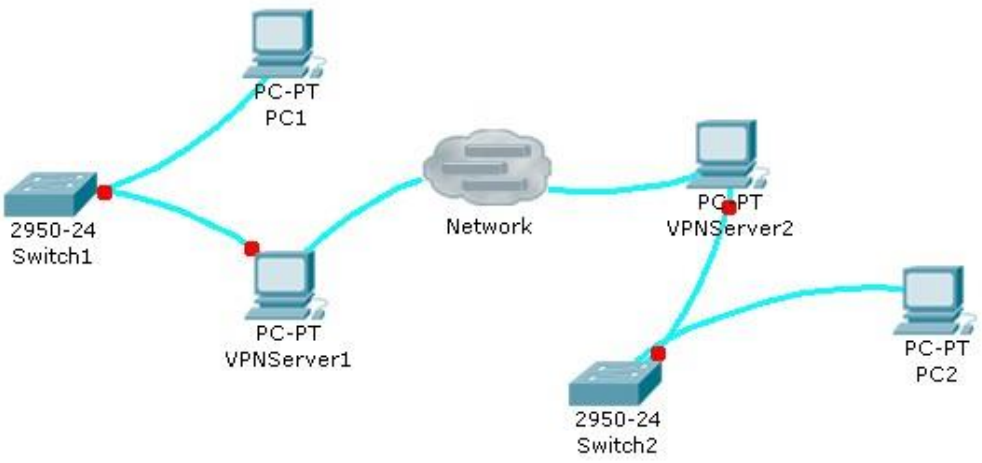


图 1

其中 VPNServer1，Network，VPNServer2 之间的连接可以看作是整个互联网，因为互联网只不过是众多的这样的连接的集合。

其中 PC 1，PC 2 是接入 VPN 的 2 个主机。VPNServer1 和 VPNServer2 是 2 个 VPN 的接入口，实际上，我们要实现的 VPN 程序就是运行在 2 台机器上的。Network，一个配置了路由转发的主机，模拟了整个互联网络。我们最终的要求是 PC 1 可以通过 VPN 的 IP 地址同 PC 2 进行交互。2 者通过系统的网络配置无法互相沟通，而当在 VPNServer1 和 VPNServer2 上运行 VPN 程序的时候，二者可以通过 VPN 虚拟出来的 IP 进行通讯。为了完成以上的拓扑，做出如下配置。（默认 eth0 为左，eth1 为右）。

PC 1 网络配置

PC 1#**ifconfig eth0 10.0.0.2 netmask 255.255.255.0**

PC 1#**route add default gw 10.0.0.1**

PC 1# **sudo /etc/init.d/networking restart**

PC 2 的配置同 PC 1 的配置类似

PC 2#**ifconfig eth0 10.0.1.2 netmask 255.255.255.0**

PC 2#**route add default gw 10.0.1.1**

PC 2# **sudo /etc/init.d/networking restart**

下面配置 2 个 VPN 接入服务器 VPNServer1，VPNServer2

VPNServer1#**ifconfig eth0 10.0.0.1 netmask 255.255.255.0**

VPNServer1#**ifconfig eth1 192.168.0.2 netmask 255.255.255.0**

VPNServer1#**route add default gw 192.168.0.1**

VPNServer1# **sudo /etc/init.d/networking restart**

VPNServer2#**ifconfig eth0 172.0.0.2 netmask 255.255.255.0**
VPNServer2#**ifconfig eth1 10.0.1.1 netmask 255.255.255.0**
VPNServer2#**route add default gw 172.0.0.1 netmask 255.255.255.0**
VPNServer2# **sudo /etc/init.d/networking restart**

下面配置里面的虚拟路由器，用来模拟整个因特网的节点 Network
Network#**ifconfig eth0 192.168.0.1 netmask 255.255.255.0**
Network #**ifconfig eth1 172.0.0.1 netmask 255.255.255.0**
Network #**echo 1 > /proc/sys/net/ipv4/ip_forward**
Network # **sudo /etc/init.d/networking restart**

2 程序实现

当整个拓扑搭建完成之后，下面我们要做的事就是实现这个简单的 VPN 程序。首先，我们要明确的是在实际的 Internet 上面，跑的是标准的 IP 包，IP 包封装的才是真正的 VPN 包。我们可以认为，在 VPN 接入点上，就是将 VPN 包进行重新封装，并传递到网络上。直到某个 VPN 接入点接收到该包，解析包头确定其确是属于某个 VPN 的包，然后将其重新解包并传递到内部的 VPN 节点上。一个基本的包在 VPN 网络上传递的流程如下：

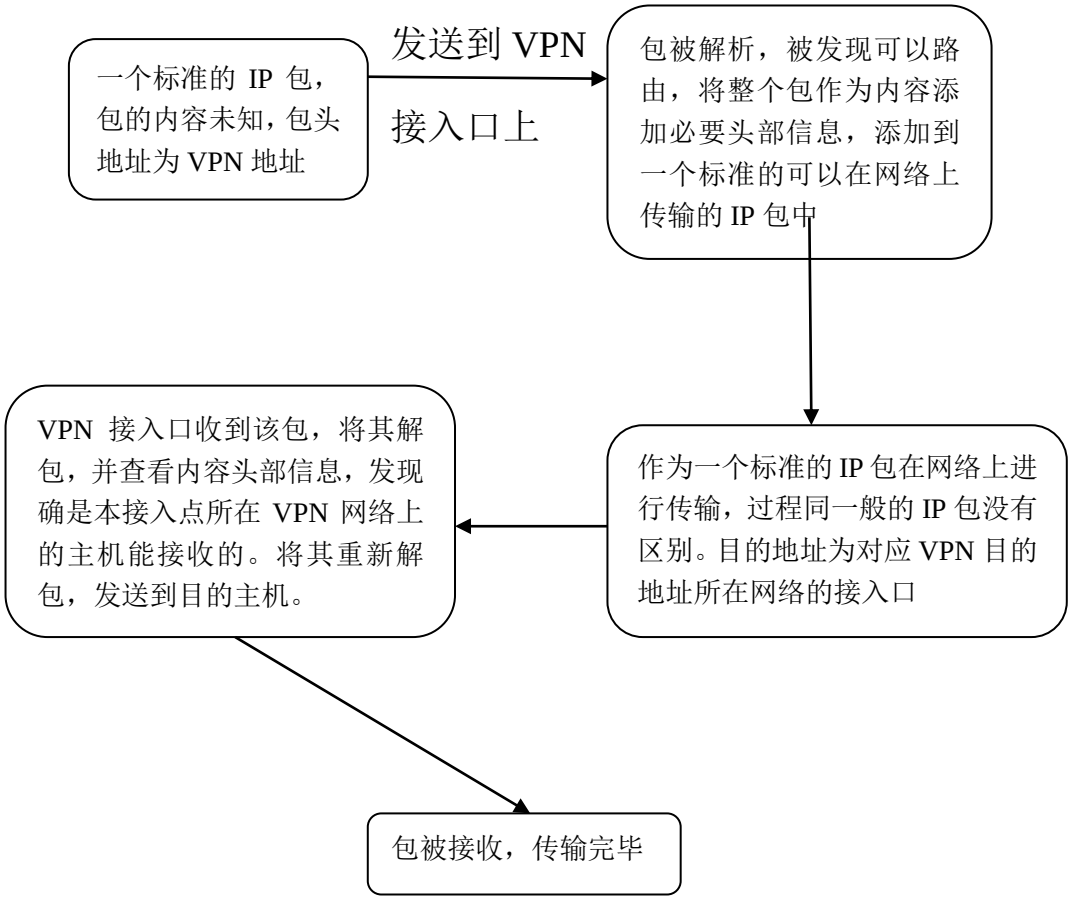


图 2

从 VPN 接入点的角度来看，有如下流程图

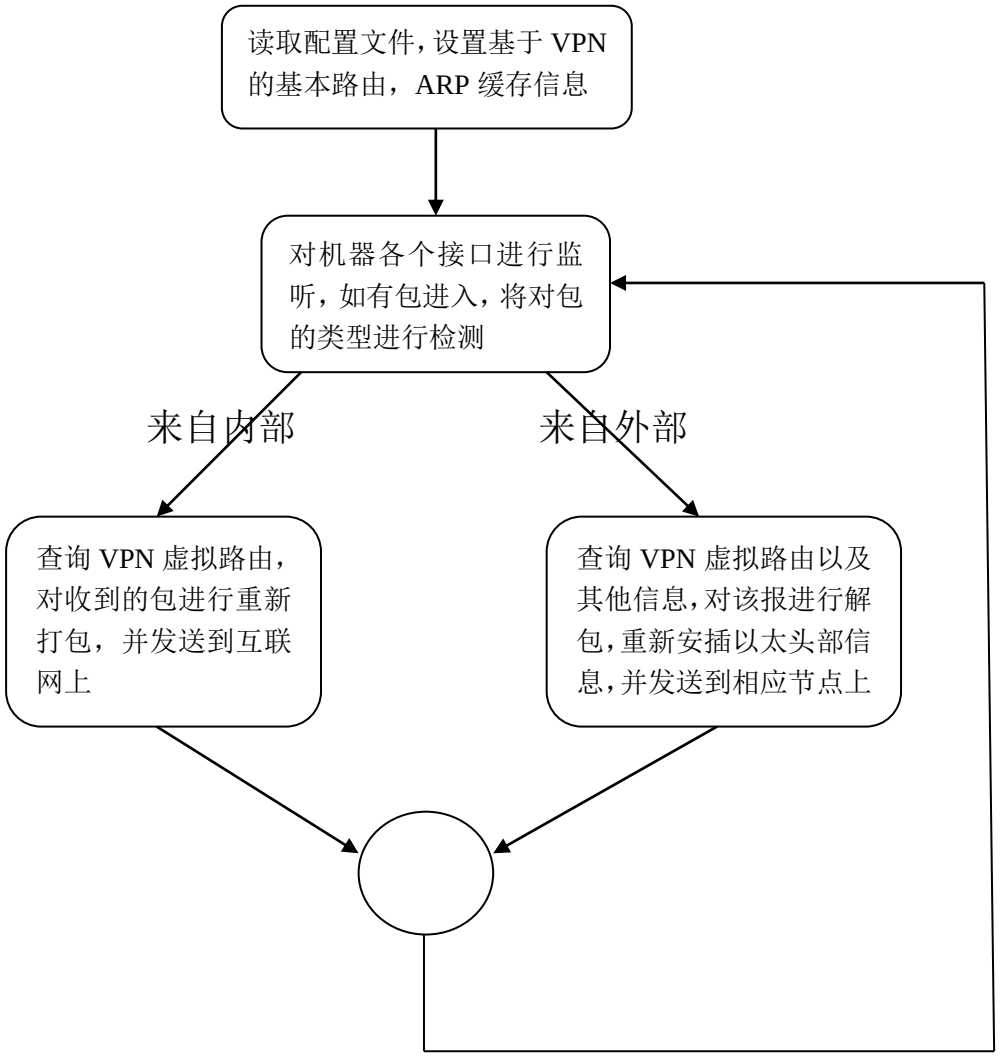


图 3

由上面的图我们可以看到，该程序仍然相当于一个路由程序，只不过需要重新封装基于 VPN 的包内容。联系实验 2，我们可以借用简单路由程序的数据结构。由于 VPN 节点需要明确指出，所以，在这里需要更改。

```
typedef struct device_ti{
    char interface[8];
    char mac_addr[6];
    int is_entrance;
}DEVICE_INFO_ITEM;
```

红色字体部分为新增的数据结构内部项，其他的都同简单路由协议。
针对包的重新封装和 VPN 包的解析工作主要集中在 2 个函数中实现

int repack_packet(char* buffer, int buffer_length,

char* error_info,　DEVICE_INFO_ITEM*　device_item_table,
int device_table_size,
ROUTE_TABLE_ITEM* route_item_table, int route_table_size);

int unpack_packet(char* buffer, int　buffer_length,
char* error_info,　DEVICE_INFO_ITEM* device_item_table,int device_table_size,
ROUTE_TABLE_ITEM* route_item_table, int route_table_size,
ARP_TABLE_ITEM* arp_table,int arp_table_size)；

函数的作用由函数名标识。源代码详见 vpn_all.c

3 测试验证

首先需要编译程序，在 VPNServer1 和 VPNServer2 编译并运行程序。
VPNServer1#gcc vpn_all.c –o vpn_all
VPNServer1#./vpn_all
在 VPNServer2 的操作同 VPNServer1
然后从 PC 1 上进行操作
PC 1# ping 10.0.1.2 （10.0.1.2 是 PC 2 的 IP）
会看到如下回应

```
[root@localhost ~]# ping 10.0.1.2
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.
64 bytes from 10.0.1.2: icmp_seq=1 ttl=64 time=14.7 ms
64 bytes from 10.0.1.2: icmp_seq=2 ttl=64 time=9.75 ms
64 bytes from 10.0.1.2: icmp_seq=3 ttl=64 time=12.9 ms
64 bytes from 10.0.1.2: icmp_seq=4 ttl=64 time=12.0 ms
64 bytes from 10.0.1.2: icmp_seq=5 ttl=64 time=13.0 ms
64 bytes from 10.0.1.2: icmp_seq=6 ttl=64 time=101 ms
^C
--- 10.0.1.2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5168ms
rtt min/avg/max/mdev = 9.756/27.394/101.778/33.298 ms
```

图 4

为了确保不是系统自己的路由功能，将 VPNServer1 或者 VPNServer2 上的任何一个 VPN 程序终止，就会发现

```
[root@localhost ~]# ping 10.0.1.2
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.
64 bytes from 10.0.1.2: icmp_seq=1 ttl=64 time=24.5 ms
64 bytes from 10.0.1.2: icmp_seq=2 ttl=64 time=21.2 ms
64 bytes from 10.0.1.2: icmp_seq=3 ttl=64 time=11.9 ms
64 bytes from 10.0.1.2: icmp_seq=4 ttl=64 time=14.7 ms
64 bytes from 10.0.1.2: icmp_seq=5 ttl=64 time=12.0 ms
^C
--- 10.0.1.2 ping statistics ---
10 packets transmitted, 5 received, 50% packet loss, time 9412ms
rtt min/avg/max/mdev = 11.947/16.917/24.515/5.085 ms
```

图 5

注：前部分收到的包是因为当时没有终止。

在实验过程中如果不能达到联通效果，请确保针对 VPNServer1，Network，VPNServer2 的 iptables 服务被关闭或者重新配置符合该实验。

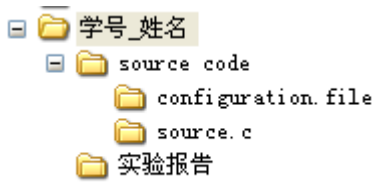
VPN 的一个经典实现是 PPTP，PPTP 的全称是点对点隧道协议，是一种支持多协议虚拟专用网络的网络技术，它工作在第二层。PPTP 是将 PPP 帧封装在 IP 数据包中，通过 IP 网络如 Internet 或者企业专用 Intranet 等发送的。而本程序的实现，是通过将一个以太帧的前 14 位（对应于 MAC 的目的地址和源地址）进行重新修改，然后将修改过的以太帧作为一个整体封装在一个 IP 数据包内在网络上传输。修改以太帧的前 14 位的目的是标识其为一个 VPN 包数据。整体来说，实现的整体思路是差不多的，不同的是封装在 IP 数据包内的数据内容不是完全相同。

实验者需要理解 VPN 的传输原理，并根据样例程序选择重新实现基于第二层或者第三层的 VPN 程序, 在标准的 VPN 网络中，除了 VPN 的接入点外，还有一个 VPN 中心服务器，用来存放 VPN 的路由，拓扑信息，实验者需根据个人情况进行设计添加。另外，作为一个递进式的网络实验，实验者需要将之前的路由程序应用于自己构建的程序测试拓扑网络之中。

实验报告

由于本次实验着重程序的设计，试验报告中将不要求统一的配置性细节，大家可以根据上面的环境进行模拟或者自行设置实验环境。

实验者需提交源代码以及实验报告。提交文件布局如下：



其中 source.c 为 C 的源文件，configuration.file 为相关配置文件。建议同学可以在 source.c 中的重要代码或者函数入口处添加注释，不做定性要求。

实验报告最好提交成 doc 或者 docx 格式的 word 文档，格式如下，同学可以根据内容进行修改，但是各个项目不可省略。说明是为了更详细的解释各个项目，提交的试验

包中无需包含。

实验目的	

数据结构说明	说明：此处需要解释各自的源代码中的数据结构，以及其用途。
	示例：
	struct xiaomaolv{
	int id;
	floag weight;
	int age;
	};
	该结构是一个描述 xiaomaolv 的结构，对应于现实生活农场中养的毛驴，其中 id 是其在农场中的编号，weight,age 分别代表了该毛驴的质量和年龄。

配置文件说明（非必须）	说明：此处需要给出配置文件的格式以及读取方式，如程序中没有用到配置文件，则该项可省略
程序设计的思路以及运行流程	说明：此处需要给出程序的运行流程或者思路。请给出如下两种格式之一： 1. 流程图 2. 流程说明 示例： 1. 程序开始 2. 读取配置 3. 检测数据 4. 处理数据 4.1 正确 goto 3 4.2 错误 goto 5 5. 程序结束
运行结果截图	说明：请给出你的运行结果截图
相关参考资料	说明：请给出你完成该实验的参考书目或者网页
对比样例程序	说明：请给出你参考样例程序的部分，假如没有参考点，填无
代码个人创新以及思考	说明：请给出你认为你的源代码中的亮点，比如，针对某个细节的处理或者算法的优化
该程序的应用场景创新（非必须）	说明：请思考一下，该类程序除了在背景中的应用之外，是否还有其他可能的应用场景。

实验七 TCP 协议的拥塞控制机制观察

实验任务

仔细阅读课本中关于 TCP 拥塞控制的介绍以及拥塞控制算法的状态自动机，完成以下任务。

1. 利用 Wireshark 记录若干 TCP 短流（少于 5 秒，如访问 web 页面，收发邮件等）和 TCP 长流（长于 1 分钟，如 FTP 下载大文件，用 HTTP 观看在线视频等）。
2. 对于每个 TCP 流，画出其 congestion window 随时间的变化曲线，并指出拥塞控制的慢启动、拥塞避免、快恢复等阶段。
3. 画出每个 TCP 流的瞬时吞吐量，并统计其平均吞吐量和丢包率。

提交材料：实验报告

附录 1. Linux 命令列表

附录 2. raw_socket