实验5：简单路由器程序的设计

学号：2010349
姓名：孟笑朵
年级：2020级
专业：计算机科学与技术
撰写时间：2022年12月15日

实验内容说明

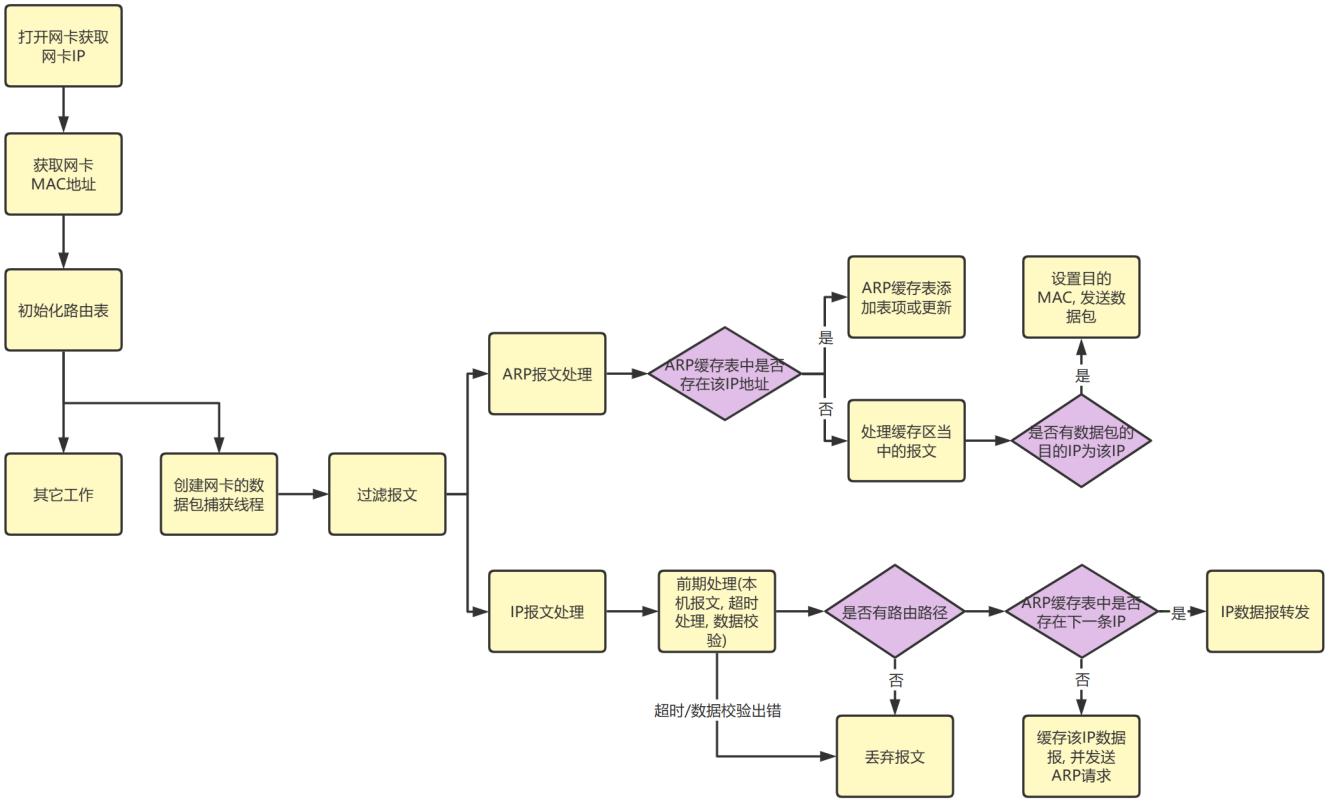
（1）设计和实现一个路由器程序，要求完成的路由器程序能和现有的路由器产品（如思科路由器、华为路由器、微软的路由器等）进行协同工作。
（2）程序可以仅实现IP数据报的获取、选路、投递等路由器要求的基本功能。可以忽略分片处理、选项处理、动态路由表生成等功能。
（3）需要给出路由表的手工插入、删除方法。
（4）需要给出路由器的工作日志，显示数据报获取和转发过程。
（5）完成的程序须通过现场测试，并在班（或小组）中展示和报告自己的设计思路、开发和实现过程、测试方法和过程。

本程序在虚拟环境中运行，需要达到的预期实验结果如下：

1. 获取并显示本机单网卡的双IP地址和MAC地址;
2. 可以显示路由表，可以添加路由表项(206.1.3.0/255.255.255.0/206.1.2.2)，可以删除路由表项但无法删除
   默认路由表项;
3. 显示工作日志，输出数据报获取和转发过程(至少展示数据报的源IP、目的IP、下一跳信息);
4. 主机A和B可以互相ping通，且显示的TTL正确(126或62);

实验准备

在设计路由程序之前，我们首先需要设计**整个路由程序的总体框架，关键数据结构和路由程序的界面**，如下图所示为本路由程序设计的框架：



在数据结构方面我们需要设计一些程序中需要用到的类，包含常规的类：

- 以太帧的数据结构；
- ARP数据结构；
- IP头的数据结构；
- IP报文数据结构；

另外，我们需要专门设计本次路由器的数据结构：

- 路由表的表项的结构；
- ARP缓存表项的结构；
- 缓存区的数据结构；

> 注：方便起见，我们将路由器的网卡信息也设计一个数据结构方便使用

如下所示为全部为本程序所用到的所有数据结构：

C++

```cpp
typedef struct FrameStruct {    // 帧首部
    BYTE    DesMAC[6];          // 目的地址
    BYTE    SrcMAC[6];          // 源地址
    WORD    FrameType;          // 帧类型
} FrameStruct;

typedef struct ARPStruct {     // ARP帧
    FrameStruct FrameHeader;  // 帧头部结构体
    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;             // 操作字段
    BYTE SendHa[6];             // 源MAC地址
    DWORD SendIP;               // 源IP地址
    BYTE RecvHa[6];             // 目的MAC地址
    DWORD RecvIP;               // 目的IP地址
} ARPStruct;

typedef struct IPStruct {       // 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
} IPStruct;

typedef struct PacketStruct {           // 包含帧首部和IP首部的数据包
    FrameStruct FrameHeader;  // 帧首部
    IPStruct IPHeader;        // IP首部
} PacketStruct;

#pragma pack()                  // 恢复默认对齐方式

typedef struct PacketCache {       // 缓存队列的数据包
```

```
    DWORD ip;                   // 目的IP地址
    int IfNo;                   // 接口序号
    int len;                    // 数据包长度
    char data[65535];           // 数据包
} PacketCache;

typedef struct DeviceInfo {     // 接口信息
    char DeviceName[64];        // 设备名
    char Description[128];      // 设备描述
    BYTE MACAddr[6];            // MAC地址
    DWORD ip[5];                // IP地址列表
    DWORD netmask[5];           // 掩码地址列表
    pcap_t* adhandle;           // pcap句柄
} IfInfo_t;

typedef struct MyRouteTable {   // 路由表表项
    DWORD Mask;                 // 子网掩码
    DWORD DstIP;                // 目的地址
    DWORD NextHop;              // 下一跳步
} MyRouteTable;

typedef struct ARPTable {       // IP-MAC地址映射表表项
    DWORD IPAddr;               // IP地址
    BYTE MACAddr[6];            // MAC地址
} ARPTable;
```

本次程序的界面设计如下:

实验过程

下面结合程序代码进行补充说明代码框架中的内容. 在本程序中路由表的设计, ARP缓存和数据报缓存都是以较为简单的数组进行存储的, 如下所示:

```C++
vector<ARPTable> IPMACTable;
vector<MyRouteTable> RouteTable;
vector<PacketCache> cache;
```

在日志的记录过程中, 我们将日志存储在字符串当中, 然后输出到程序的日志界面当中, 为了方便查看, 我们也将日志存储在txt文件当中:

```C++
string info;
void WriteLog() {
    FILE* fptr;
    fptr = fopen("log.txt", "w");
    if (fptr == NULL)
    {
        printf("Error!");
        exit(1);
    }
    fprintf(fptr, "%s", info.c_str());
    fclose(fptr);
}
```

1. 获取网卡与网卡IP地址

获取网卡比较简单这里不做说明,获取网卡IP只需要进行遍历设备`adapters->addresses`当中就可以,关键代码如下所示:

```C++
    //初始化的时候就进行设备选择等
    pcap_if_t* adapters;
    char errbuf[PCAP_ERRBUF_SIZE];
    if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &adapters, errbuf)
== -1) {
        info.append("Error in pcap_findalldevs_ex :").append(errbuf);
        log.SetWindowTextW(CA2T(info.c_str()));
        return -1;
    }

    int flag = 0, ip_num = 0;
    for (pcap_addr_t* a = adapters->addresses; a != NULL; a = a->next) {
        if (a->addr->sa_family == AF_INET) {
            flag = 1;
            strcpy(nowAdapter.DeviceName, adapters->name);
            if (adapters->description) {
                strcpy(nowAdapter.Description, adapters->description);
            }
            else {
                strcpy(nowAdapter.Description, "null");
            }
            //考虑到一块网卡可能绑定多个IP地址
            nowAdapter.ip[ip_num] = ((struct sockaddr_in*)a->addr)-
>sin_addr.S_un.S_addr;
            nowAdapter.netmask[ip_num] = ((struct sockaddr_in*)a-
>netmask)->sin_addr.S_un.S_addr;
            ip_num++;
        }
    }
    if (flag == 0) {
        MessageBox(_T("error!"));
        return 0;
    }
```

2. 获取本机网卡MAC地址

分为以下几步进行获取:
1. 设置目的IP地址为网卡IP地址, 这里只需要设置网卡的第一个IP地址就行, 一张网卡只有一个MAC地址;
2. 设置目的MAC地址为0xFFFFFFFFFFFF;
3. 设置ARP为ARP请求;
4. 获取目的IP为网卡IP的目的报文, 得到网卡MAC地址;

C++

```
//获取本地网卡MAC地址
void GetDeviceMAC(LPVOID lpParameter) {
    DeviceInfo* device = (DeviceInfo*)lpParameter;
    ARPStruct* ARPFrame;
    pcap_pkthdr* header;
    const u_char* pkt_data;
    UCHAR SrcMAC[6];
    DWORD SrcIP;
    for (int i = 0; i < 6; i++) {
        SrcMAC[i] = 0x55;
    }
    SrcIP = inet_addr("5.5.5.5");
    // 发送ARP报文
    ARPRequest(device->adhandle, SrcMAC, SrcIP, device->ip[0]);
    int res;
    while (true) {
        res = pcap_next_ex(device->adhandle, &header, &pkt_data);
        if (res == 0) {
            // 超时时间到
            continue;
        }
        else if (res == 1) {
            ARPFrame = (ARPStruct*)(pkt_data);
            if (ARPFrame->FrameHeader.FrameType == htons(0x0806)
                && ARPFrame->Operation == htons(0x0002)
                && ARPFrame->SendIP == device->ip[0]) {
                ARPTable ipmac;
                for (int i = 0; i < 6; i++) {
                    device->MACAddr[i] = ARPFrame->SendHa[i];
                    ipmac.MACAddr[i] = ARPFrame->SendHa[i];
                }
                for (int j = 0; device->ip[j] != '\0'; j++) {
                    ipmac.IPAddr = device->ip[j];
```

```
                    IPMACTable.push_back(ipmac);
                }
                return;
            }
        }
    }
    return;
}

//发送ARP请求
void ARPRequest(pcap_t* adhandle, BYTE* SrcMAC, DWORD SendIp, DWORD
RecvIp) {
    ARPStruct  ARPFrame;
    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 = SendIp;
    ARPFrame.RecvIP = RecvIp;

    for (int i = 0; i < 6; i++)
    {
        ARPFrame.FrameHeader.DesMAC[i] = 0xff;
        ARPFrame.FrameHeader.SrcMAC[i] = SrcMAC[i];
        ARPFrame.SendHa[i] = SrcMAC[i];
        ARPFrame.RecvHa[i] = 0x00;
    }
    if (pcap_sendpacket(adhandle, (u_char*)&ARPFrame, sizeof(ARPStruct))
!= 0) {
        return;
    }
}
```

3. 初始化路由表与路由表的增删改查

- 路由表的初始化:
  在获取网卡的MAC地址之后, 设置路由表项掩码为网卡IP掩码, 目的IP为网卡IP与网卡掩码取与, 下一跳设置为0, 代表直接投递.

```C++
    // 初始化路由表
    MyRouteTable rt;
    // 直接投递的路由表项
    for (int i = 0; i < ip_num; i++) {
        rt.Mask = nowAdapter.netmask[i];
        rt.DstIP = nowAdapter.ip[i] & nowAdapter.netmask[i];
        rt.NextHop = 0;
        RouteTable.push_back(rt);
    }
```

- 路由表的添加
  当按下"添加路由"后进行路由表项的添加,添加的方式比较简单:设置目的IP,掩码,下一跳IP即可将路由添加在路由表当中

```C++
void CmyRouterDlg::OnBnClickedButton1()
{
    MyRouteTable rt;
    cmask.GetAddress(mask);
    //如果不逆序的话，最终输出的IP是反着的
    WORD hiWord = HIWORD(mask);
    WORD loWord = LOWORD(mask);
    BYTE nf1 = HIBYTE(hiWord);
    BYTE nf2 = LOBYTE(hiWord);
    BYTE nf3 = HIBYTE(loWord);
    BYTE nf4 = LOBYTE(loWord);
    DWORD rmask = nf1 | nf2 << 8 | nf3 << 16 | nf4 << 24;
    rt.Mask = rmask;

    cdes.GetAddress(des);
    hiWord = HIWORD(des);
    loWord = LOWORD(des);
    nf1 = HIBYTE(hiWord);
    nf2 = LOBYTE(hiWord);
    nf3 = HIBYTE(loWord);
    nf4 = LOBYTE(loWord);
    DWORD rdes = nf1 | nf2 << 8 | nf3 << 16 | nf4 << 24;
    rt.DstIP = rdes;

    cnext.GetAddress(next);
    hiWord = HIWORD(next);
```

```
    loWord = LOWORD(next);
    nf1 = HIBYTE(hiWord);
    nf2 = LOBYTE(hiWord);
    nf3 = HIBYTE(loWord);
    nf4 = LOBYTE(loWord);
    DWORD rnext = nf1 | nf2 << 8 | nf3 << 16 | nf4 << 24;
    rt.NextHop = rnext;

    RouteTable.push_back(rt);
    info.append("已增加该路由表项\r\n");
    log.SetWindowTextW(CA2T(info.c_str()));
    string table = ShowRouteTable();
    Ctable.SetWindowTextW(CA2T(table.c_str()));
}
```

- 路由表的删除
  当按下"删除路由"后进行路由表项的删除,删除的过程分为以下几步:
  1. 判断下一跳的IP是否为0,如果为0则不能删除;
  2. 判断是否存在该路由表项,存在则删除,不存在则不能删除;

C++

```
void CmyRouterDlg::OnBnClickedButton2()
{
    cmask.GetAddress(mask);
    //如果不逆序的话，最终输出的IP是反着的
    WORD hiWord = HIWORD(mask);
    WORD loWord = LOWORD(mask);
    BYTE nf1 = HIBYTE(hiWord);
    BYTE nf2 = LOBYTE(hiWord);
    BYTE nf3 = HIBYTE(loWord);
    BYTE nf4 = LOBYTE(loWord);
    DWORD rmask = nf1 | nf2 << 8 | nf3 << 16 | nf4 << 24;

    cdes.GetAddress(des);
    hiWord = HIWORD(des);
    loWord = LOWORD(des);
    nf1 = HIBYTE(hiWord);
    nf2 = LOBYTE(hiWord);
    nf3 = HIBYTE(loWord);
    nf4 = LOBYTE(loWord);
    DWORD rdes = nf1 | nf2 << 8 | nf3 << 16 | nf4 << 24;
```

```
    cnext.GetAddress(next);
    hiWord = HIWORD(next);
    loWord = LOWORD(next);
    nf1 = HIBYTE(hiWord);
    nf2 = LOBYTE(hiWord);
    nf3 = HIBYTE(loWord);
    nf4 = LOBYTE(loWord);
    DWORD rnext = nf1 | nf2 << 8 | nf3 << 16 | nf4 << 24;
    // TODO: Add your control notification handler code here
    if (rnext == 0) {
        info.append("直接投递路由，不能删除\r\n");
        log.SetWindowTextW(CA2T(info.c_str()));
        return;
    }
    if (RouteTable.empty()) {
        return;
    }
    // 遍历路由表项
    vector<MyRouteTable>::iterator i;
    for (i = RouteTable.begin(); i != RouteTable.end(); i++) {
        if ((i->DstIP == rdes)
            && (i->Mask == rmask)
            && (i->NextHop == rnext)) {
            RouteTable.erase(i);
            info.append("已删除该路由表项\r\n");
            log.SetWindowTextW(CA2T(info.c_str()));
            return;
        }
    }
    info.append("不存在该路由表项\r\n");
    log.SetWindowTextW(CA2T(info.c_str()));
    string table = ShowRouteTable();
    Ctable.SetWindowTextW(CA2T(table.c_str()));
}
```

- 路由表的查找
  查找的方式是从头到尾遍历路由表,选择其中掩码最长的且目的IP和掩码的与刚好对应表项当中的目的IP的值

```C++
// 查询路由表：策略是从头查到尾，选择掩码长度最长的
// 下一跳为0的直接投递
DWORD SearchRouteTable(DWORD DstIP) {
    DWORD temp;
    int flag = 0;
    DWORD maxmask = 0;
    vector<MyRouteTable>::iterator i;
    for (i = RouteTable.begin(); i != RouteTable.end(); i++) {
        if ((DstIP & i->Mask) == i->DstIP) {
            if (i->Mask >= maxmask) {
                flag = 1;
                if (i->NextHop == 0) {// 直接投递
                    temp = DstIP;
                }
                else {
                    temp = i->NextHop;
                }
                maxmask = i->Mask;
            }
        }
    }
    if (flag) {
        return temp;
    }
    return -1;
}
```

> 对应ARP缓存表的更新和查找也是同样的道理,这里就不做说明了

4. 数据报的捕获与过滤

根据数据报的类型进行区分:

1. 以太帧的类型为`0x0806`时为`ARP数据报`;
2. 以太帧的类型为`0x0800`时为`IP数据报`;

```C++
//4. 捕获报文的过滤条件（ARP & IP）;
DWORD WINAPI CapturePacket(LPVOID lpParameter) {
    DeviceInfo* device = (DeviceInfo*)lpParameter;
    pcap_pkthdr* header;
```

```
    const u_char* pkt_data;
    int res;
    while (true) {
        // 捕获接口的数据包
        res = pcap_next_ex(device->adhandle, &header, &pkt_data);
        if (res == 0) {
            // 超时时间到
            continue;
        }
        else if (res == 1) {
            FrameStruct* FrameHeader;
            FrameHeader = (FrameStruct*)pkt_data;
            // 判断数据报类型
            if (memcmp(FrameHeader->DesMAC, device->MACAddr, 6) == 0) {
                // 如果是ARP数据报：ARP回应报文
                if (ntohs(FrameHeader->FrameType) == 0x0806) {
                    DealARP(header, pkt_data);
                }
                // 如果是IP数据报：能处理处理，不能处理暂时放入缓存区当中
                else if (ntohs(FrameHeader->FrameType) == 0x0800) {
                    DealIP(header, pkt_data);
                }
            }
        }
    }
    return 0;
}
```

5. 捕获ARP数据报的处理

这里只处理ARP数据报的响应报文:

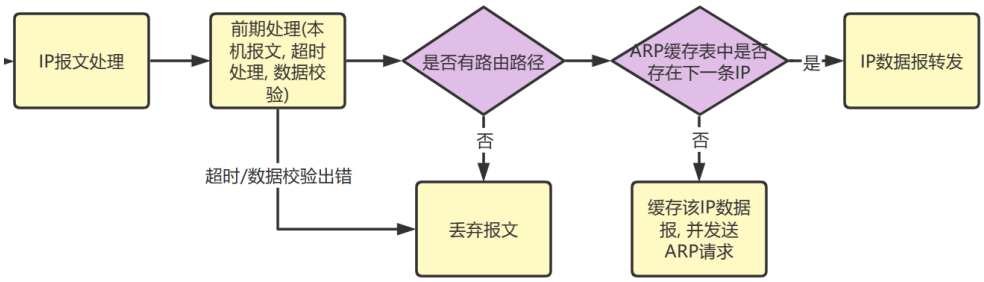
1. 如果IP和MAC的对应关系已存在, 查看是否需要更新IP和MAC地址的对应关系;
2. 如果IP和MAC的对应关系不存在, 添加该对应关系到ARP缓存表当中, 查看数据包缓存区中是否存在该对应关系相关的数据包, 如果存在则发送该数据报;

```C++
// 处理ARP数据报
void DealARP(struct pcap_pkthdr* header, const u_char* pkt_data) {
    // 更新IP地址和MAC地址的对应关系
    ARPStruct* ARP;
    ARP = (ARPStruct*)pkt_data;
```

```
    BYTE mac[6];
    if (ARP->Operation == htons(0x0002)) {
        // 收到响应数据报，搜索ARP缓存表
        if (MACTableSearch(ARP->SendIP, mac)) {
            // 判断MAC地址映射表中的MAC地址是否为最新
            if (memcmp(mac, ARP->SendHa, 6) == 0) {
                return;
            }
            else {
                MACTableUpdate(ARP->SendIP, ARP->SendHa);
            }
        }
        else {
            IP_MAC_t ipmac;
            ipmac.IPAddr = ARP->SendIP;
            for (int i = 0; i < 6; i++) {
                ipmac.MACAddr[i] = ARP->SendHa[i];
            }
            IPMACTable.push_back(ipmac);
            info.append("该映射关系已保存。\r\n\r\n");
            if (PacketQueue.empty()) {
                return;
            }
            list<Packet_t>::iterator i;// 遍历数据包缓存队列查看是否有可以转
发的IP数据包
            int j = 0;
            for (i = PacketQueue.begin(); i != PacketQueue.end() && j <
PacketQueue.size(); i++, j++) {
                if (i->ip == ARP->SendIP) {// 如果有数据包的目标IP地址为ARP
包IP地址
                    Data_t* IPFrame;
                    IPFrame = (Data_t*)i->data;
                    for (int j = 0; j < 6; j++) {
                        IPFrame->FrameHeader.DesMAC[j] = ARP->SendHa[j];
                    }
                    if (pcap_sendpacket(nowAdapter.adhandle, (u_char*)i-
>data, i->len) == -1) {
                        info.append("发送失败。\r\n\r\n");
                        return;
                    }
```

```
                    else {
                        PacketQueue.erase(i);
                        info.append("转发缓存区中目的地址是该MAC地址的IP数据包,转发IP数据包: \n");
                    }
                }
            }
        }
    }
}
```

6. 捕获IP数据报的处理与缓存区的处理



- 前期处理的部分
  - 本地发出的报文不进行处理, 即源MAC地址为路由器的MAC地址;
  - 超时的进行丢弃;
  - 数据包的校验出错的进行丢弃;
- 路由选择出错的不进行处理;
- 处理存在路由路径的:
  - 查找ARP缓存表中下一跳的IP地址是否存在对应的MAC地址;
  - 若存在, 则进行IP数据报的转发, 需要设置当前源MAC地址为路由器的MAC地址, 设置TTL - 1;
  - 若不存在, 则进行IP数据报的缓存, 并进行ARP请求;

> 如果缓冲区满的话, 就直接丢弃该数据报.

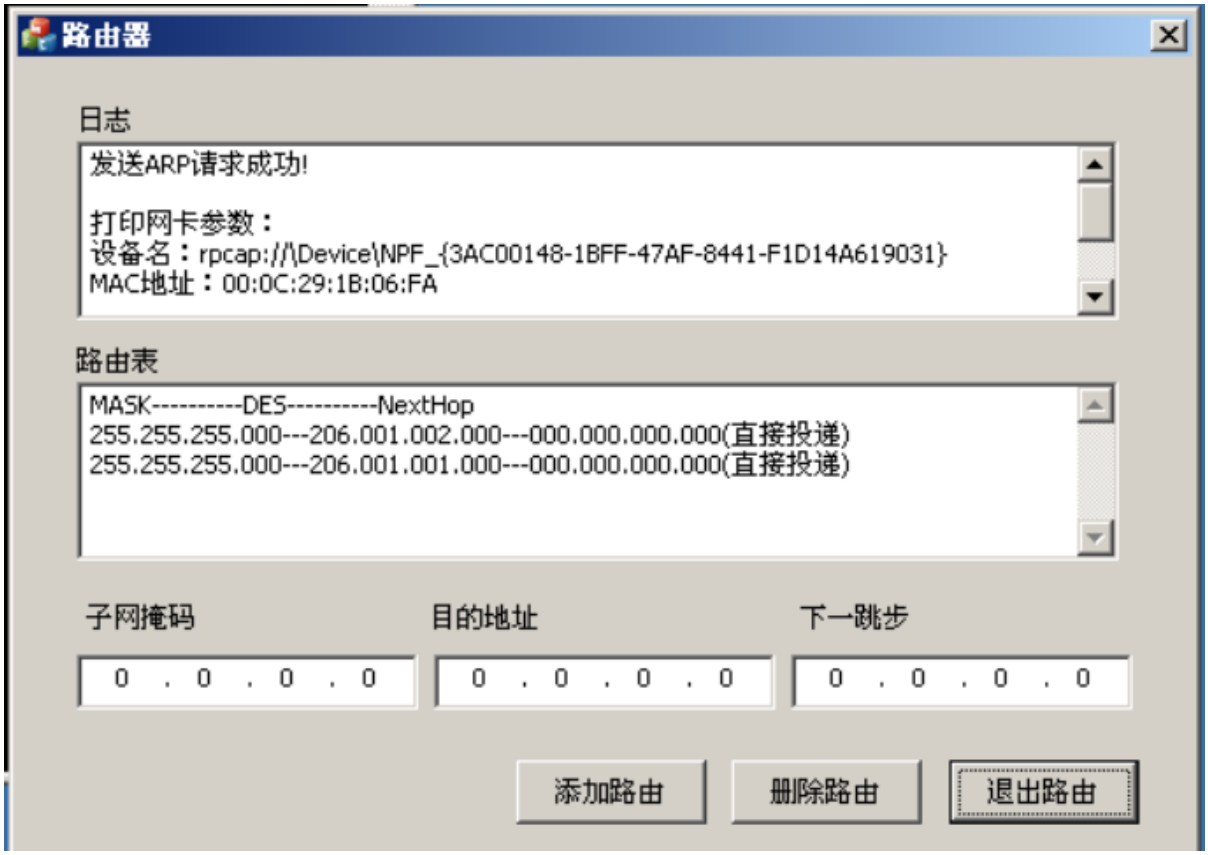
```C++
// 处理IP数据包
void DealIP(struct pcap_pkthdr* header, const u_char* pkt_data) {
    PacketStruct* IPFrame;
    IPFrame = (PacketStruct*)pkt_data;
    // 本地发出的报文不进行处理
```

```
    if (IPFrame->FrameHeader.SrcMAC[0] == nowAdapter.MACAddr[0]
        && IPFrame->FrameHeader.SrcMAC[1] == nowAdapter.MACAddr[1]
        && IPFrame->FrameHeader.SrcMAC[2] == nowAdapter.MACAddr[2]
        && IPFrame->FrameHeader.SrcMAC[3] == nowAdapter.MACAddr[3]
        && IPFrame->FrameHeader.SrcMAC[4] == nowAdapter.MACAddr[4]
        && IPFrame->FrameHeader.SrcMAC[5] == nowAdapter.MACAddr[5]
        )
    return;
    // 超时的进行丢弃
    if (IPFrame->IPHeader.TTL <= 0) {// 超时
        return;
    }
    // 数据报校验
    IPStruct* IpHeader = &(IPFrame->IPHeader);
    if (checkIPHeader((char*)IpHeader) == 0) {
        // 校验出错
        return;
    }
    // 如果当前路由表中存在该网络路由路径，直接投递
    DWORD NextHop;      // 经过路由选择算法得到的下一站目的IP地址
    int IfNo;           // 下一跳的接口序号
    if ((NextHop = RouteSearch(IfNo, (DWORD)IPFrame->IPHeader.DstIP)) ==
-1) {
        // 找不到对应路由表项，抛弃数据报
        return;
    }
    // 转发数据报
    else {
        // 转发数据帧的源MAC地址变成路由器端口MAC地址，目的地址将在MAC地址映射
表中查询
        for (int i = 0; i < 6; i++) {
            IPFrame->FrameHeader.SrcMAC[i] = nowAdapter.MACAddr[i];
        }
        IPFrame->IPHeader.TTL -= 1;// TTL减一
        u_short check_buff[sizeof(IPStruct)];
        IPFrame->IPHeader.Checksum = 0;
        memset(check_buff, 0, sizeof(IPStruct));
        IPStruct* ip_header = &(IPFrame->IPHeader);
        memcpy(check_buff, ip_header, sizeof(IPStruct));
        // 计算IP头部校验和
```

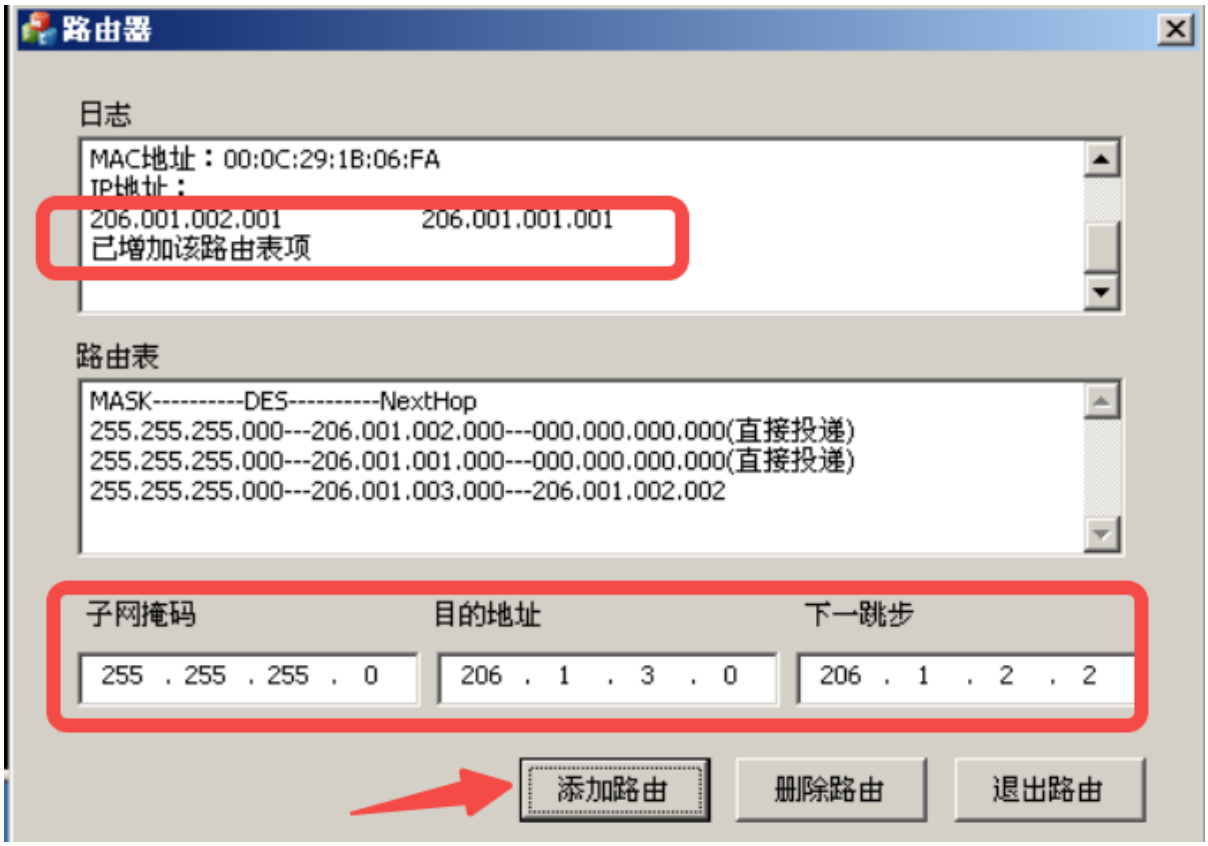
```
        IPFrame->IPHeader.Checksum = checksum(check_buff,
sizeof(IPStruct));
        // 查询MAC地址映射表，更新数据帧头目的MAC地址
        if (MACTableSearch(NextHop, IPFrame->FrameHeader.DesMAC)) {
            int res = pcap_sendpacket(nowAdapter.adhandle,
(u_char*)pkt_data, header->len);
            if (res == 0) {
                // 成功转发
            }
        }
        else {
            // MAC地址映射表不存在映射关系
            PacketCache* Packet = new PacketCache();
            Packet->ip = NextHop;
            Packet->IfNo = IfNo;
            Packet->len = header->len;
            memset(Packet->data, 0, Packet->len);
            memcpy(Packet->data, pkt_data, Packet->len);
            if (PacketQueue.size() < 65535) {// 存入缓存队列
                PacketQueue.push_back(*Packet);
                ARPRequest(nowAdapter.adhandle, nowAdapter.MACAddr,
nowAdapter.ip[0], Packet->ip);
            }
            else {
                // 存满了丢弃数据包
            }
            delete Packet;
        }
    }
}
```

实验结果

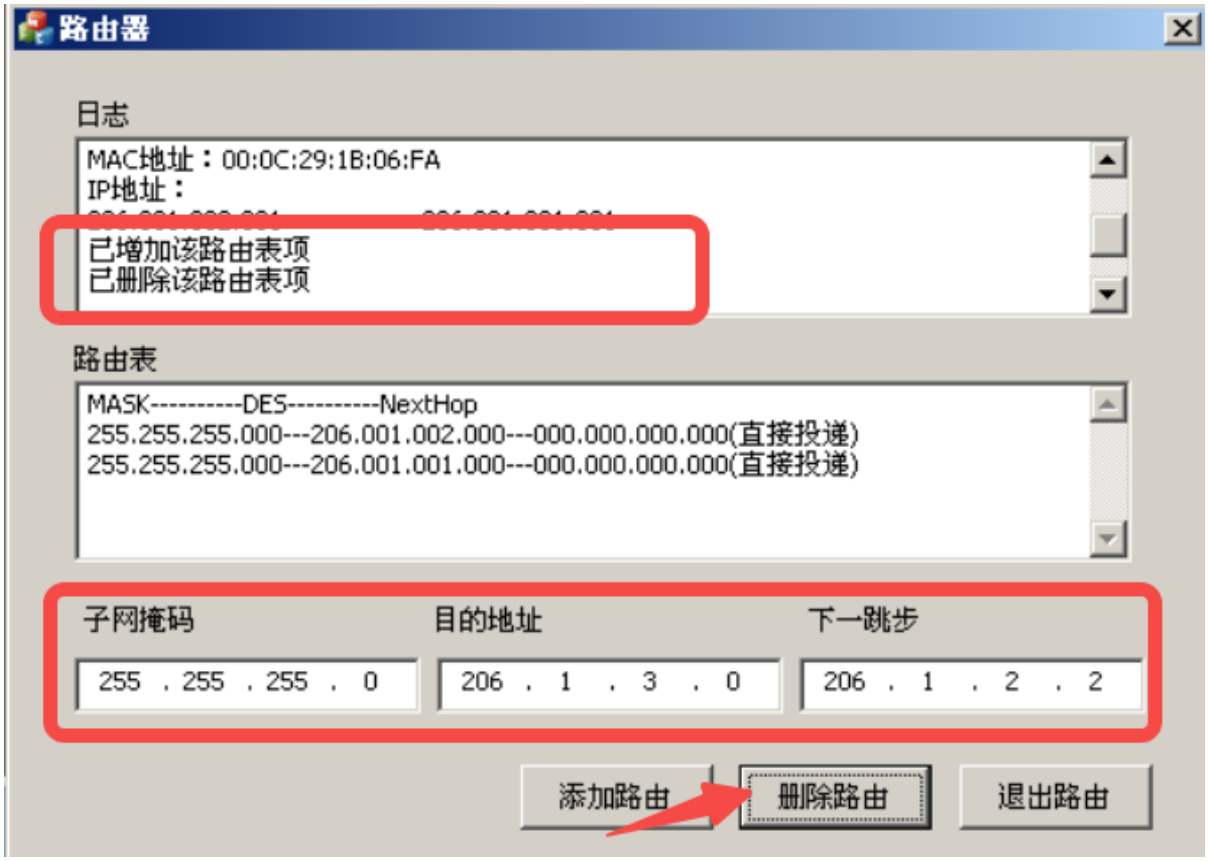
打开路由程序,看到如下界面:


路由器
日志
发送ARP请求成功!
打印网卡参数：
设备名：rpcap://\Device\NPF_{3AC00148-1BFF-47AF-8441-F1D14A619031}
MAC地址：00:0C:29:1B:06:FA
路由表
MASK----------DES----------NextHop
255.255.255.000---206.001.002.000---000.000.000.000(直接投递)
255.255.255.000---206.001.001.000---000.000.000.000(直接投递)
子网掩码
目的地址
下一跳步
0 . 0 . 0 . 0
0 . 0 . 0 . 0
0 . 0 . 0 . 0
添加路由
删除路由
退出路由

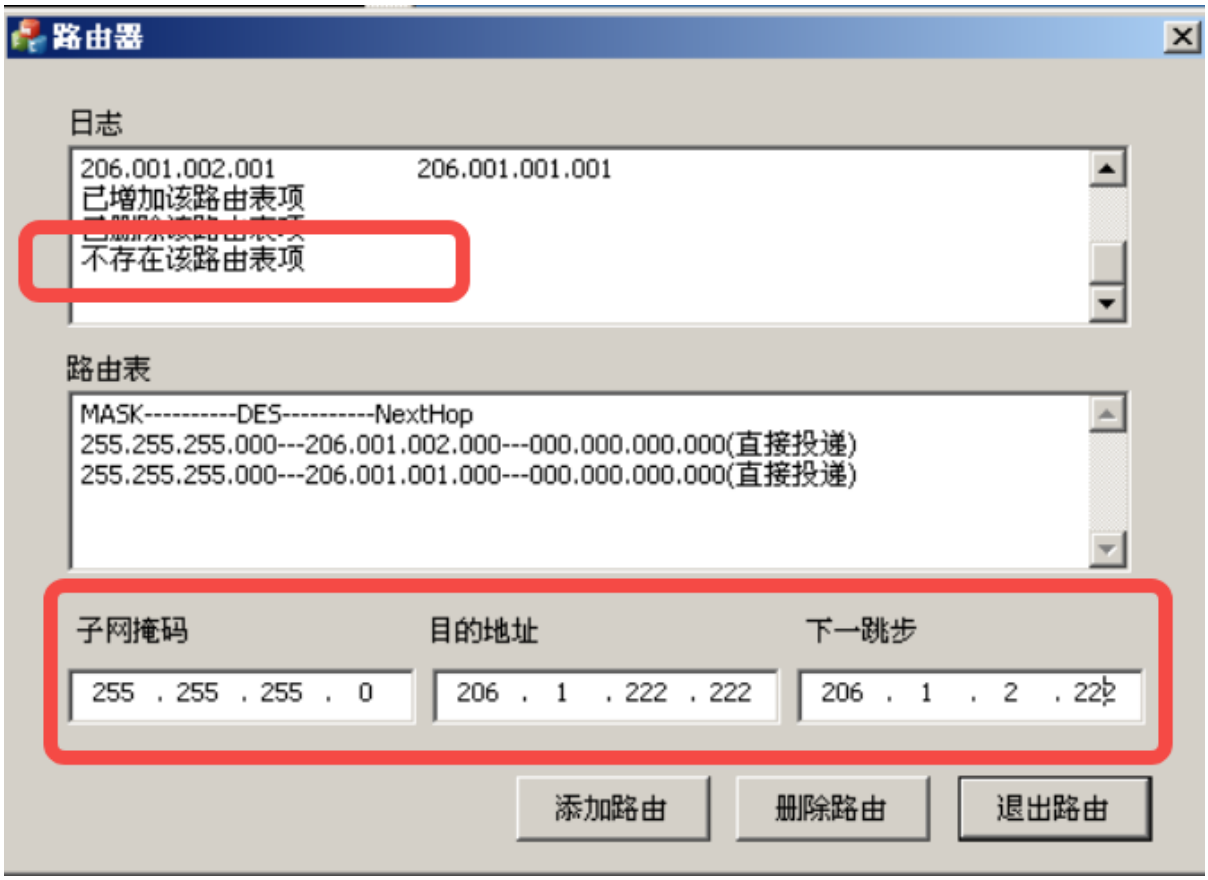
进行添加路由:


路由器
日志
MAC地址：00:0C:29:1B:06:FA
IP地址：
206.001.002.001 206.001.001.001
已增加该路由表项
路由表
MASK----------DES----------NextHop
255.255.255.000---206.001.002.000---000.000.000.000(直接投递)
255.255.255.000---206.001.001.000---000.000.000.000(直接投递)
255.255.255.000---206.001.003.000---206.001.002.002
子网掩码
目的地址
下一跳步
255 . 255 . 255 . 0
206 . 1 . 3 . 0
206 . 1 . 2 . 2
添加路由
删除路由
退出路由

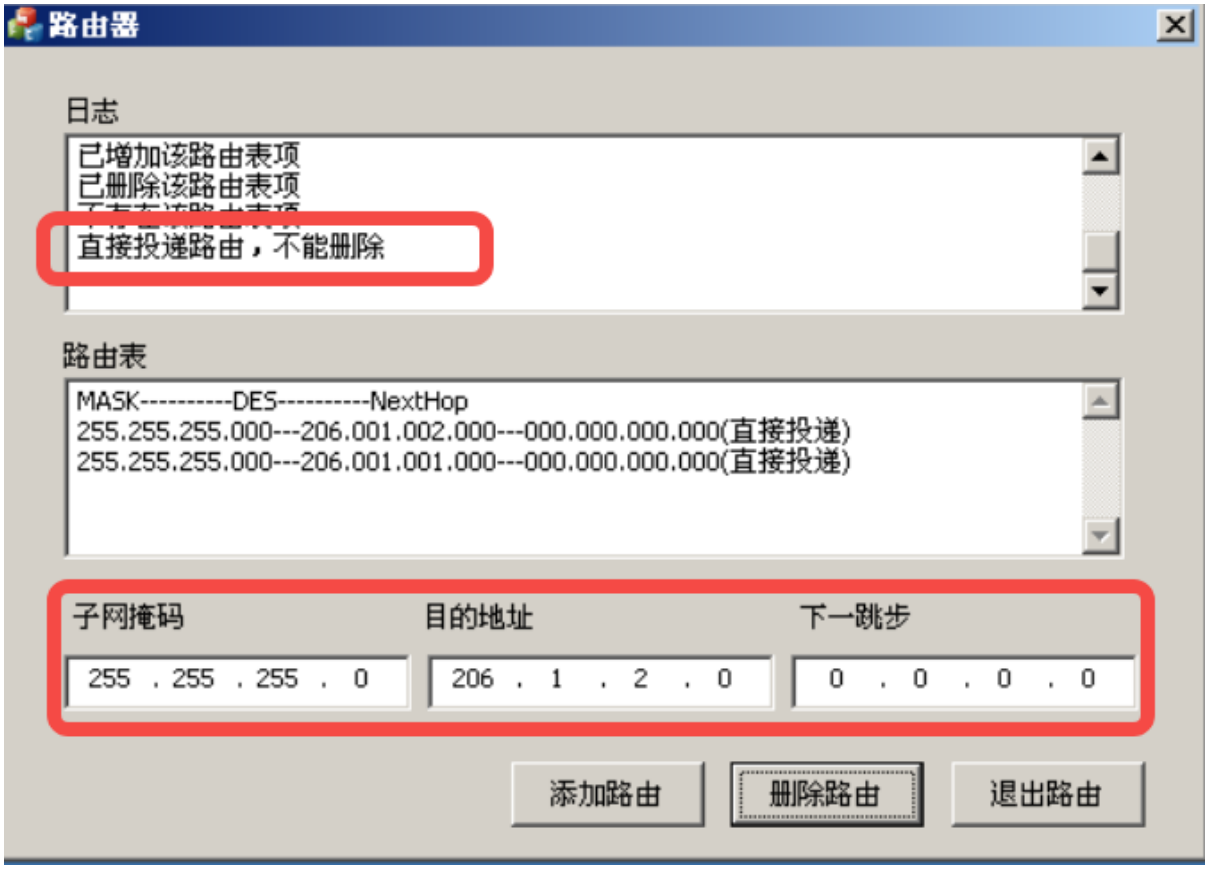
删除路由:



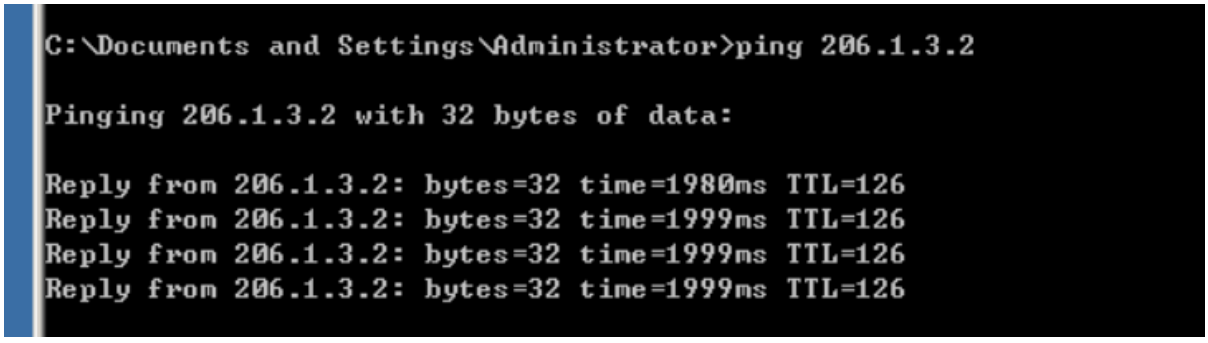
删除不存在的路由:



删除默认路由:



添加路由，A主机ping B主机的结果如下：



日志记录如下：


log.txt - 记事本
收到IP数据报：206.001.001.002->206.001.003.002
缺少目的MAC地址，已存入缓冲区：
206.001.001.002->206.001.003.002 00:0C:29:1B:06:FA->xx:xx:xx:xx:xx:xx已发送ARP请求:206.001.002.002
1. 存入缓存区
发送ARP请求成功！
2. 发送ARP请求报文
收到ARP响应数据报：206.001.002.002--00:0C:29:DD:A2:4A
该映射关系已保存。
3. 收到ARP应答报文
转发缓存区中目的地址是该MAC地址的IP数据包,转发IP数据包：
206.001.001.002->206.001.002.002 A3:BA:72:70:63:61->C9:E8:B1:B8:C3:FB
4. 转发缓存区IP数据报
收到IP数据报：206.001.003.002->206.001.001.002
缺少目的MAC地址，已存入缓冲区：
206.001.003.002->206.001.001.002 00:0C:29:1B:06:FA->xx:xx:xx:xx:xx:xx已发送ARP请求:206.001.001.002
发送ARP请求成功！
收到ARP响应数据报：206.001.001.002--00:0C:29:92:D6:06
该映射关系已保存。
上面的同样的步骤
转发缓存区中目的地址是该MAC地址的IP数据包,转发IP数据包：
206.001.003.002->206.001.001.002 36:2E:30:30:31:2E->33:01:08:00:32:30
收到IP数据报：206.001.001.002->206.001.003.002
IP数据报转发：206.001.001.002->206.001.002.002
00:0C:29:1B:06:FA->00:0C:29:DD:A2:4A
收到IP数据报：206.001.003.002->206.001.001.002
IP数据报转发：206.001.003.002->206.001.001.002
00:0C:29:1B:06:FA->00:0C:29:92:D6:06
收到IP数据报：206.001.001.002->206.001.003.002
IP数据报转发：206.001.001.002->206.001.002.002
00:0C:29:1B:06:FA->00:0C:29:DD:A2:4A
转发IP数据报过程
收到IP数据报：206.001.003.002->206.001.001.002
IP数据报转发：206.001.003.002->206.001.001.002
00:0C:29:1B:06:FA->00:0C:29:92:D6:06
收到IP数据报：206.001.001.002->206.001.003.002
IP数据报转发：206.001.001.002->206.001.002.002
00:0C:29:1B:06:FA->00:0C:29:DD:A2:4A
收到IP数据报：206.001.003.002->206.001.001.002
IP数据报转发：206.001.003.002->206.001.001.002
00:0C:29:1B:06:FA->00:0C:29:92:D6:06