**实验5：简单路由器程序的设计**

|  |  |
| --- | --- |
| 学院：网络空间安全学院  专业：密码科学与技术  姓名：梁婧涵  学号： 2112155 | |
| **实验要求**  简单路由器程序设计实验的具体要求为： |  |
| （1）设计和实现一个路由器程序，要求完成的路由器程序能和现有的路由器产品（如思科路由器、华为 路由器、微软的路由器等）进行协同工作。  （2）程序可以仅实现IP数据报的获取、选路、投递等路由器要求的基本功能。可以忽略分片处理、选项  处理、动态路由表生成等功能。  （3）需要给出路由表的手工插入、删除方法。  （4）需要给出路由器的工作日志，显示数据报获取和转发过程。  （5）完成的程序须通过现场测试，并在班（或小组）中展示和报告自己的设计思路、开发和实现过程、 测试方法和过程。 | |

**实验步骤**

**1、设计思路**

简单路由处理软件可以仅接收需要转发的IP数据报，目的MAC地址指向本机但目的IP地址不属于本机的 IP地址。利用pcap包过滤机制筛选出需要处理的IP数据报提交给简单路由处理程序。

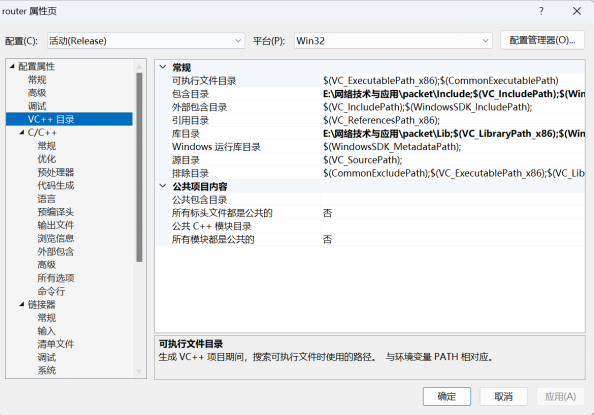
根据IP路由选择算法，在利用pcap获取到需要转发的IP数据报后，路由处理软件首先需要提取该报文的 目的IP地址；如果不成功，则将报文丢弃。

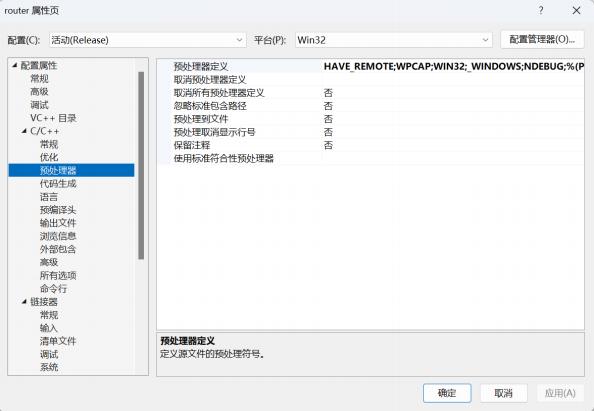
在将路由选择成功发IP数据报发送到相应的接口前，应利用ARP获取下一个路由器接口的MAC地址。 一 旦得到向下一个路由器的MAC地址，就把IP数据报封装成数据帧并通过相应的接口发送出去。

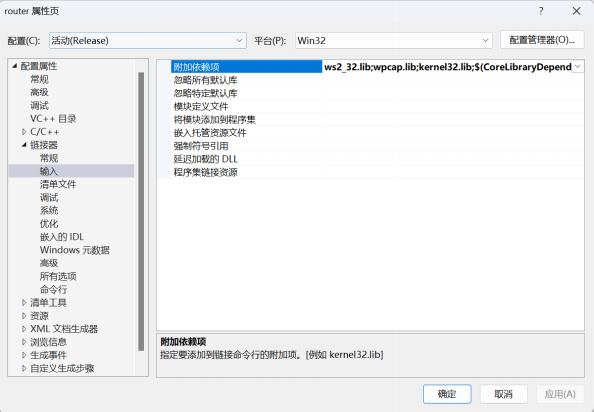
**2、具体流程**

VS2022创建基于对话框的MFC程序

配置文件属性：





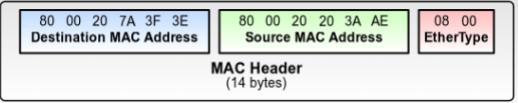


**3、相关数据结构定义**

网络中传输数据报要经过封装，每封装一次都增加首部。 pcap在数据链路层捕获数据包，在以太网中传 送数据都包含以太网帧头信息。由于捕获数据包要保存在一个无结构的缓冲区中，在实际实验中要定义 一些首部的数据结构。通过这些结构将数据包存放在无结构缓冲区，简化数据提取。

**注意：** pcap捕获数据包在缓冲区连续存放，通常VC++默认IDE的设置不是字节对齐。需要使用#pragma pack(1)进入字节对齐方式， #pragma pack恢复默认对齐方式。

1. 帧头部



//帧头部结构体 共14字节

typedef struct FrameHeader\_t {

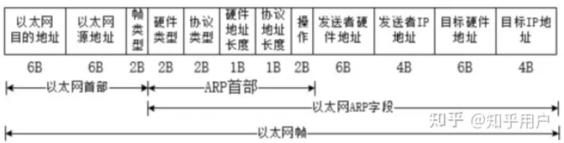
UCHAR DesMAC[6]; // 目的MAC地址 6字节

UCHAR SrcMAC[6]; // 源MAC地址 6字节

USHORT FrameType; // 上一层协议类型 2字节

} FrameHeader\_t;

2. ARP帧



|  |
| --- |
| typedef struct ARPFrame\_t { // ARP帧  FrameHeader\_t FrameHeader; // 帧首部  WORD HardwareType; // 硬件类型  WORD ProtocolType; // 协议类型  BYTE HLen; // 硬件地址长度  BYTE PLen; // 协议地址长度  WORD Operation; // 操作值  UCHAR SendHa[6]; // 源MAC地址  ULONG SendIP; // 源IP地址  UCHAR RecvHa[6]; // 目的MAC地址  ULONG RecvIP; // 目的IP地址  } ARPFrame\_t; |

3. IP首部



typedef struct IPHeader\_t { // IP首部

BYTE Ver\_HLen; // 版本+头部长度

BYTE TOS; // 服务类型

WORD TotalLen; // 总长度

WORD ID; // 标识

WORD Flag\_Segment; // 标志+片偏移

BYTE TTL; // 生存时间

BYTE Protocol; // 协议

WORD Checksum; // 头部校验和

ULONG SrcIP; // 源IP地址

ULONG DstIP; // 目的IP地址

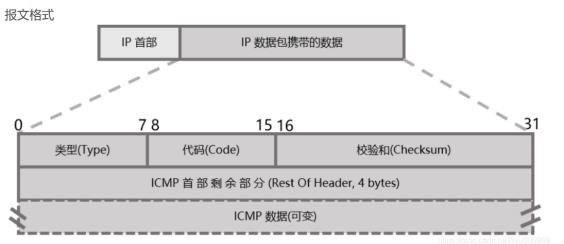
} IPHeader\_t;

4. 包含帧首部和IP首部的数据包

将 FrameHeader\_t 和 IPHeader\_t 结合

|  |
| --- |
| typedef struct IPFrame\_t { // IP帧  FrameHeader\_t FrameHeader; // 帧首部  IPHeader\_t IPHeader; // IP首部  } IPFrame\_t; |

5. ICMP首部



|  |
| --- |
| typedef struct ICMPHeader\_t { // ICMP首部  BYTE Type; // 类型  BYTE Code; // 代码  WORD Checksum; // 校验和  WORD Id; // 标识  WORD Sequence; // 序列号  } ICMPHeader\_t; |

6. 网络地址

IP地址和子网掩码相与

|  |
| --- |
| typedef struct ip\_t { // 网络地址  ULONG IPAddr; // IP地址  ULONG IPMask; // 子网掩码  } ip\_t; |

7. 接口地址

需要用列表保存一块网卡的IP信息

|  |
| --- |
| typedef struct IfInfo\_t { // 接口信息  char\* DeviceName; // 设备名  CString Description; // 设备描述  UCHAR MACAddr[6]; // MAC地址  CArray <ip\_t,ip\_t&> ip; // IP地址列表  pcap\_t \*adhandle; // pcap句柄  } IfInfo\_t; |

8. 发送数据包结构

包括长度、数据缓存、目的IP地址、定时器、接口序号

|  |
| --- |
| typedef struct SendPacket\_t { // 发送数据包结构  int len; // 长度  BYTE PktData[2000]; // 数据缓存  ULONG TargetIP; // 目的IP地址  UINT\_PTR n\_mTimer; // 定时器  UINT IfNo; // 接口序号  } SendPacket\_t; |

9. 路由表项结构

包括子网掩码、目的地址、下一跳步、接口序号

|  |
| --- |
| typedef struct RouteTable\_t { // 路由表项结构  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; |

10. IP-MAC地址映射结构

保存IP地址和MAC地址的映射关系。在数据包转发时，通过IP地址获得MAC地址，修改数据包首部 之后再转发

|  |
| --- |
| typedef struct IP\_MAC\_t { // IP-MAC地址映射结构  ULONG IPAddr; // IP地址  UCHAR MACAddr[6]; // MAC地址  } IP\_MAC\_t; |



**重要代码解析**

**1、全局变量**

包括接口信息数组、接口个数、定时器个数、 CList类型的发送数据包缓存队列、 IP-MAC地址映射列表以 及路由表。发送数据包缓存队列保存因没有获得MAC地址而暂时发送不出去的数据包。 IP-MAC地址映射 列表保存IP和MAC地址的映射信息，路由器得到MAC地址并转发。路由表是一个路由表项的列表。为了 保证多个线程对统一资源的互斥访问，申请一个互斥锁。

|  |
| --- |
| // -----------------全局变量------------------  IfInfo\_t IfInfo[MAX\_IF]; // 接口信息数组  int IfCount; // 接口个数  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  CMutex mMutex(0,0,0); //互斥 |

**2、获取本机设备列表**

在初始化对话框函数 OnInitDialog() 中通过调用 pcap\_findalldevs\_ex() 函数获取本机的设备列

表，如果获取失败，弹出对话框打印失败信息。将设备列表显示在 listbox 控件上。

|  |
| --- |
| // 获得本机的设备列表  if (pcap\_findalldevs\_ex(PCAP\_SRC\_IF\_STRING, NULL /\*无需认证\*/, &m\_alldevs, errbuf) == -1)  {  // 错误，返回错误信息  sprintf\_s(strbuf, "pcap\_findalldevs\_ex错误: %s", errbuf);  //MessageBox(strbuf);  PostMessage(WM\_QUIT, 0, 0);  }  for (m\_selectdevs = m\_alldevs; m\_selectdevs != NULL; m\_selectdevs = m\_selectdevs->next) // 显示接口列表  m\_dev.AddString(CString(m\_selectdevs->name)); // 利用d->name获取该网络接口设 备的名字  m\_dev.SetCurSel(0);  m\_dev.SetHorizontalExtent(1600);  m\_log.SetHorizontalExtent(1600); |

**3、选择并打开设备**

获取 listbox 被选中的行数目，遍历获取到被选中的接口，通过接口的 addresses 信息分别获取到 IP 地

址和掩码地址，进行类型转换，转换为 struct sockaddr\_in 类型的指针名，然后通过其

sin\_addr.s\_addr 属性获取地址。同时，网卡上可能绑定了不止一个 IP ，用一个循环获取到这块网卡 上绑定的所有 IP 地址信息。在获取到 IP 地址信息后，检查它的 IP 地址个数，本次实验要求路由器必须 至少有两个 IP ，因此若不符合路由器 IP 地址数目要求则弹出对话框打印错误信息。用 pcap\_open 函 数打开设备，返回值为 NULL ，表示打开错误，输出错误信息，直接释放设备列表，返回；否则，设备 已正确打开，进行数据包的发送和捕获。

// 选择接口

int N = m\_dev.GetCurSel(); // 获取listbox被选中的行的数目

m\_selectdevs = m\_alldevs;

while (N--)

m\_selectdevs = m\_selectdevs->next;

IfInfo[0].DeviceName = m\_selectdevs->name;

IfInfo[0].Description = m\_selectdevs->description;

for (a = m\_selectdevs->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[0].ip.Add(ipaddr);

j++;

}

}

// 不符合路由器IP地址数目要求

if (j < 2)

{

MessageBox(L"该路由程序要求本地主机至少应具有2个IP地址");

GetDlgItem(IDC\_BUTTON\_START)->EnableWindow(TRUE);

m\_dev.EnableWindow(TRUE);

return;

}

// 保存实际的网卡数

IfCount = 1;//IfCount = i;

// 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\_s(strbuf, "接口未能打开", IfInfo[i].DeviceName);

// 释放设备列表

pcap\_freealldevs(m\_alldevs);

return;

}

}

**4、获取网卡MAC地址**

开启数据包捕获线程，将列表中网卡硬件地址清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，此处设置虚假的 MAC 地址为66.66.66.66.66.66，设置虚假的 IP 地址为 <112.112.112.112> ，发送 ARP 数据包请求对应的 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(L"创建数据包捕获线程失败！");

PostMessage(WM\_QUIT, 0, 0);

}

}

// 将列表中网卡硬件地址清0

for (i = 0; i < IfCount; i++)

{

setMAC(IfInfo[i].MACAddr, 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); }

通过 pcap\_next\_ex() 函数不停地捕获数据包，捕获的时候需要设置过滤条件为ARP 包、响应包且源 IP 地址为该网卡的 IP 地址（作为参数传入），当捕获到这样一个数据包就从中获得 MAC 地址并且给传 入的参数赋值

|  |
| --- |
| // 获取本地接口MAC地址线程  UINT CaptureLocalARP(PVOID pParam)  {  int res;  struct pcap\_pkthdr\* header;  const u\_char\* pkt\_data;  IfInfo\_t\* pIfInfo;  ARPFrame\_t\* ARPFrame;  CString DisplayStr;  pIfInfo = (IfInfo\_t\*)pParam;  while (true)  {  res = pcap\_next\_ex(pIfInfo->adhandle, &header, &pkt\_data);  // 超时  if (res == 0)  continue;  if (res > 0)  {  ARPFrame = (ARPFrame\_t\*)(pkt\_data);  // 得到本接口的MAC地址  if ((ARPFrame->FrameHeader.FrameType == htons(0x0806)) && (ARPFrame->Operation == htons(0x0002))  && (ARPFrame->SendIP == pIfInfo->ip[0].IPAddr))  {  cpyMAC(pIfInfo->MACAddr, ARPFrame->SendHa);  return 0;  }  }  }  } |

ARPRequest() 函数发送 ARP 请求，函数传入的参数有设备列表的句柄、源 MAC 地址、源 IP 地址以 及目的 IP 地址。构造 FrameHeader ，目的MAC地址设置为全广播地址，源 MAC 地址设置为

66:66:66:66:66:66，然后填充 ARP 帧，设置其硬件类型、保护协议等参数，其源 IP 和目的 IP 均为传进 来的参数，分别为<112.112.112.112>和本机网卡上的 IP 。然后通过 pcap\_sendpacket() 函数发送数据 包。

// 发送ARP请求

void ARPRequest(pcap\_t\* adhandle, UCHAR\* srcMAC, ULONG srcIP, ULONG targetIP)

{

ARPFrame\_t ARPFrame;

int i;

for (i = 0; i < 6; i++)

{

ARPFrame.FrameHeader.DesMAC[i] = 255;

ARPFrame.FrameHeader.SrcMAC[i] = srcMAC[i];

ARPFrame.SendHa[i] = srcMAC[i];

ARPFrame.RecvHa[i] = 0;

}

ARPFrame.FrameHeader.FrameType = htons(0x0806);

ARPFrame.HardwareType = htons(0x0001);

ARPFrame.ProtocolType = htons(0x0800);

ARPFrame.HLen = 6;

ARPFrame.PLen = 4;

ARPFrame.Operation = htons(0x0001);

ARPFrame.SendIP = srcIP;

ARPFrame.RecvIP = targetIP;

pcap\_sendpacket(adhandle, (u\_char\*)&ARPFrame, sizeof(ARPFrame\_t)); }

**5、初始化路由表**

于网卡上的每一个 IP 地址，目标网络为 IP 地址与掩码地址相与，下一跳地址为0，代表直接投递，子网 掩码即为网卡上的掩码地址，将这条路由表项加入到路由表中，并且在 listbox 控件中显示

// 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)");

}

}

**6、设置过滤规则**

该路由器只需要接收两种数据包， ARP 相应帧，它是路由器先发送 ARP 数据包请求与 IP 地址相对应的 MAC 地址，当该 IP 地址对应的主机收到这个请求包，会发送响应包，其中包括了自己的 MAC 地址，解 析这个 ARP 响应包即可获得想要的 MAC 地址；以及需要路由的帧，在两个子网中的主机若想要通信，

则必须经过路由器的转发。主机A发送数据包的 IP 地址为主机B的 IP 地址， MAC 地址为路由器的 MAC 地址，当路由器收到这样一个数据包之后，就查找主机B的 MAC 地址并转发数据包。设置过滤规则将上 述两种数据包过滤出来。 ARP 数据包且为响应包，或者目的 IP 不是自己网卡上任何一个 IP 的数据包。 然后通过 pcap\_compile() 函数把刚才设置的过滤字符串编译进过滤信息。

|  |
| --- |
| // 设置过滤规则:仅接收arp响应帧和需要路由的帧  CString Filter, Filter0, Filter1;  Filter1 = "(";  for (i = 0; i < IfCount; i++)  {  Filter0 += L"(ether dst " + MACntoa(IfInfo[i].MACAddr) + L")"; for (j = 0; j < IfInfo[i].ip.GetSize(); j++) {  Filter1 += L"(ip dst host " + IPntoa(IfInfo[i].ip[j].IPAddr) + L")"; if (((j == (IfInfo[i].ip.GetSize() - 1))) && (i == (IfCount - 1)))  Filter1 += ")";  else  Filter1 += " or ";  }  if (i != (IfCount - 1))  Filter0 += " or ";  }  Filter = Filter0 + L" and ((arp and (ether[21]=0x2)) or (not" + Filter1 + L"))";  for (int i = 0; i < Filter.GetLength(); i++)  {  strbuf[i] = char(Filter[i]);  }  strbuf[Filter.GetLength()] = '\0';  for (i = 0; i < IfCount; i++)  {  if ((pcap\_compile(IfInfo[i].adhandle, &fcode, strbuf, 1,  IfInfo[i].ip[0].IPMask) < 0) || (pcap\_setfilter(IfInfo[i].adhandle, &fcode) < 0))  {  MessageBox(Filter + L"过滤规则编译不成功，设置过滤器错误！"); PostMessage(WM\_QUIT, 0, 0);  }  } |

**7、释放设备列表**

|  |
| --- |
| pcap\_freealldevs(m\_alldevs); |

**8、开启捕获数据包线程**

// 开始捕获数据包

for (i = 0; i < IfCount; i++)

{

pthread = AfxBeginThread(Capture, &IfInfo[i], THREAD\_PRIORITY\_NORMAL); if (!pthread)

{

MessageBox(L"创建数据包捕获线程失败！");

PostMessage(WM\_QUIT, 0, 0);

}

}

m\_log.AddString(L">>>>>>转发数据报开始！");

}

在循环中调用 pcap\_next\_ex() 函数捕获数据包，分析捕获到的数据包，如果它的帧类型是0806，则 它是 ARP 数据包，转到 ARP 数据包处理函数；如果它的帧类型是0800，则它是 IP 数据包，转到 IP 数 据包处理函数

// 数据包捕获线程

UINT Capture(PVOID pParam)

{

int res;

IfInfo\_t\* pIfInfo;

struct pcap\_pkthdr\* header;

const u\_char\* pkt\_data;

pIfInfo = (IfInfo\_t\*)pParam;

// 开始正式接收并处理帧

while (true)

{

res = pcap\_next\_ex(pIfInfo->adhandle, &header, &pkt\_data);

if (res == 1)

{

FrameHeader\_t\* fh;

fh = (FrameHeader\_t\*)pkt\_data;

switch (ntohs(fh->FrameType))

{

case 0x0806:

ARPFrame\_t \* ARPf;

ARPf = (ARPFrame\_t\*)pkt\_data;

// 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;

// IP包，转到IP包处理函数

IPPacketProc(pIfInfo, header, pkt\_data);

break;

default:

break;

}

}

else if (res == 0) // 超时

{

continue;

}

else

{

AfxMessageBox(L"pcap\_next\_ex函数出错!");

}

}

return 0;

}

**9、ARP数据包处理**

当收到一个 ARP 数据包后，调用该 ARP 数据包处理函数，判断该 ARP 帧的操作数，如果是2，则其是 ARP 响应包，将收到该数据包的信息打印到日志里，查询 IP-MAC 地址映射表，若 IP-MAC 地址映射表 中已经存在这个对应关系，输出日志信息后直接返回，倘若没有这个对应关系，将其加入进去。由于新 获得了一个对应关系，需要检查缓冲区中有无可以发送的数据包，在对缓冲区进行操作之前，先加锁， 遍历转发缓冲区，若找到了数据包的目的 IP 地址为刚加进来的映射中的 IP 地址，则修改数据包的目的 MAC 地址为刚才获得的 MAC 地址，将其转发。当缓冲区里没有可以转发的数据包，解锁。

// 处理ARP数据包

void ARPPacketProc(struct pcap\_pkthdr\* header, const u\_char\* pkt\_data) {

bool flag;

ARPFrame\_t\* ARPf;

IPFrame\_t\* IPf;

SendPacket\_t sPacket;

POSITION pos, CurrentPos;

IP\_MAC\_t ip\_mac;

UCHAR macAddr[6];

ARPf = (ARPFrame\_t\*)pkt\_data;

if (ARPf->Operation == ntohs(0x0002))

{

//pDlg->m\_log.AddString(L"----------------------------------------------

------------------------------------");

pDlg->m\_log.InsertString(-1, L"收到ARP响应包");

pDlg->m\_log.InsertString(-1, (L" ARP " + (IPntoa(ARPf->SendIP)) + L" -

"

-

+ MACntoa(ARPf->SendHa)));

// IP－MAC地址映射表中已经存在该对应关系

if (IPLookup(ARPf->SendIP, macAddr))

{

pDlg->m\_log.InsertString(-1, L" 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);

// 日志输出信息

pDlg->m\_log.InsertString(-1, L" 将该对应关系存入IP－MAC地址映射表中");

}

mMutex.Lock(INFINITE);

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);

for (int t = 0; t < 6; t++)

{

IPf->FrameHeader.SrcMAC[t] =

IfInfo[sPacket.IfNo].MACAddr[t];

}

// 发送IP数据包

pcap\_sendpacket(IfInfo[sPacket.IfNo].adhandle, (u\_char\*)sPacket.PktData, sPacket.len);

SP.RemoveAt(CurrentPos);

// 日志输出信息

pDlg->m\_log.InsertString(-1, L" 转发缓存区中目的地址是该MAC地址

的IP数据包");

pDlg->m\_log.InsertString(-1, (L" 发送IP数据包：" +

IPntoa(IPf->IPHeader.SrcIP) + L"->"

+ IPntoa(IPf->IPHeader.DstIP) + " " + MACntoa(IPf-

>FrameHeader.SrcMAC)

+ L"->" + MACntoa(IPf->FrameHeader.DesMAC)));

flag = true;

break;

}

}

} while (flag);

mMutex.Unlock();

}

}

IPLookup() 函数查询 IP-MAC 映射表，通过给定的 IP 在 IP-MAC 映射表中查找对应的 MAC 地址

|  |
| --- |
| // 查询IP-MAC映射表  bool IPLookup(ULONG ipaddr, UCHAR\* p)  {  IP\_MAC\_t ip\_mac;  POSITION pos;  if (IP\_MAC.IsEmpty())  return false;  pos = IP\_MAC.GetHeadPosition();  for (int i = 0; i < IP\_MAC.GetCount(); i++)  {  ip\_mac = IP\_MAC.GetNext(pos);  if (ipaddr == ip\_mac.IPAddr)  {  for (int j = 0; j < 6; j++)  {  p[j] = ip\_mac.MACAddr[j];  }  return true;  }  }  return false;  } |

**10、 IP数据报处理**

收到一个 IP 数据包后，调用该 IP 数据包处理函数。检查是否超时（检查 IP 数据包首部的TTL） ，若超 时，则发送类型值为11的 ICMP 超时数据包，直接返回。进行校验和检验，如果检验结果不正确，在日 志里打印错误信息，直接返回。进行路由查询，调用 RouteLookup() 函数在路由表中查找下一跳地

址，如果未找到对应信息，则发送类型值为3的 ICMP 目的不可达数据包。若找到了下一跳地址，则需要 进行数据包的转发。修改数据包的源 MAC 地址，重新计算校验和。在 IP-MAC 地址映射表中查找目的

IP 对应的 MAC 地址，若找到，则转发该数据包，若没找到，将该数据包加入缓冲区，发送 ARP 请求 包，请求这个 IP 对应的 MAC 地址。在加入缓冲区前要加锁，同时开启一个定时器，若超时或者缓冲区 已满，则直接丢弃该数据包

// 处理IP数据包

void IPPacketProc(IfInfo\_t\* pIfInfo, struct pcap\_pkthdr\* header, const u\_char\* pkt\_data)

{

//pDlg->m\_log.AddString(L"---------------------------------------------------

-------------------------------");

IPFrame\_t\* IPf;

SendPacket\_t sPacket;

IPf = (IPFrame\_t\*)pkt\_data;

pDlg->m\_log.InsertString(-1, (L"收到IP数据包:" + IPntoa(IPf->IPHeader.SrcIP) + L"->"

+ IPntoa(IPf->IPHeader.DstIP)));

// ICMP超时

if (IPf->IPHeader.TTL <= 1)///////<>if (IPf->IPHeader.TTL <= 1) {

ICMPPacketProc(pIfInfo, 11, 0, pkt\_data);

return;

}

IPHeader\_t\* IpHeader = &(IPf->IPHeader);

// ICMP差错

if (IsChecksumRight((char\*)IpHeader) == 0)

{

// 日志输出信息

pDlg->m\_log.InsertString(-1, L" IP数据包包头校验和错误，丢弃数据包");

return;

}

// 路由查询

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);

return;

}

else

{

sPacket.IfNo = ifNo;

sPacket.TargetIP = nextHop;

cpyMAC(IPf->FrameHeader.SrcMAC, IfInfo[sPacket.IfNo].MACAddr);

// TTL减1

IPf->IPHeader.TTL -= 1;

unsigned short check\_buff[sizeof(IPHeader\_t)];

// 设IP头中的校验和为0

IPf->IPHeader.Checksum = 0;

memset(check\_buff, 0, sizeof(IPHeader\_t));

IPHeader\_t\* ip\_header = &(IPf->IPHeader);

memcpy(check\_buff, ip\_header, sizeof(IPHeader\_t));

// 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)

{

// 错误处理

AfxMessageBox(L"发送IP数据包时出错!");

return;

}

// 日志输出信息

pDlg->m\_log.InsertString(-1, L" 转发IP数据包：");

pDlg->m\_log.InsertString(-1, (L" " + IPntoa(IPf->IPHeader.SrcIP) +

L"->"

+ IPntoa(IPf->IPHeader.DstIP) + " " + MACntoa(IPf- >FrameHeader.SrcMAC)

+ L"->" + MACntoa(IPf->FrameHeader.DesMAC)));

}

// 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);

// 在某一时刻只允许一个线程维护链表

mMutex.Lock(INFINITE);

sPacket.n\_mTimer = TimerCount;

if (TimerCount++ > 65533)

{

TimerCount = 1;

}

pDlg->SetTimer(sPacket.n\_mTimer, 10000, NULL);

SP.AddTail(sPacket);

mMutex.Unlock();

// 日志输出信息

pDlg->m\_log.InsertString(-1, L" 缺少目的MAC地址，将IP数据包存入转发

缓冲区");

pDlg->m\_log.InsertString(-1, (L" 存入转发缓冲区的数据包为：" +

IPntoa(IPf->IPHeader.SrcIP)

+ L"->" + IPntoa(IPf->IPHeader.DstIP) + L" " + MACntoa(IPf-

>FrameHeader.SrcMAC)

+ L"->xx:xx:xx:xx:xx:xx"));

pDlg->m\_log.InsertString(-1, L" 发送ARP请求");

// 发送ARP请求

ARPRequest(IfInfo[sPacket.IfNo].adhandle,

IfInfo[sPacket.IfNo].MACAddr,

IfInfo[sPacket.IfNo].ip[1].IPAddr, sPacket.TargetIP); }

else // 如缓存队列太长，抛弃该报

{

// 日志输出信息

pDlg->m\_log.InsertString(-1, L" 转发缓冲区溢出，丢弃IP数据包");

pDlg->m\_log.InsertString(-1, (L" 丢弃的IP数据包为：" +

IPntoa(IPf->IPHeader.SrcIP) + L"->"

+ IPntoa(IPf->IPHeader.DstIP) + L" " + MACntoa(IPf-

>FrameHeader.SrcMAC)

+ L"->xx:xx:xx:xx:xx:xx"));

}

}

}

}

IP 数据包转发时，查询路由表来获得下一跳地址。为了进行最长匹配，设置一个变量记录以匹配到的掩 码地址，然后遍历路由表，找到目的 IP 所在的网络且掩码最长的路由表项，获得其下一跳地址。若为直 接投递，则将目的 IP 地址设置为原来的 IP 地址

// 查询路由表

DWORD RouteLookup(UINT& ifNO, DWORD desIP, CList <RouteTable\_t, RouteTable\_t&>\* routeTable)

{

// desIP为网络序

DWORD MaxMask = 0; // 获得最大的子网掩码的地址，没有获得时初始化为-1

int Index = -1; // 获得最大的子网掩码的地址对应的路由表索引，以便获得下一站路由器的地址

POSITION pos;

RouteTable\_t rt;

DWORD tmp;

pos = routeTable->GetHeadPosition();

for (int i = 0; i < routeTable->GetCount(); i++)

{

rt = routeTable->GetNext(pos);

if ((desIP & rt.Mask) == rt.DstIP)

{

Index = i;

if (rt.Mask >= MaxMask)

{

MaxMask = rt.Mask;

ifNO = rt.IfNo;

if (rt.NextHop == 0) // 直接投递

{

tmp = desIP;

}

else

{

tmp = rt.NextHop;

}

}

}

}

if (Index == -1)

// 目的不可达

{

return -1;

}

else // 找到了下一跳地址

{

return tmp;

}

}

**11、发送ICMP数据包**

若发生超时或者目的不可达，则发送 ICMP 数据包，填充帧首部，将源 MAC 地址和目的 MAC 地址互

换，并设置帧类型为 IP 帧。填充 IP 首部，修改源 IP 地址为本机网卡上的 IP 地址。填充 ICMP 首部，根 据传入的参数设置 Type 和 code ，计算校验和并填充。最后填充数据并发送数据包、打印日志信息

// 发送ICMP数据包

void ICMPPacketProc(IfInfo\_t\* pIfInfo, BYTE type, BYTE code, const u\_char\* pkt\_data)

{

u\_char\* ICMPBuf = new u\_char[70];

// 填充帧首部

memcpy(((FrameHeader\_t\*)ICMPBuf)->DesMAC, ((FrameHeader\_t\*)pkt\_data)- >SrcMAC, 6);

memcpy(((FrameHeader\_t\*)ICMPBuf)->SrcMAC, ((FrameHeader\_t\*)pkt\_data)- >DesMAC, 6);

((FrameHeader\_t\*)ICMPBuf)->FrameType = htons(0x0800);

// 填充IP首部

((IPHeader\_t\*)(ICMPBuf + 14))->Ver\_HLen = ((IPHeader\_t\*)(pkt\_data + 14))- >Ver\_HLen;

((IPHeader\_t\*)(ICMPBuf + 14))->TOS = ((IPHeader\_t\*)(pkt\_data + 14))->TOS; ((IPHeader\_t\*)(ICMPBuf + 14))->TotalLen = htons(56);

((IPHeader\_t\*)(ICMPBuf + 14))->ID = ((IPHeader\_t\*)(pkt\_data + 14))->ID;

((IPHeader\_t\*)(ICMPBuf + 14))->Flag\_Segment = ((IPHeader\_t\*)(pkt\_data + 14))->Flag\_Segment;

((IPHeader\_t\*)(ICMPBuf + 14))->TTL = 64;

((IPHeader\_t\*)(ICMPBuf + 14))->Protocol = 1;

//((IPHeader\_t \*)(ICMPBuf+14))->SrcIP = ((IPHeader\_t \*)(pkt\_data+14))- >DstIP;

((IPHeader\_t\*)(ICMPBuf + 14))->SrcIP = pIfInfo->ip[1].IPAddr;

((IPHeader\_t\*)(ICMPBuf + 14))->DstIP = ((IPHeader\_t\*)(pkt\_data + 14))- >SrcIP;

((IPHeader\_t\*)(ICMPBuf + 14))->Checksum = htons(ChecksumCompute((unsigned short\*)(ICMPBuf + 14), 20));

// 填充ICMP首部

((ICMPHeader\_t\*)(ICMPBuf + 34))->Type = type;

((ICMPHeader\_t\*)(ICMPBuf + 34))->Code = code;

((ICMPHeader\_t\*)(ICMPBuf + 34))->Id = 0;

((ICMPHeader\_t\*)(ICMPBuf + 34))->Sequence = 0;

((ICMPHeader\_t\*)(ICMPBuf + 34))->Checksum = htons(ChecksumCompute((unsigned short\*)(ICMPBuf + 34), 8));

// 填充数据

memcpy((u\_char\*)(ICMPBuf + 42), (IPHeader\_t\*)(pkt\_data + 14), 20); memcpy((u\_char\*)(ICMPBuf + 62), (u\_char\*)(pkt\_data + 34), 8);

// 发送数据包

pcap\_sendpacket(pIfInfo->adhandle, (u\_char\*)ICMPBuf, 70);

// 日志输出信息

if (type == 11)

{

pDlg->m\_log.InsertString(-1, L" 发送ICMP超时数据包：");

}

if (type == 3)

{

pDlg->m\_log.InsertString(-1, L" 发送ICMP目的不可达数据包：");

}

pDlg->m\_log.InsertString(-1, (L" ICMP ->" + IPntoa(((IPHeader\_t\*)(ICMPBuf

+ 14))->DstIP)

+ L"-" + MACntoa(((FrameHeader\_t\*)ICMPBuf)->DesMAC)));

delete[] ICMPBuf;

}

**12、校验和计算与检验**

将数据包首部的数据每16位相加，若有进位则加到最低位，取反，得到校验和。检验校验和的过程类 似，先计算校验和，并与数据包首部的校验和进行比较，如果相同则数据包传输正确，反之则数据包传 输过程中丢失了数据

// 判断IP数据包头部校验和是否正确

int IsChecksumRight(char\* buffer)

{

// 获得IP头内容

IPHeader\_t\* ip\_header = (IPHeader\_t\*)buffer;

// 备份原来的校验和

unsigned short checksumBuf = ip\_header->Checksum;

unsigned short check\_buff[sizeof(IPHeader\_t)];

// 设IP头中的校验和为0

ip\_header->Checksum = 0;

memset(check\_buff, 0, sizeof(IPHeader\_t));

memcpy(check\_buff, ip\_header, sizeof(IPHeader\_t));

// 计算IP头部校验和

ip\_header->Checksum = ChecksumCompute(check\_buff, sizeof(IPHeader\_t));

// 与备份的校验和进行比较

if (ip\_header->Checksum == checksumBuf)

return 1;

else

return 0;

}

// 计算校验和

unsigned short ChecksumCompute(unsigned short\* buffer, int size)

{

// 32位，延迟进位

unsigned long cksum = 0;

while (size > 1)

{

cksum += \*buffer++;

// 16位相加

size -= sizeof(unsigned short);

}

if (size)

{

// 最后可能有单独8位

cksum += \*(unsigned char\*)buffer;

}

// 将高16位进位加至低16位

cksum = (cksum >> 16) + (cksum & 0xffff);

cksum += (cksum >> 16);

// 取反

return (unsigned short)(~cksum);

}

**13、添加路由表项**

在程序中输入掩码地址、目的网络、下一跳地址后，点击添加，记录下输入框中的内容，检查合法性。 如果路由表中已经包含了一条掩码地址相同、目的网络号相同但是下一跳地址不相同的路由表项，输入 错误，弹出提示框。检验下一跳地址的网络是否是该网卡所处的网络，如果不是，输入错误，弹出提示 框。如果通过了合法性检查，把该条路由表项添加到路由表，并且在路由表窗口中显示该路由表项

void CrouterDlg::OnBnClickedButtonAdd()

{

// TODO: 在此添加控件通知处理程序代码

bool flag;

int i, j;

DWORD ipaddr;

RouteTable\_t rt;

m\_next.GetAddress(ipaddr);

ipaddr = htonl(ipaddr);

// 检查合法性

DWORD ipaddr1;

DWORD ipaddr2;

DWORD ipaddr3;

POSITION pos, CurrentPos;

// 记录子网掩码

m\_mask.GetAddress(ipaddr1);

// 记录目的IP

m\_dest.GetAddress(ipaddr2);

// 记录下一跳

m\_next.GetAddress(ipaddr3);

pos = RouteTable.GetHeadPosition();

for (i = 0; i < RouteTable.GetCount(); i++)

{

CurrentPos = pos;

rt = RouteTable.GetNext(pos);

if ((rt.Mask == htonl(ipaddr1)) && ((rt.Mask & rt.DstIP) == (rt.Mask & htonl(ipaddr2))) && (htonl(ipaddr3) != rt.NextHop))

{

MessageBox(L"该路由无法添加，请重新输入！");

return;

}

}

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))

{

rt.IfNo = i;

// 记录子网掩码

m\_mask.GetAddress(ipaddr);

rt.Mask = htonl(ipaddr);

// 记录目的IP

m\_dest.GetAddress(ipaddr);

rt.DstIP = htonl(ipaddr);

// 记录下一跳

m\_next.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;

}

}

}

if (!flag)

{

MessageBox(L"输入错误，请重新输入！");

}

}

**14、删除路由表项**

选中需要删除的路由表项，分别获得子网掩码选项、目的地址选项以及下一跳地址选项，将其从路由表 窗口中删除，遍历路由表，把需要删除的路由表项从路由表中删除，若下一跳地址为0，为直接连接路 由，不允许删除

void CrouterDlg::OnBnClickedButtonDelete()

{

// TODO: 在此添加控件通知处理程序代码

int i;

char str[100], ipaddr[20];

ULONG mask, destination, nexthop;

RouteTable\_t rt;

POSITION pos, CurrentPos;

str[0] = NULL;

ipaddr[0] = NULL;

if ((i = m\_routeTable.GetCurSel()) == LB\_ERR)

{

return;

}

CString STR;

m\_routeTable.GetText(i, STR);

for (int i = 0; i < STR.GetLength(); i++)

str[i] = STR[i];

str[STR.GetLength()] = '\0';

// 取得子网掩码选项

strncat\_s(ipaddr, str, 15);

mask = inet\_addr(ipaddr);

// 取得目的地址选项

ipaddr[0] = 0;

strncat\_s(ipaddr, &str[19], 15);

destination = inet\_addr(ipaddr);

// 取得下一跳选项

ipaddr[0] = 0;

strncat\_s(ipaddr, &str[38], 15);

nexthop = inet\_addr(ipaddr);

if (nexthop == 0)

{

MessageBox(L"直接连接路由，不允许删除！");

return;

}

// 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);

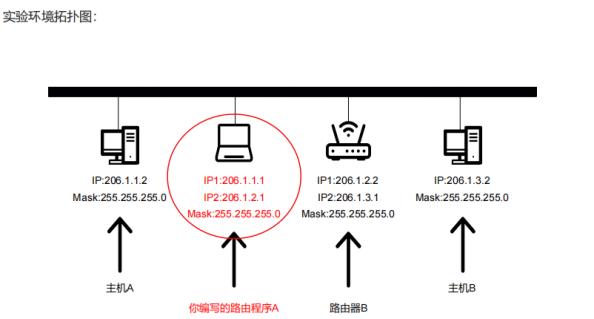
return;

}

}

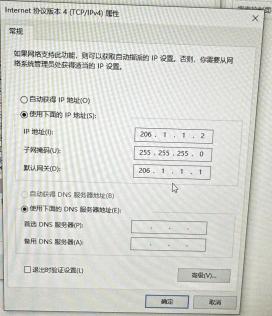
}

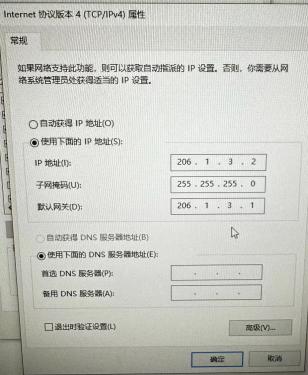
**实验环境配置**



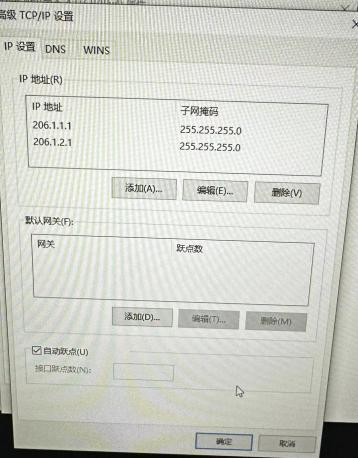
**1、配置主机的IP 地址和默认路由**

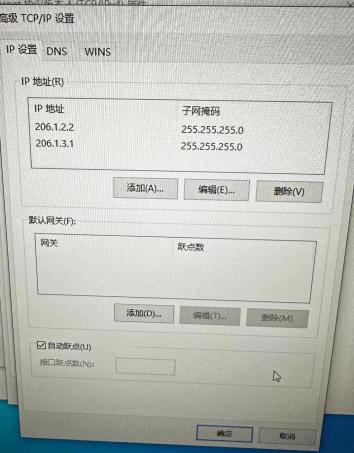
配置主机A和主机B的IP地址和子网掩码：





**2、配置路由设备的IP地址**



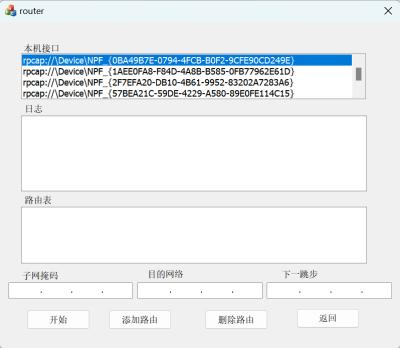


**3、利用命令行程序配置路由设备的静态路由**

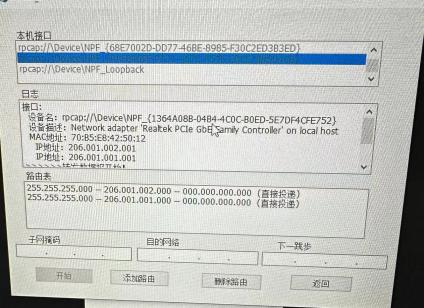
在第二台路由设备的终端输入 route add 206.1.1.0 mask 255.255.255.0 206.1.2.1 （增加一条路 由表项，目的网络<206.1.1.0> ，掩码<255.255.255.0> ，下一路由器地址<206.1.2.1>）

**4、启动路由器**

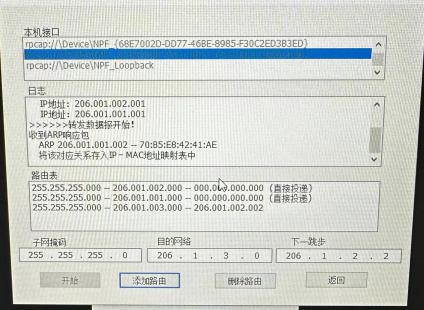
路由器程序如图所示：



选择一个网卡，可以看到mac地址和两个IP地址还有路由表：



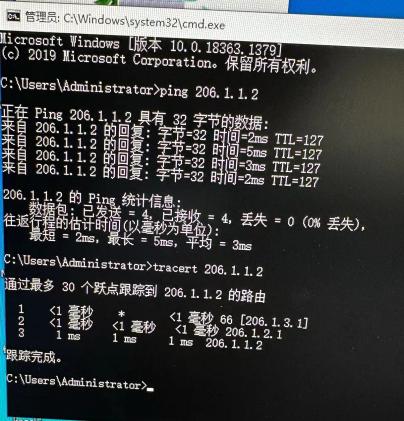
添加一条掩码地址<255.255.255.0> ，目的地址<206.1.3.0> ，下一跳步地址<206.1.2.2>：



**5、测试网络连通性**

在主机1终端输入ping <206.1.3.2>,tracert <206.1.3.2>;在主机2终端输入ping <206.1.1.2>,tracert <206.1.1.2>:





全部ping通

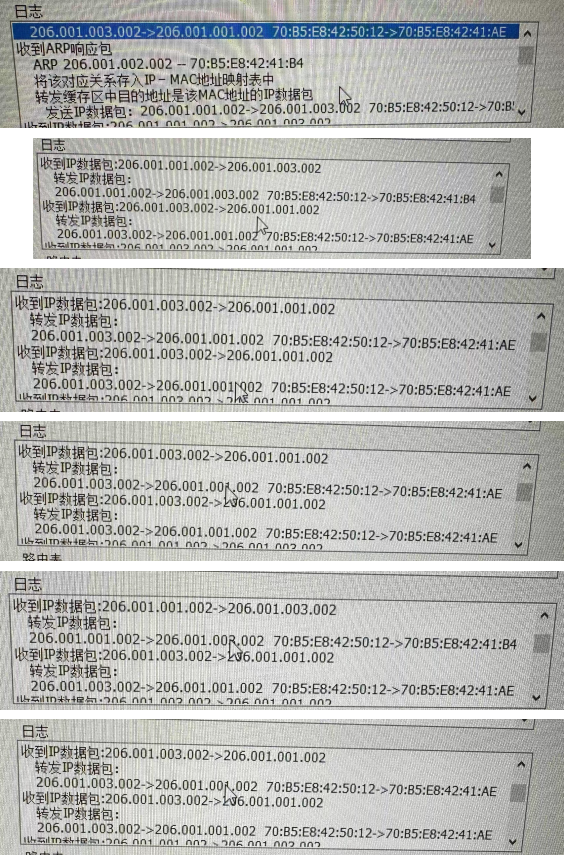
**6、路由器输出日志信息**

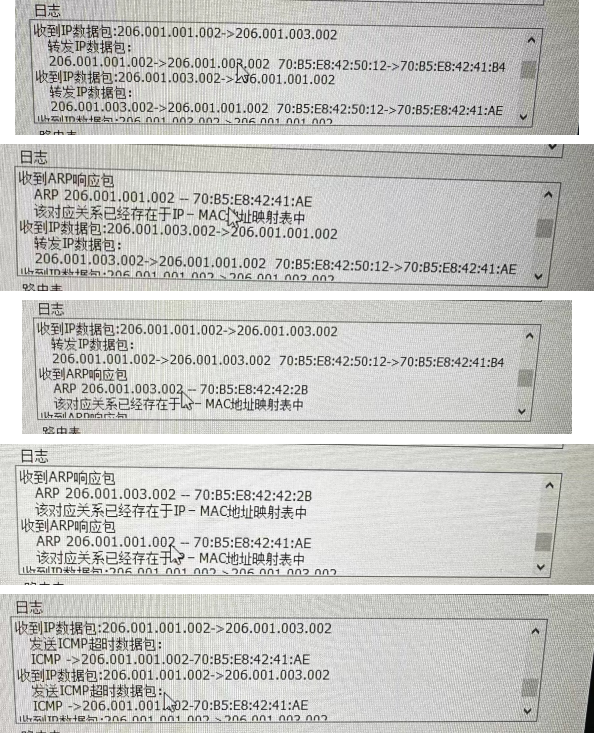
首先， <206.1.1.2>给<206.1.3.2>发送一个数据包，由于第一次发送，路由器缺少<206.1.3.2>的MAC地址，因 此将IP数据包存入转发缓冲区，同时发送ARP数据包请求<206.1.3.2>的MAC地址。然后路由器收到ARP响 应包，获得<206.1.3.2>的MAC地址，并将IP-MAC映射关系存入IP-MAC地址映射表中，然后将缓冲区存入 的<206.1.1.2>—><206.1.3.2>的IP数据包转发，在转发之前路由器查询了路由表找到下一跳地址。

路由器收到<206.1.3.2>给<206.1.1.2>回复的数据包，路由器没有<206.1.1.2>的MAC地址，先将该数据包存入 转发缓冲区，然后发送ARP数据包请求<206.1.1.2>的MAC地址。然后路由器收到ARP响应包，将<206.1.1.2> 和MAC地址的对应关系存入IP-MAC地址映射表中，然后将缓冲区中存入的<206.1.3.2>—><206.1.1.2>的IP数 据包转发。

路由器再次收到<206.1.1.2>给<206.1.3.2>发送的数据包，路由器存入了<206.1.3.2>的IP与MAC地址的对应关 系，通过查找IP-MAC地址映射表即可获得，因此在获得路由表中的下一跳地址后直接转发。对<206.1.3.2> 给<206.1.1.2>发送的数据包同样直接查找IP-MAC地址映射表即可获得<206.1.1.2>的MAC地址，完成转发。

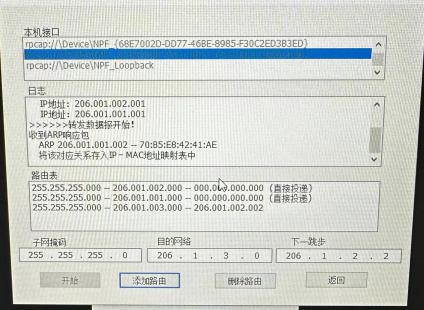
在运行tracert命令时路由器会发送ICMP超时数据包，若收到的数据包的目的IP地址不可达则会发送 ICMP目的不可达数据包





**7、添加、删除路由表项**

添加路由表会进行简单的合法性检查，不合法会输出“该路由无法添加，请重新输入！ ℽ



删除路由表也会进行合法性检查，直接连接路由，不允许删除

