

《网络技术与应用》课程大作业

实验三：简单的路由程序设计

姓名：王科

学号：1310583

专业：计算机科学与技术

完成日期：2015/12/23

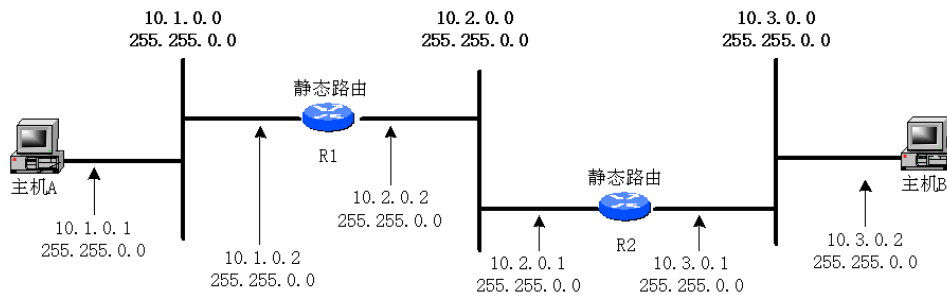
目录

一、	实验环境	2
1.1	实验要求	2
1.2	路由软件应该处理的内容	2
1.3	路由选择的基本原理	3
1.3.1	IP 互联网采用表驱动的路由选择算法	3
1.3.2	IP 路由选择利用 IP 地址隐藏主机的信息	3
1.4	相关的 ICMP 报文	4
1.4.1	目的不可达报文	4
1.4.2	超时报文	4
1.5	实验编译运行环境	4
二、	“简单的路由器”程序的编写	5
2.1	程序界面	5
2.2	程序实现的功能	6
2.3	路由程序的设计	6
2.3.1	路由器初始化模块	8
2.3.2	静态路由表维护模块	13
2.3.3	数据包捕获与处理模块	14
三、	实验结果	19
1、	程序检测环境的构建	19
2、	程序的正确性检测	20
四、	实验总结与心得体会	23
1.	实验时遇到的困难	23
2.	实验的心得体会	23

一、 实验环境

1.1 实验要求

本实验的目的是利用 visual C++编写一个简单的路由程序，实现 IP 数据报的转发。本实验可以在一个局域网中进行，采用如下图所示的实验环境：



其中路由 R1 和 R2 是连接不同子网的通用计算机，通过 R1 和 R2 上运行编制的路由程序，绑定双 IP，通过路由程序添加路由表，对接收的数据包进行分析和转发，实现处于不同网络中的主机 I 和 II 的相互通信。通信测试由 ping 命令执行，可以通过 tracer +IP 地址来追踪数据包在网络中的转发路径。

1.2 路由软件应该处理的内容

编制一个较为完整的路由软件相当复杂包含如下工作：

1. 为经过的 IP 数据报选择路由：路由器基本功能，对接受的 IP 数据报提取目的 IP 地址，根据自己的路由表信息为该数据报选择最优的转发路径。
2. 处理 IP 数据报的 TTL 域中的数值，抛弃 $TTL \leq 0$ 的数据报，将转发的 IP 数据报 TTL 减一。
3. 分片处理。
4. 处理 IP 数据报选项：处理 IP 数据报可能的选项（记录路由、源路由、时间戳等）。
5. 重新计算 IP 数据报的头部校验和：由于路由软件需要进行 TTL 处理、分片处理、选项处理等工作，因此，需要送出的 IP 数据报报头需要重新计算校验和。
6. 生成和处理 ICMP 报头
7. 实现路由协议，维护静态路由。

8. 实现 ARP 协议，形成数据帧：在将一个 IP 数据报送往下一跳步之前，路由处理软件需要获取下一站的物理地址，然后形成数据帧从选择的网络接口发送出去。

其中，简单的路由软件可以忽略分片处理、选项处理、动态路由等功能的实现，将精力重点放在路由的选择与 IP 数据报的转发。

1.3 路由选择的基本原理

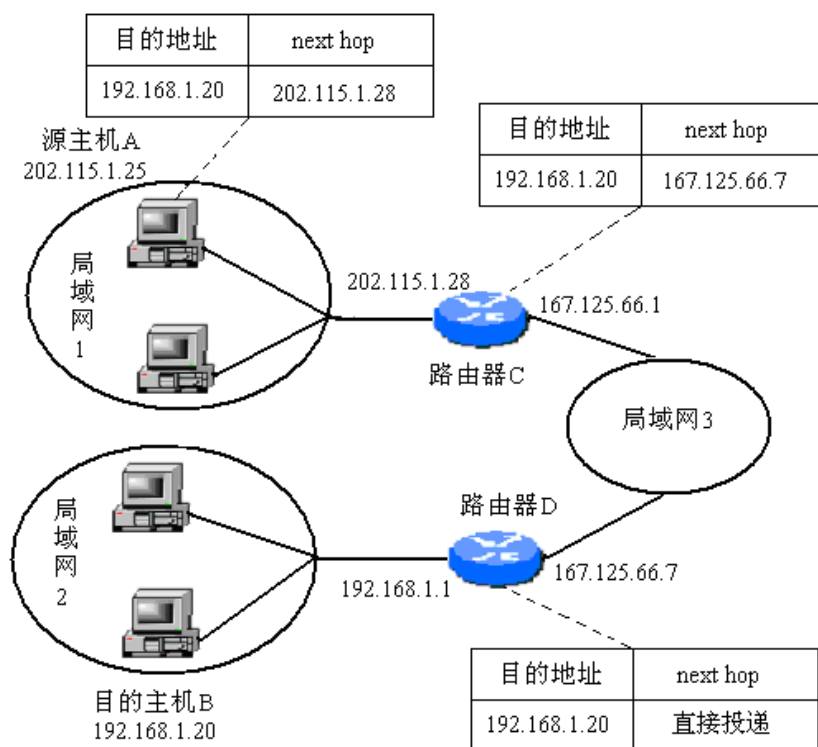
1. IP 互联网采用表驱动的路由选择算法

- 需要路由选择的设备保存一张 IP 路由表
- 路由表存储有关目的地址及怎样到达目的地址的信息
- 通过查询路由表决定把数据报发往何处

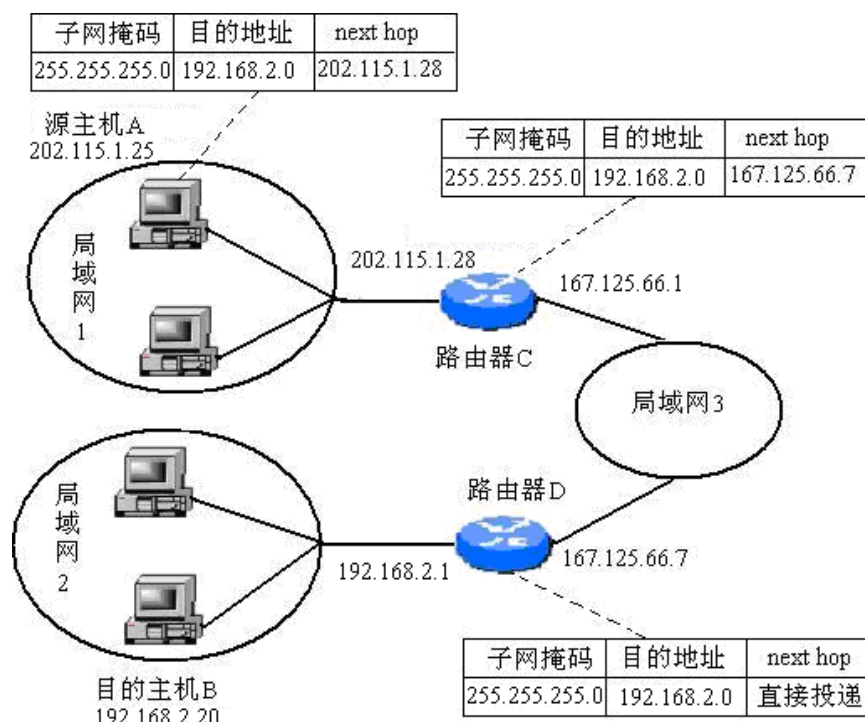
2. IP 路由选择利用 IP 地址隐藏主机的信息

- 连接到同一网络的所有主机共享同一网络号

一个标准的路由选择算法示意图如下：



一个子网选路的示意图如下：



1.4 相关的 ICMP 报文

1. 目的不可达报文

- 当路由器不能为数据包找到路由器或主机交付数据包时，就丢弃该数据包，然后向源主机发出 ICMP 目的不可达报文。

2. 超时报文

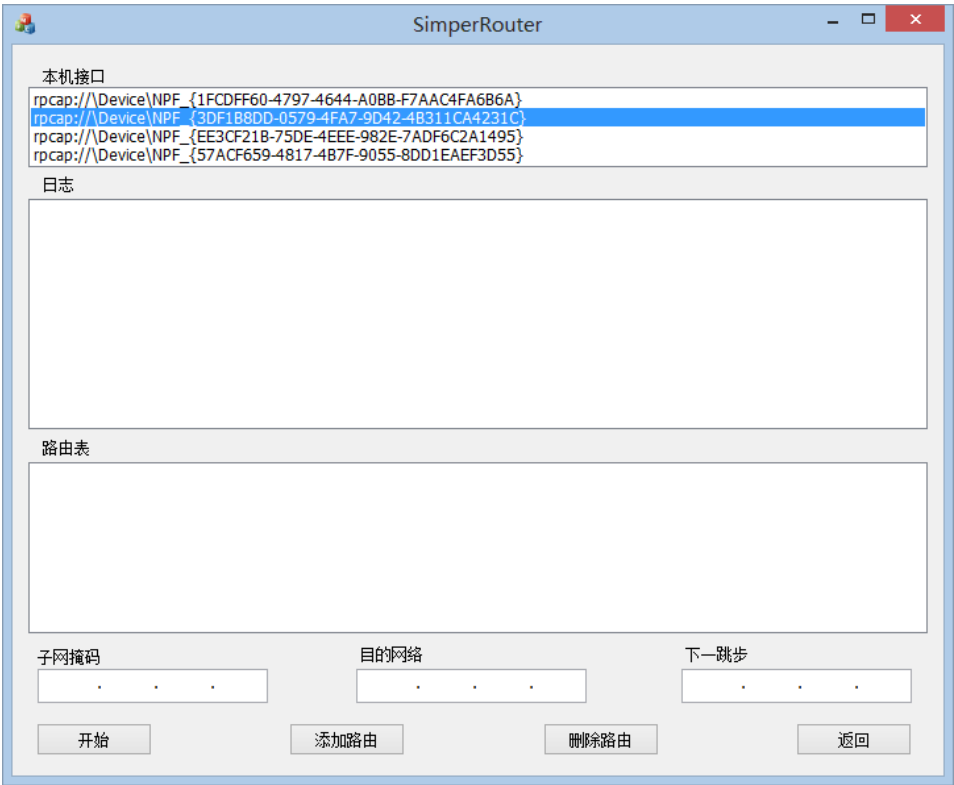
- 路由器在转发数据包时，如果生存周期 TTL 值减 1 后为 0，就丢弃这个数据包。当丢弃这个数据包时，路由器向源主机发出 ICMP 超时报文。
- 当计时器的时限到，而目的主机还没有接收到一个数据包的所有分片时，它就会将数据包的所有分片丢弃并向源主机发出 ICMP 超时报文。

1.5 实验编译运行环境

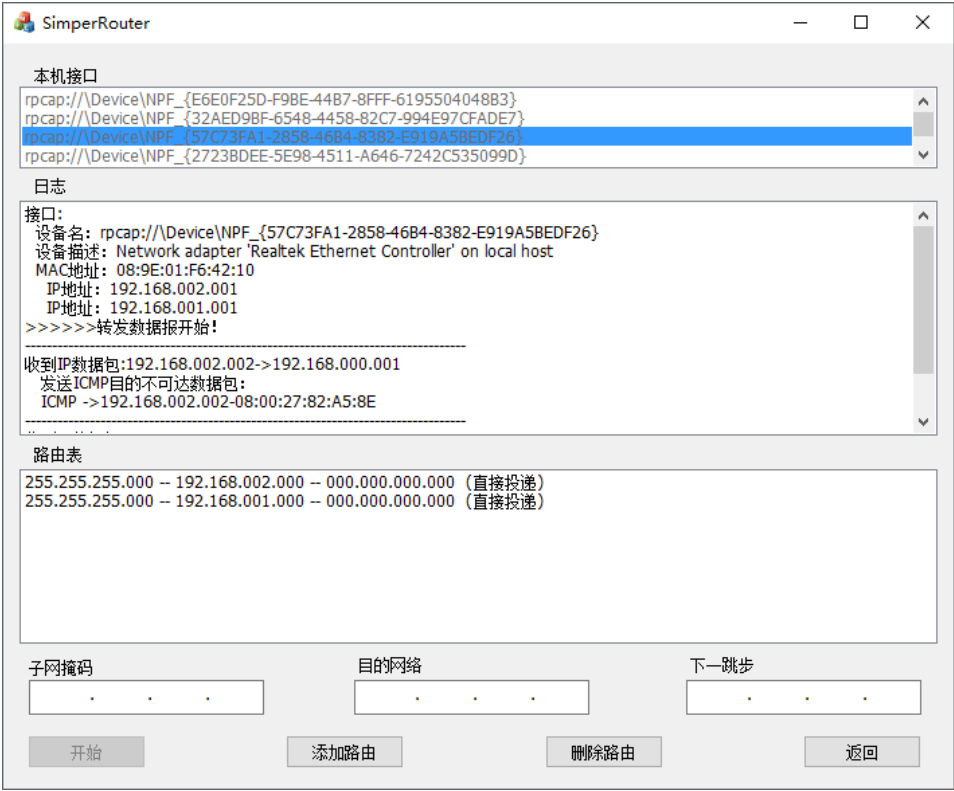
本程序编译环境是：Visual Studio 2012；系统环境是：Windows 8（64 位）；

二、“简单的路由器”程序的编写

2.1 程序界面



程序在转发数据包测试界面如下：



2.2 程序实现的功能

基本要求：

1) 静态路由表的维护：静态路由的添加、修改和删除等维护功能；自动获得与本机直接相连的路由信息

2) IP 数据报的处理：IP 数据报的接收、选路、发送（包括 ARP 解析）等工作；忽略分片处理、选项处理、动态路由等功能。

3) 日志：显示本机的网络接口、IP 数据报的接收情况、IP 数据报的选路情况、IP 数据报的发送情况。

扩展实现：

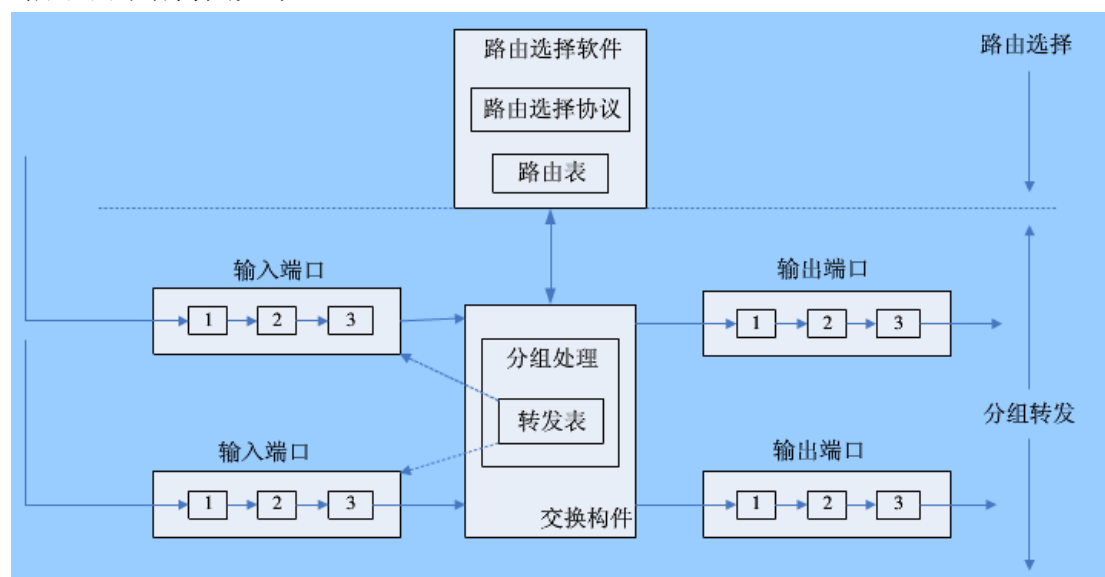
1) 实现了处理 IP 数据报的 TTL 值

2) 重新计算 IP 数据报的头部校验和

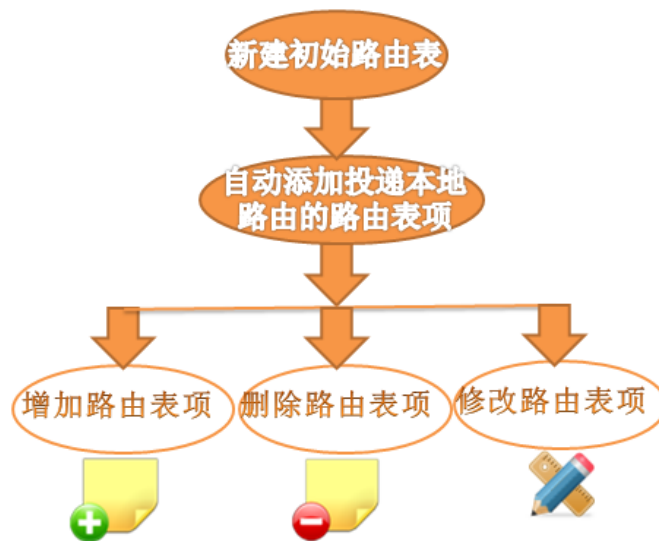
3) 生成和处理 ICMP 报文

2.3 路由程序的设计

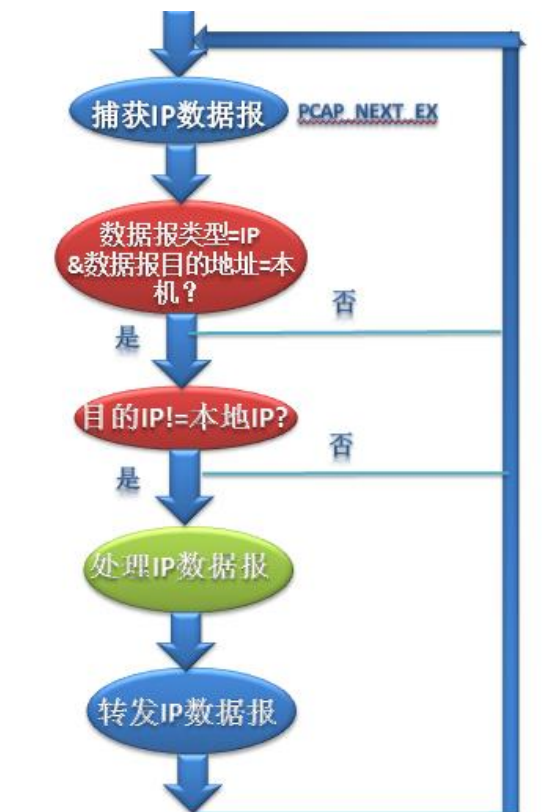
路由器的结构图如下：



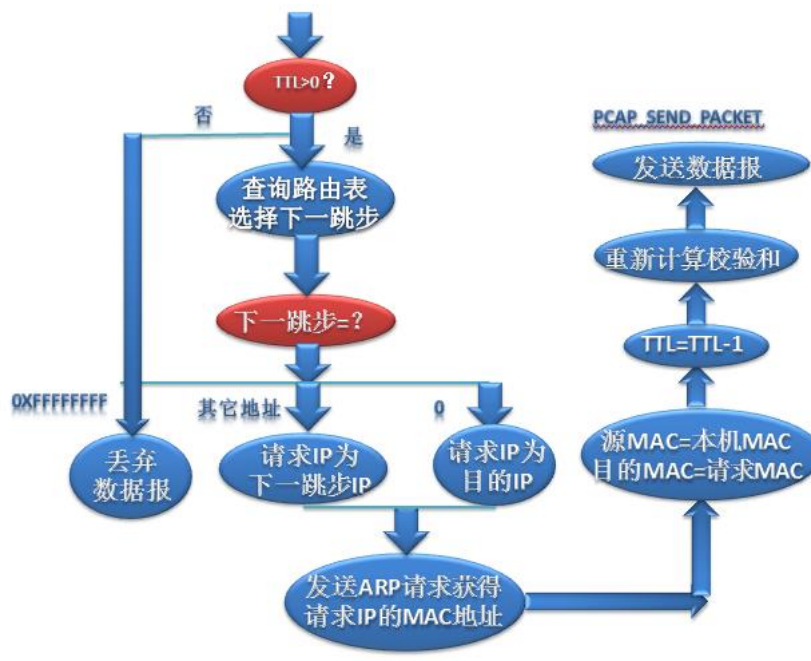
本程序着重处理 IP 数据报的接收、匹配路由、重设数据报中的一些数据、转发数据报的工作。路由表的操作如下图所示：



程序设计流程如下图所示：



其中，处理 IP 数据报的过程如下图所示：



按照模块划分可以分成三部分：

1. 路由器初始化模块
2. 静态路由表维护模块
3. 数据包捕获与处理模块

下面分别介绍各部分的实现：

2.3.1 路由器初始化模块

初始化模块负责初始化设备，添加与路由器直接相连网络的路由表项，并启动相应的数据包捕获与处理模块；

1、定义路由表数据结构

```

//路由表
typedef struct RouteTable_t {    // 路由表结构
    ULONG Mask;                // 子网掩码
    ULONG DstIP;                // 目的地址
    ULONG NextHop;              // 下一跳步
    UINT IfNo;                  // 接口序号
} RouteTable_t;
  
```

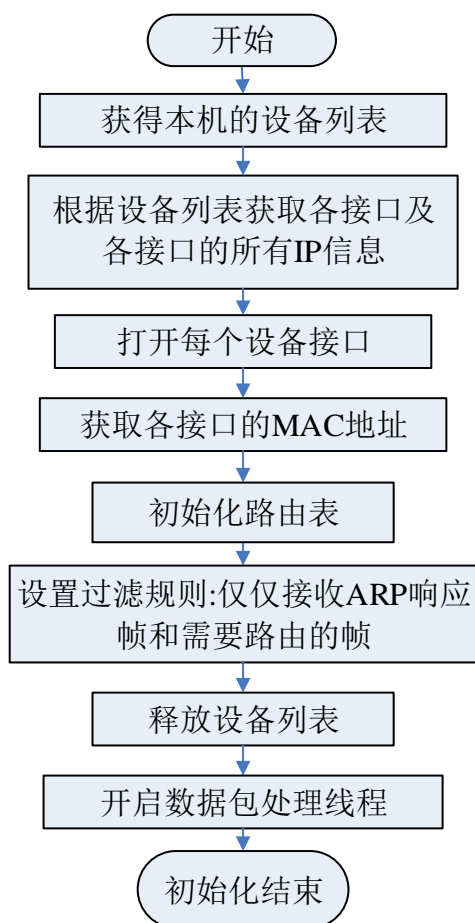
2、定义全局变量

```
IfInfo_t  IfInfo;           // 接口信息数组
UINT_PTR  TimerCount;       // 定时器个数

CList<SendPacket_t, SendPacket_t> SP;           // 发送数据包缓存队列
CList<IP_MAC_t, IP_MAC_t> IP_MAC;              // IP-MAC 地址映射列表
CList<RouteTable_t, RouteTable_t> RouteTable;   // 路由表
```

3、路由器的初始化

路由器的初始化过程图如下：



➤ 初始化-获得本机的设备列表

```
// 获得本机的设备列表
if(pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL /*无需认证*/, &alldevs, errbuf) == -1)
{
    // 错误, 返回错误信息
    sprintf(strbuf, "pcap_findalldevs_ex 错误: %s", errbuf);
    MessageBox(strbuf);
    PostMessage(WM_QUIT, 0, 0);
}
```

➤ 初始化-获得接口信息和 IP 地址信息

```
for(d = alldevs; d != NULL; d = d->next)
{
    if(d->addresses != NULL) // 排除集成 modem 的影响（没有 IP 地址）{
        // 得到一个有效的接口和其 IP 地址列表
        IfInfo[i].DeviceName = d->name;
        IfInfo[i].Description = d->description;
        for(a = d->addresses; a; a = a->next){
            if (a->addr->sa_family == AF_INET){
                ipaddr.IPAddr = (((struct sockaddr_in *)a->addr)->sin_addr.s_addr);
                ipaddr.IPMask = (((struct sockaddr_in *)a->netmask)-
                                >sin_addr.s_addr);

                IfInfo[i].ip.Add(ipaddr);
                j++;
            }
        }
        if (i==MAX_IF)// 最多处理 MAX_IF 个接口
        {break;    }
        else{i++;}
    }
}
```

➤ 初始化-打开设备接口

```
// 打开接口
for (i=0; i < IfCount; i++)
{
    if ( (IfInfo[i].adhandle = pcap_open(IfInfo[i].DeviceName, // 设备名
        65536, // 最大包长度
        PCAP_OPENFLAG_PROMISCUOUS, // 混杂模式
        1000, // 超时时间
        NULL, // 远程认证
        errbuf // 错误缓存
    )) == NULL)
    {
        // 错误，显示错误信息
        sprintf(strbuf, "接口未能打开。WinPcap 不支持%s。", IfInfo[i].DeviceName);
        MessageBox(strbuf);
        PostMessage(WM_QUIT, 0, 0);
    }
}
```

➤ 初始化-获得各接口 MAC 地址

```
// 开启数据包捕获线程，获取本地接口的 MAC 地址，线程数目为网卡个数
CWinThread* pthread;
for (i = 0; i < IfCount; i++)
{
    pthread = AfxBeginThread(CaptureLocalARP, &IfInfo[i], THREAD_PRIORITY_NORMAL);
    if(!pthread)
    {
        MessageBox("创建数据包捕获线程失败！");
        PostMessage(WM_QUIT, 0, 0);
    }
}
// 为得到真实网卡地址，使用虚假的 MAC 地址和 IP 地址向本机发送 ARP 请求
setMAC(srcMAC, 66); // 设置虚假的 MAC 地址
srcIP = inet_addr("112.112.112.112"); // 设置虚假的 IP 地址
for (i = 0; i < IfCount; i++)
{
    ARPRequest(IfInfo[i].adhandle, srcMAC, srcIP, IfInfo[i].ip[0].IPAddr);
}
```

➤ 初始化-初始化路由表

```
// 初始化路由表并显示
RouteTable_t rt;
for (i = 0; i < IfCount; i++)
{
    for (j = 0; j < IfInfo[i].ip.GetSize(); j++)
    {
        rt.IfNo = i;
        rt.DstIP = IfInfo[i].ip[j].IPAddr & IfInfo[i].ip[j].IPMask;
        rt.Mask = IfInfo[i].ip[j].IPMask;
        rt.NextHop = 0; // 直接投递
        RouteTable.AddTail(rt);
        m_RouteTable.InsertString(-1, IPntoa(rt.Mask) + " -- " + IPntoa(rt.DstIP) + " -- " +
IPntoa(rt.NextHop) + " (直接投递)");
    }
}
```

➤ 初始化-设置过滤规则开始捕获

```
// 设置过滤规则:仅仅接收 arp 响应帧和需要路由的帧
CString Filter, Filter0, Filter1;    只接收需要转发的 IP 数据报, 共同特点是目的 Filter0 =
“(” ;                               MAC 地址指向本机, 但目的 IP 地址不属于本
机
Filter1 = “(” ;                     转发需要 MAC 地址, 所以还需接收
ARP 应答包
for (i = 0; i < IfCount; i++)
{
    Filter0 += "(ether dst " + MACntoa(IfInfo[i].MACAddr) + ")";
    for (j = 0; j < IfInfo[i].ip.GetSize(); j++)    //mac 地址为本路由器接口
    {
        Filter1 += "(ip dst host " + IPntoa(IfInfo[i].ip[j].IPAddr) + ")";
        if (((j == (IfInfo[i].ip.GetSize() - 1))) && (i == (IfCount - 1)))
        {Filter1 += “)” ;}           //IP 地址为本机的接口地址
        else {Filter1 += "or ";}
    }
    if (i == (IfCount - 1))
    {Filter0 += ");"}
    else {Filter0 += "or ";}
} //需要转发的数据包
Filter = Filter0 + " and ((arp and (ether[21]=0x2)) or (not " + Filter1 + "))";
```

2.3.2 静态路由表维护模块

静态路由表维护模块完成路由表的添加、删除以及显示；

1、添加路由表项

```
void CRouterDlg::OnInsert()
{flag = false;
for (i=0; i < IfCount; i++)
{    for (j = 0; j < IfInfo[i].ip.GetSize(); j++)    {
        if (((IfInfo[i].ip[j].IPAddr) & (IfInfo[i].ip[j].IPMask)) == ((IfInfo[i].ip[j].IPMask) & ipaddr))
//判断该路由表项属于哪个接口
        {
            rt.IfNo = i;// 记录子网掩码
            m_Mask.GetAddress(ipaddr);                                rt.Mask = htonl(ipaddr); //
记录目的 IP
            m_Destination.GetAddress(ipaddr);
            rt.DstIP = htonl(ipaddr); // 记录下一跳
            m_NextHop.GetAddress(ipaddr);
            rt.NextHop = htonl(ipaddr);// 把该条路由表项添加到路由表
            RouteTable.AddTail(rt); // 在路由表窗口中显示该路由表项
            m_RouteTable.InsertString(-1, IPntoa(rt.Mask) + "--"
            + IPntoa(rt.DstIP) + "--" + IPntoa(rt.NextHop));
            flag = true;
        }
    }
}
```

2、删除路由表项

```
// 删除路由表项
void CRouterDlg::OnDel()
{
    RouteTable_t rt;
    POSITION pos, CurrentPos;
    m_RouteTable.GetText(i, str);
    // 取得子网掩码选项
    strncat(ipaddr, str, 15);
    mask = inet_addr(ipaddr);
    // 取得目的地址选项
    ipaddr[0] = 0;
    strncat(ipaddr, &str[19], 15);
    destination = inet_addr(ipaddr);
    // 取得下一跳选项
    ipaddr[0] = 0;
    strncat(ipaddr, &str[38], 15);
    nexthop = inet_addr(ipaddr);
}
```

```

if (nexthop == 0)
{
    MessageBox("直接连接路由，不允许删除！");
}
// 把该路由表项从路由表窗口中删除
m_RouteTable.DeleteString(i);
// 路由表中没有需要处理的内容，则返回
if (RouteTable.IsEmpty())
{
    return;
}
// 遍历路由表,把需要删除的路由表项从路由表中删除
pos = RouteTable.GetHeadPosition();
for (i=0; i<RouteTable.GetCount(); i++)
{
    CurrentPos = pos;
    rt = RouteTable.GetNext(pos);
    if ((rt.Mask == mask) && (rt.DstIP == destination) && (rt.NextHop == nexthop))
    {
        RouteTable.RemoveAt(CurrentPos);
    }
}
}

```

2.3.3 数据包捕获与处理模块

数据包捕获与处理模块用于捕获流经本路由器的数据包并按照路由协议进行处理；

1、数据包的捕获

```

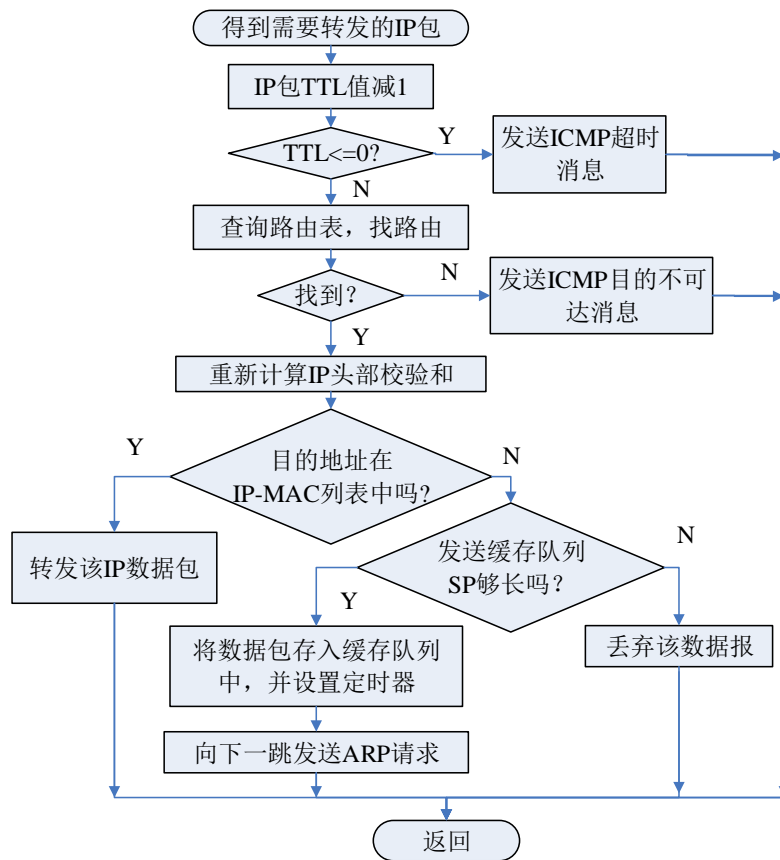
switch (ntohs(fh->FrameType))
{
    case 0x0806:
        ARPFrame_t *ARPf;
        ARPf = (ARPFrame_t *)pkt_data;
        //TRACE1("收到 ARP 包 源 IP 为: %d\n", ARPf->SendIP);

        // ARP 包，转到 ARP 包处理函数
        ARPPacketProc(header, pkt_data);
        break;
    case 0x0800:
        IPFrame_t *IPf;
        IPf = (IPFrame_t *)pkt_data;
        //TRACE1("收到 IP 包 源 IP 为: %d\n", IPf->IPHeader.SrcIP);
        // IP 包，转到 IP 包处理函数
        IPPacketProc(pIfInfo, header, pkt_data);
        break;
    default:
}

```

2、IP 数据包的处理

IP 数据包处理流程图如下：



```
void IPPacketProc(IfInfo_t *pIfInfo, struct pcap_pkthdr *header, const u_char *pkt_data)
{
    IPFrame_t      *IPf;
    SendPacket_t    sPacket;
    IPf = (IPFrame_t *) pkt_data;
    // ICMP 超时 IPf->IPHeader.TTL --; //TTL 值减去 1
    if (IPf->IPHeader.TTL <= 0)
    {
        ICMPPacketProc(pIfInfo, 11, 0, pkt_data);}
        IPHeader_t *IpHeader = &(IPf->IPHeader);
    // ICMP 差错
    if (IsChecksumRight((char *)IpHeader) == 0)
    {
        // 日志输出信息
        pDlg->m_Log.InsertString(-1, "    IP 数据包包头校验和错误, 丢弃数据包");
    }
}
```



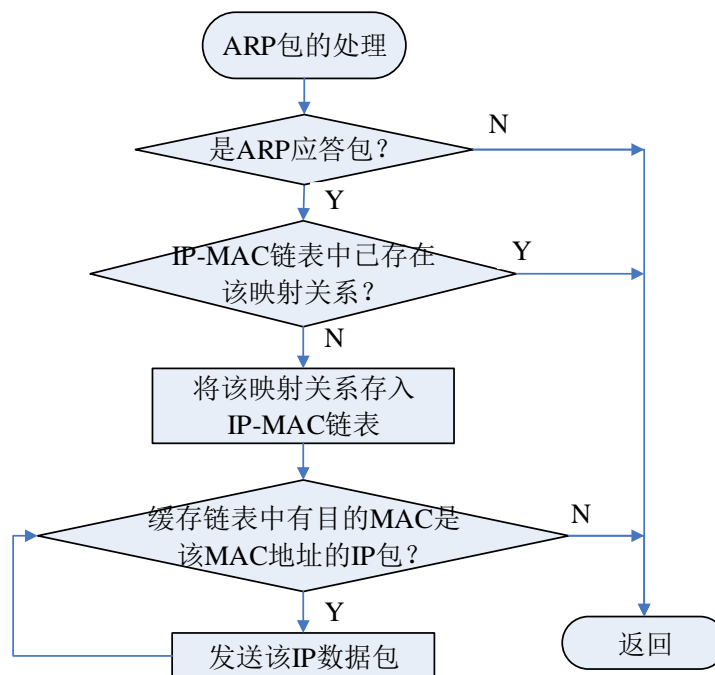
```

DWORD nextHop;           // 经过路由选择算法得到的下一站目的 IP 地址
UINT ifNo;               // 下一跳的接口序号
// 路由查询
if((nextHop = RouteLookup(ifNo, IPf->IPHeader.DstIP, &RouteTable)) == -1)
{
    // ICMP 目的不可达
    ICMPPacketProc(pIfInfo, 3, 0, pkt_data);
}
else //查找到了下一跳
{
    sPacket.IfNo = ifNo;
    sPacket.TargetIP = nextHop;
    copyMAC(IPf->FrameHeader.SrcMAC, IfInfo[sPacket.IfNo].MACAddr);
    // 设 IP 头中的校验和为 0
    IPf->IPHeader.Checksum = 0;
    // 计算 IP 头部校验和
    IPf->IPHeader.Checksum = ChecksumCompute(check_buff, sizeof(IPHeader_t));
    // IP-MAC 地址映射表中存在该映射关系
    if (IPLookup(sPacket.TargetIP, IPf->FrameHeader.DesMAC))
    {
        memcpy(sPacket.PktData, pkt_data, header->len);
        sPacket.len = header->len;
        if(pcap_sendpacket(IfInfo[sPacket.IfNo].adhandle, (u_char *) sPacket.PktData,
sPacket.len) != 0)
    }
    // IP-MAC 地址映射表中不存在该映射关系
    else
    {
        if (SP.GetCount() < 65530) // 存入缓存队列
        {
            sPacket.len = header->len;
            // 将需要转发的数据报存入缓存区
            memcpy(sPacket.PktData, pkt_data, header->len);
            SP.AddTail(sPacket);
        }
        // 发送 ARP 请求
        ARPRequest(IfInfo[sPacket.IfNo].adhandle, IfInfo[sPacket.IfNo].MACAddr,
IfInfo[sPacket.IfNo].ip[1].IPAddr, sPacket.TargetIP);
    }
}

```

3、ARP 数据包的处理

arp 数据包的处理流程图如下：



```
void ARPPacketProc(struct pcap_pkthdr *header, const u_char *pkt_data)
{
    ARPf = (ARPFrame_t *)pkt_data;
    if (ARPf->Operation == ntohs(0x0002))
    {
        if (IPLookup(ARPf->SendIP, macAddr))
        {
            pDlg->m_Log.InsertString(-1, "    该对应关系已经存在于 IP-MAC 地址映射表中");
            return;
        }
        else
        {
            ip_mac.IPAddr = ARPf->SendIP;
            memcpy(ip_mac.MACAddr, ARPf->SendHa, 6);
            // 将 IP-MAC 映射关系存入表中
            IP_MAC.AddHead(ip_mac);
        }
    }
}
```

4、发送 IP 数据包

```
do{
    // 查看是否能转发缓存中的 IP 数据报
    flag = false;
    // 没有需要处理的内容
    if (SP.IsEmpty())
    {    break;    }
    // 遍历转发缓存区
    pos = SP.GetHeadPosition();
    for (int i=0; i < SP.GetCount(); i++)
    {
        CurrentPos = pos;
        sPacket = SP.GetNext(pos);
        if (sPacket.TargetIP == ARPf->SendIP)
        {
            IPf = (IPFrame_t *) sPacket.PktData;
            cpyMAC(IPf->FrameHeader.DesMAC, ARPf->SendHa);
            pcap_sendpacket(IfInfo[sPacket.IfNo].adhandle, (u_char *) sPacket.PktData,
sPacket.len);
        }
    }
}
```

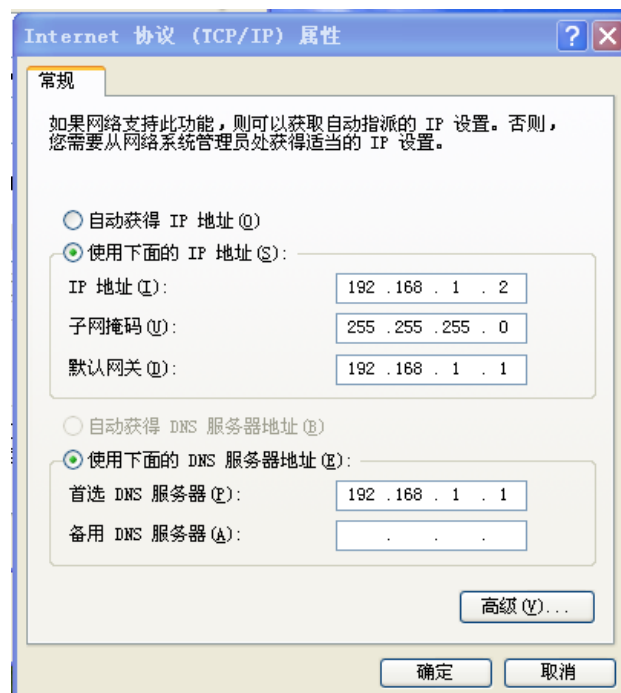
三、 实验结果

1、 程序检测环境的构建

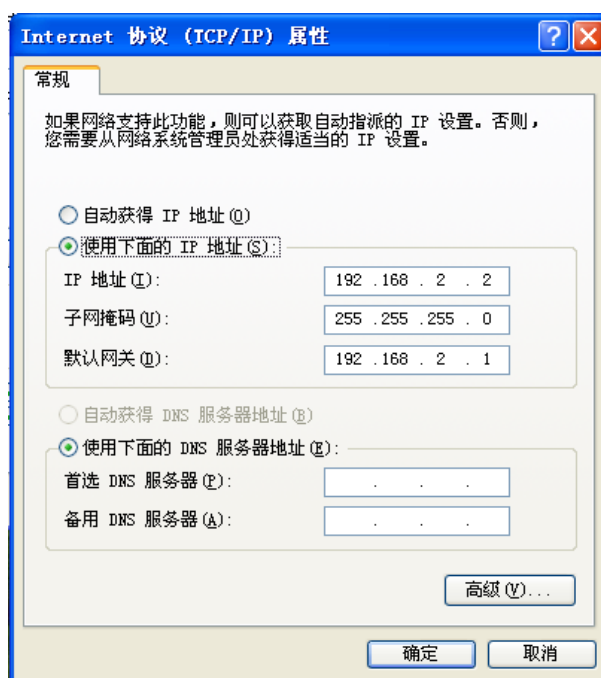
首先在一台计算机（路由器）上配置双 IP 分别是 192. 168. 1. 1 和 192. 168. 2. 1 如下：



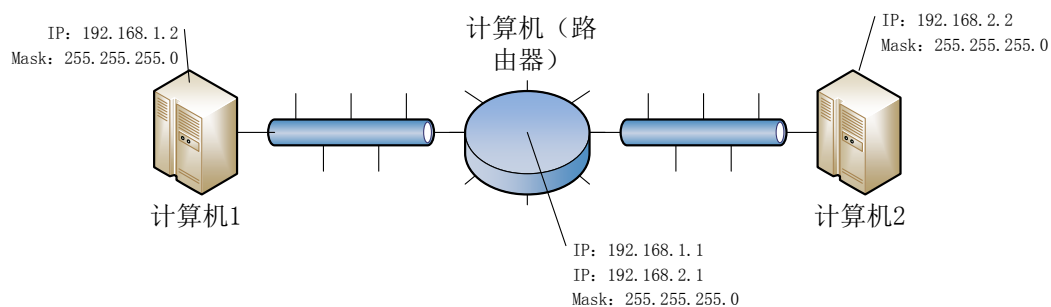
然后分别配置计算机 1 的 IP 是 192. 168. 1. 2，网关是 192. 168. 1. 1 如下：



配置计算机 2 的 IP 是 192.168.2.2，网关是 192.168.2.1 如下：



构造的网络图如下：



2、程序的正确性检测

首先在未开启路由器的情况下，计算机 1 ping 计算机 2 ping 不同：

```
命令提示符
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\7s>ping 192.168.2.2

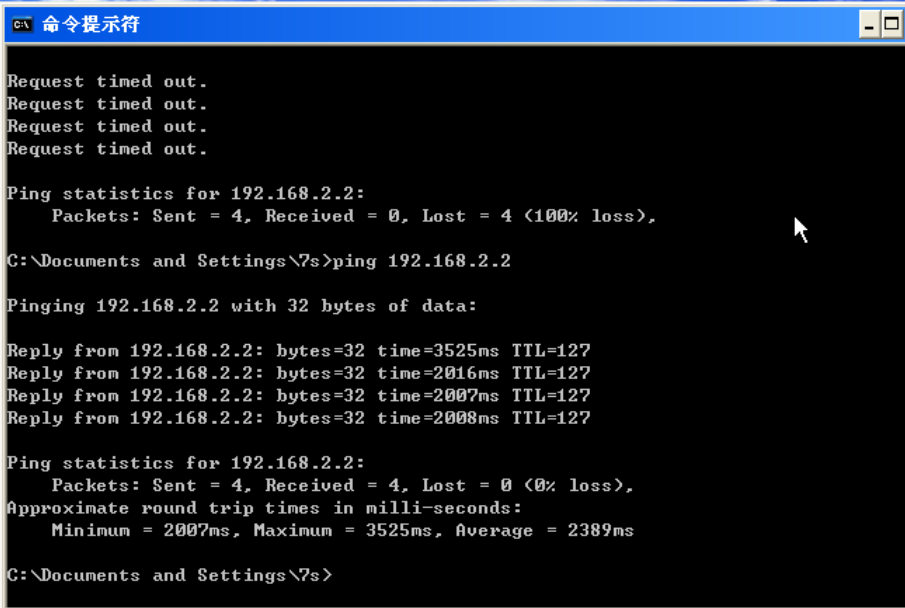
Pinging 192.168.2.2 with 32 bytes of data:

Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.168.2.2:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),

C:\Documents and Settings\7s>
```

开启路由器的情况下，计算机 1 可以 ping 同计算机 2，即路由程序正确的转发了数据包：



```
C:\>ping 192.168.2.2

Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.168.2.2:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),

C:\Documents and Settings\7s>ping 192.168.2.2

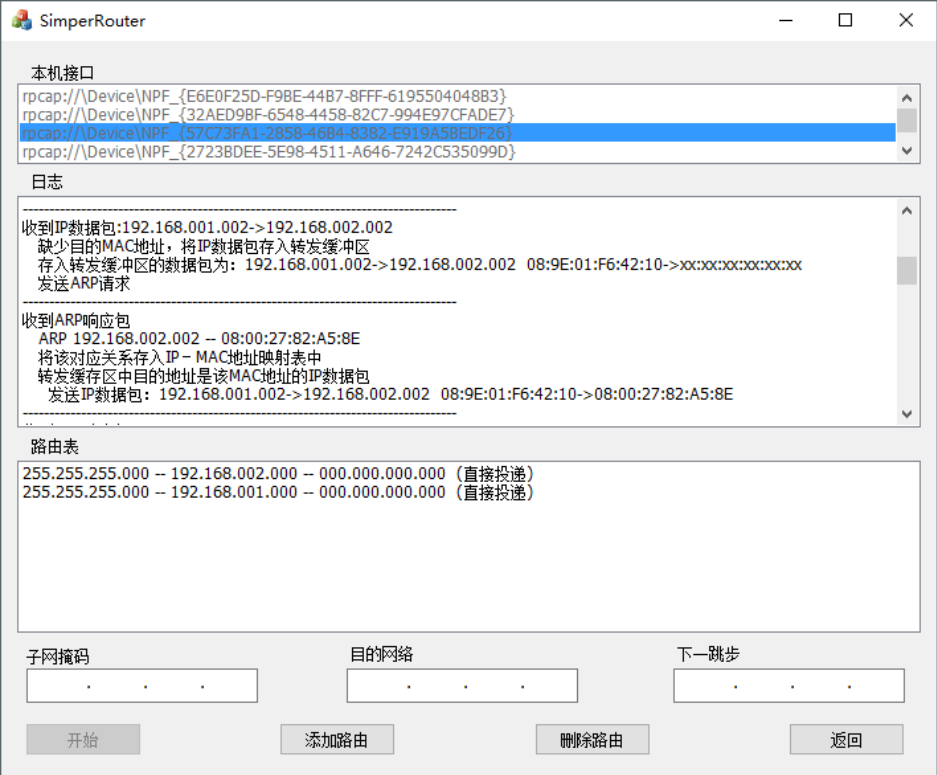
Pinging 192.168.2.2 with 32 bytes of data:

Reply from 192.168.2.2: bytes=32 time=3525ms TTL=127
Reply from 192.168.2.2: bytes=32 time=2016ms TTL=127
Reply from 192.168.2.2: bytes=32 time=2007ms TTL=127
Reply from 192.168.2.2: bytes=32 time=2008ms TTL=127

Ping statistics for 192.168.2.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 2007ms, Maximum = 3525ms, Average = 2389ms

C:\Documents and Settings\7s>
```

观察路由程序的日志可以发现，程序先发送了 arp 包：



本机接口

- rpcap://Device\NPF_{E6E0F25D-F9BE-44B7-8FFF-619504048B3}
- rpcap://Device\NPF_{32AED98F-6548-4458-82C7-994E97CFADE7}
- rpcap://Device\NPF_{57C73FA1-2858-4684-8382-E919A58EDF26}
- rpcap://Device\NPF_{2723BDEE-5E98-4511-A646-7242C535099D}

日志

收到IP数据包:192.168.001.002->192.168.002.002
缺少目的MAC地址，将IP数据包存入转发缓冲区
存入转发缓冲区的数据包为：192.168.001.002->192.168.002.002 08:9E:01:F6:42:10->xx:xx:xx:xx:xx:xx
发送ARP请求

收到ARP响应包
ARP 192.168.002.002 -- 08:00:27:82:A5:8E
将该对应关系存入IP - MAC地址映射表中
转发缓冲区中目的地址是该MAC地址的IP数据包
发送IP数据包：192.168.001.002->192.168.002.002 08:9E:01:F6:42:10->08:00:27:82:A5:8E

路由表

- 255.255.255.000 -- 192.168.002.000 -- 000.000.000.000 (直接投递)
- 255.255.255.000 -- 192.168.001.000 -- 000.000.000.000 (直接投递)

子网掩码: . . . 目的网络: . . . 下一跳步: . . .

开始 添加路由 删除路由 返回

最后成功转发了数据报：

SimperRouter

本机接口

rpcap://\\Device\\NPF_{E6E0F25D-F9BE-44B7-8FFF-6195504048B3}
rpcap://\\Device\\NPF_{32AED9BF-6548-4458-82C7-994E97CFADE7}
rpcap://\\Device\\NPF_{57C73FA1-2858-4684-8382-E919A58EDF26}
rpcap://\\Device\\NPF_{2723BDEE-5E98-4511-A646-7242C535099D}

日志

收到IP数据包:192.168.001.002->192.168.002.002
转发IP数据包:
192.168.001.002->192.168.002.002 08:9E:01:F6:42:10->08:00:27:82:A5:8E
收到IP数据包:192.168.002.002->192.168.001.002
转发IP数据包:
192.168.002.002->192.168.001.002 08:9E:01:F6:42:10->08:00:27:F4:02:9D
收到IP数据包:192.168.001.002->192.168.002.002
转发IP数据包:
192.168.001.002->192.168.002.002 08:9E:01:F6:42:10->08:00:27:82:A5:8E

路由表

255.255.255.000 -- 192.168.002.000 -- 000.000.000.000 (直接投递)
255.255.255.000 -- 192.168.001.000 -- 000.000.000.000 (直接投递)

子网掩码

目的网络

下一跳步

开始

添加路由

删除路由

返回

四、 实验总结与心得体会

1. 实验时遇到的困难

这次实验是在之前的捕获数据报程序和 arp 的程序之上组建的路由程序，通过捕获数据报和 arp 获取 mac 地址实现数据报的转发；在编写程序的过程中主要遇到了以下困难：

- vc 中的编译优化中有数据对齐的选项，如开启可能影响数据包的结构，造成发送错误的数据包，或是识别错误，所以编写网络程序务必将其关闭。
- 转发数据包时不仅要改写目的 mac 地址，也要改写源 mac 地址，否则无法使数据包进入另一子网。
- 在配置路由程序的检测环境时，由于一开始没有把计算机的防火墙关掉，导致一直没有 ping 通，反复修改仔细检查程序确保没有错误之后才想到会不会是防火墙的原因，结果真的是这个原因!!!

2. 实验的心得体会

通过这次“简单的路由程序设计”实验使我对于 MAC 地址、IP 地址和网络传输协议、数据报的转发、路由表的构建等等有了更深的理解，不但在查找资料的过程中学习了路由转发的机制、各种协议的作用，而且在处理细节的过程中得到了一次又一次的提高，对于如此庞大的网络世界有了更深的认识。

经过一周，由于又临近期末时间紧张，好几次的通宵查阅各种资料使我终于完成了这次的实验，当看到自己编写的程序能够成功 ping 通另一台主机时，激动之情实在是难以表达，随着这个程序的完成，《网络技术与应用》课程也快要结束了，这个课程实在是教会了我很多很多，因为只有当自己亲自编程实现书本中的各种协议、各种算法才知道这些是多么的巧妙，而在实际编写程序的过程中遇到的诸如“关闭防火墙才能使用 ping 命令”，“网络序与实际数据序不一样”等等问题是在单纯的理论学习中是无论如何也学不来的！这些不仅仅加深了对理论知识的理解，更提高了我对于探求网络技术的兴致。

最后感谢老师和可爱的助教们不厌其烦的为我们解答疑惑，也感谢我的室友们耽搁自己的事情一遍又一遍的提供电脑帮助我搭建局域网调试程序，谢谢！