

# 网络技术与应用课程第 5 次实验报告

## 实验名称：简单路由器程序的设计

学号：1711409 姓名：张嘉尧 班次：周二上午

### 一、实验目的和要求

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 数据包的获取、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，显示数据包获取和转发过程。
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。

### 二、实验过程

#### （一）程序工作原理和执行过程

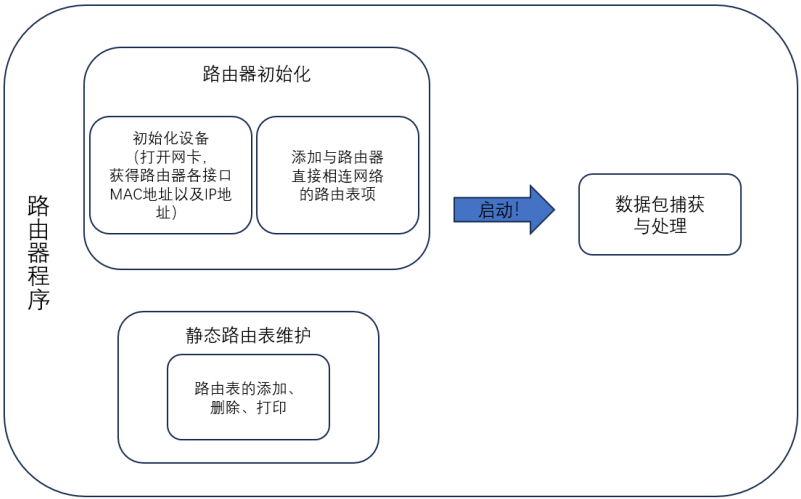
利用 Npcap 编写一个简单的路由器程序，在前两次抓包以及 ARP 程序的基础上，实现以下功能：

- 静态路由表维护。程序应该提供静态路由的添加、修改、打印和删除功能。
- IP 数据包处理。IP 数据包处理包括 IP 数据包的接收、IP 数据包的选路和 IP 数据包的发送等工作。
- ARP 请求与解析。在将一个 IP 数据包发送到下一跳之前，路由处理程序需要获得下一跳的物理 MAC 地址，因此路由器需要有 ARP 请求与解析功能。
- 处理 IP 数据包的 TTL 值。IP 数据包中的 TTL 控制数据包在网络中的停留时间，因此数据包经过时，路由器程序需要判断 TTL 的值，抛弃 TTL 小于等于 0 的数据包，并将需要转发的数据包的 TTL 减 1。
- 重新计算 IP 数据包的头部校验和。因为路由器程序需要进行 TTL 处理，因此需要送出的 IP 数据包与接收时的 IP 数据包头部会存在差异，因此需要重新计算 IP 数据包的头部校验和。
- 生成和处理 ICMP 报文。ICMP 报文的生成和处理功能应该是路由器程序的

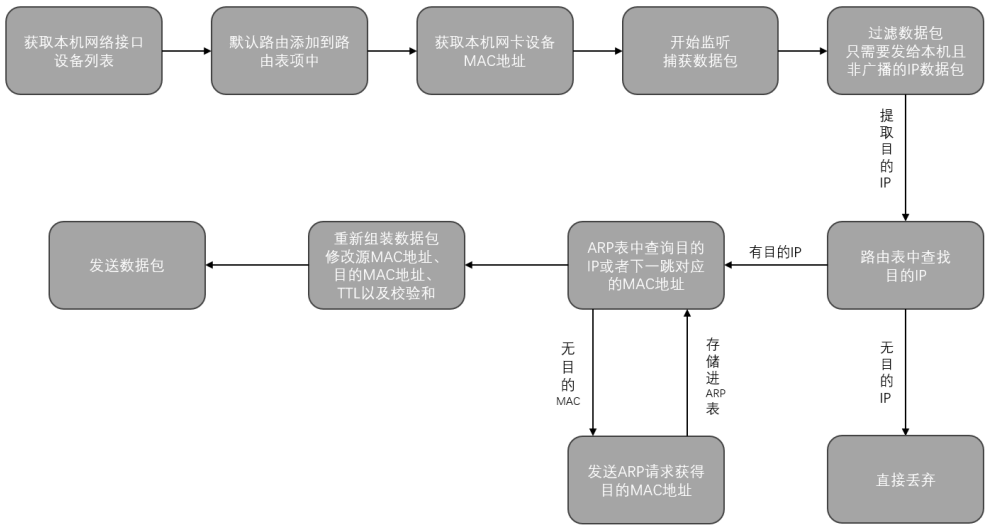
一部分。在抛弃掉校验和错误的 IP 数据包时，形成和发送 ICMP 差错报文，本程序只要求发送目的不可达报文和 ICMP 超时报文。

按照简单路由程序的实现要求，本程序可以分为路由器初始化、静态路由表维护以及数据包捕获与处理三大模块。

- 初始化模块负责初始化设备，包括打开网卡、获得路由器各接口的 MAC 地址以及 IP 地址，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。
- 静态路由表维护模块负责完成路由表的添加、删除以及打印。
- 数据包捕获与处理模块负责捕获流经本路由器的数据包并按照路由协议进行处理。



而路由器程序实现路由转发的大致流程如下图所示：



1. 路由器程序获取本机网络接口的设备信息，对其进行打印并存储对应的 IP 地址和子网掩码。
2. 路由器程序对路由表进行初始化，并添加默认路由表项。
3. 初始化 ARP 表，获取本机网卡的 MAC 地址并添加到 ARP 表中。
4. 完成初始化工作后，路由器程序进入监听状态捕获数据包并对其进行筛选。
5. 路由器程序筛选出目的 MAC 地址为本机且非广播的 IP 数据包进行解析，提取其中的目的 IP 地址。
6. 路由器程序在路由表中查找是否有目的 IP 地址，如果没有，则直接丢弃数据包；如果有目的 IP 地址，则到 ARP 表中查询目的主机或者下一跳对应的 MAC 地址。
7. 如果 ARP 表中没有查到对应的 MAC 地址，路由器程序则会发送 ARP 报文获取对应 IP 的 MAC 地址，并存储到 ARP 表中。
8. 获取到对应的 MAC 地址后，路由器程序重新组装 IP 数据包，并修改其中源 MAC 地址、目的 MAC 地址、TTL 以及校验和。
9. 路由器程序将重新组装的 IP 数据包发送出去，完成路由转发。

## （二）协议设计

由于路由器程序需要处理以太网帧、IP 数据包、ARP 数据包以及 ICMP 数据包，则需要定义相关数据包头部的数据结构：

```
#pragma pack (1)//进入字节对齐方式
//帧首部 14字节
typedef struct FrameHeader_t {
    BYTE DesMAC[6]; // 目的地址
    BYTE SrcMAC[6]; //源地址
    WORD FrameType; //帧类型
}FrameHeader_t;
//ARP帧 28字节
typedef struct ARPFrame_t {
    FrameHeader_t FrameHeader;//以太网帧头
    WORD HardwareType; //硬件类型
    WORD ProtocolType; //协议类型
    BYTE HLen; //地址长度MAC
    BYTE PLen; //地址长度IP
    WORD Operation; //协议动作
```

```

    BYTE SendHa[6];          //源地址MAC
    DWORD SendIP;           //源地址IP
    BYTE RecvHa[6];          //目的地址MAC
    DWORD RecvIP;           //目的地址IP
} ARPFrame_t;
//IP首部
typedef struct IPHeader_t {
    BYTE Ver_HLen;           //版本+头部长度的
    BYTE TOS;                //服务类型
    WORD TotalLen;           //总长度字段
    WORD ID;                 //标识
    WORD Flag_Segment;       //标志+片偏移
    BYTE TTL;               //生命周期
    BYTE Protocol;          //协议
    WORD Checksum;           //校验和
    ULONG SrcIP;            //源IP
    ULONG DstIP;            //目的IP
} IPHeader_t;
//包含帧首部和IP首部的数据包
typedef struct Data_t {
    FrameHeader_t FrameHeader; //帧首部
    IPHeader_t IPHeader;       //IP首部
} Data_t;
//包含帧首部和IP首部的ICMP包
typedef struct ICMP {
    FrameHeader_t FrameHeader; //帧首部
    IPHeader_t IPHeader;       //IP首部
    char buf[40];

} ICMP_t;

#pragma pack()              //恢复缺省对齐方式

//arp条目
class ArpEntry
{
public:
    DWORD ip; //IP
    BYTE mac[6]; //MAC
    void printEntry() {
        unsigned char* pIP = (unsigned char*)&ip;
        printf("IP地址: %u.%u.%u.%u \t MAC地址: ", *pIP, *(pIP + 1), *(pIP + 2), *(pIP
+ 3));
        printf("%02x-%02x-%02x-%02x-%02x-%02x\n", mac[0], mac[1], mac[2], mac[3],
mac[4], mac[5]);
    }
}

```

```

};
//arp表
class ArpTable
{
public:
    ArpEntry arp_table[50];
    ArpTable() {
        arpNum = 0;
    };
    int arpNum = 0;
    void insert(DWORD ip, BYTE mac[6]) {
        for (int i = 0; i < arpNum; i++) {
            if (arp_table[i].ip == ip) {
                CopyMAC(mac, arp_table[i].mac);
                printf("表项已经存在, 更新!\n");
                return;
            }
        }
        arp_table[arpNum].ip = ip;
        CopyMAC(mac, arp_table[arpNum].mac);
        arpNum++;
        printf("成功插入ARP表项!\n");
    }
    int lookup(DWORD ip, BYTE mac[6]) {
        unsigned char* pIP = (unsigned char*)&ip;
        printf("在ARP表中查询IP : %u.%u.%u.%u\n", *pIP, *(pIP + 1), *(pIP + 2), *(pIP
+ 3));
        for (int i = 0; i < arpNum; i++) {
            pIP = (unsigned char*)&arp_table[i].ip;
            if (ip == arp_table[i].ip) {
                CopyMAC(arp_table[i].mac, mac);
                return i;
            }
        }
        printf("ARP表中无该项!\n");
        return -1;
    }
    void printTable() {
        printf("\n-----ARP表-----\n");
        for (int i = 0; i < arpNum; i++) {
            arp_table[i].printEntry();
        }
        printf("-----\n\n");
    }
}

```

```
};
```

```
RouteTable routeTable;
```

```
ArpTable arpTable;
```

## 1. 路由器初始化模块

路由器初始化模块主要工作是获得路由器各接口的 MAC 地址和各接口的所有 IP 地址，建立初始的路由表以及启动相应的数据包捕获与处理模块。

路由器初始化模块需要初始化各个接口信息和路由表，则要先定义存储所有接口设备以及 IP 地址、MAC 地址、接口数量的全局变量：

```
//全局变量
```

```
pcap_if_t* alldevs;
```

```
//pcap_t* adhandle;          //捕捉实例,是pcap_open返回的对象
```

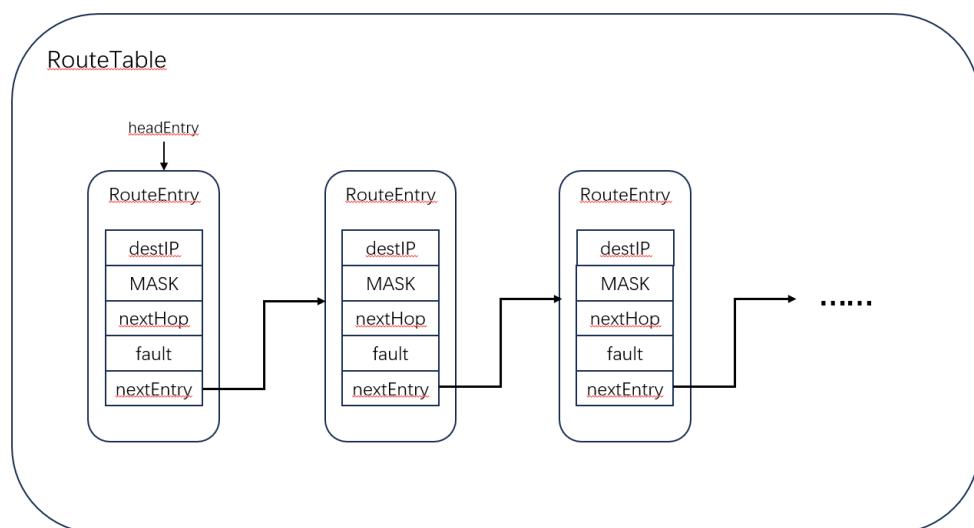
```
char ipList[10][32];        //存储网卡设备IP地址
```

```
char maskList[10][32];
```

```
int dev_nums = 0;          //适配器计数变量
```

```
BYTE MyMAC[6];             //本机设备 MAC 地址
```

路由器程序中，我们采用链式结构来定义路由表的数据结构，结构大致如下：



路由表项数据结构，包含路由表项信息的打印：

```
class RouteEntry
```

```
{
```

```
public:
```

```
    DWORD destIP; //目的地址
```

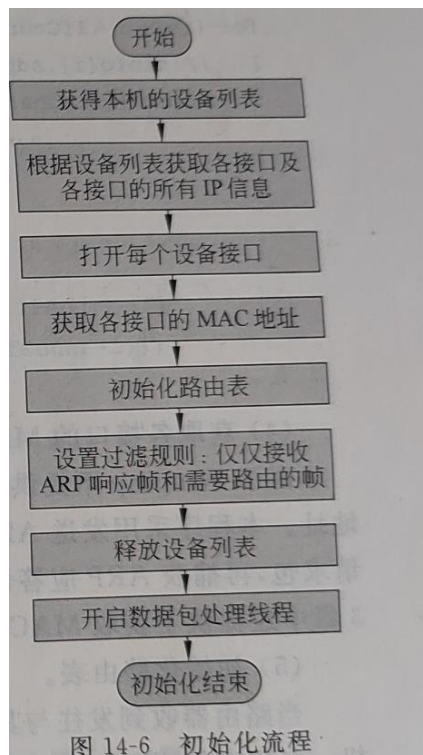
```
    DWORD mask;    //子网掩码
```

```

DWORD nextHop;    //下一跳
bool fault;       //是否为默认路由
RouteEntry* nextEntry;//链式存储
RouteEntry() {
    memset(this, 0, sizeof(*this)); //初始化为全0
    nextEntry = NULL;
}
void printEntry() //打印表项内容，打印出掩码、目的网络和下一跳IP、类型（是否是直接投
递）
{
    unsigned char* pIP = (unsigned char*)&destIP;
    printf("目的IP : %u.%u.%u.%u\t", *pIP, *(pIP + 1), *(pIP + 2), *(pIP + 3));
    pIP = (unsigned char*)&mask;
    printf("子网掩码 : %u.%u.%u.%u\t", *pIP, *(pIP + 1), *(pIP + 2), *(pIP + 3));
    pIP = (unsigned char*)&nextHop;
    printf("下一跳: %u.%u.%u.%u\t", *pIP, *(pIP + 1), *(pIP + 2), *(pIP + 3));
    if (fault) {
        printf("直接投递\n");
    }
    else {
        printf("非直接投递\n");
    }
}
};

```

初始化模块的主要流程：



获取本机所有网卡设备信息列表 Npcap 的 pcap\_findalldevs 函数：

```
char errbuf[PCAP_ERRBUF_SIZE];    //错误缓冲区, 大小为256
pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &alldevs, errbuf);
```

获取路由器各接口信息以及各接口的所有 IP 地址、子网掩码并存储到

ipList 和 maskList 中：

```
void DevsInfo(pcap_if_t* alldevs) {
    for (pcap_if_t* d = alldevs; d != nullptr; d = d->next) //显示接口列表
    {
        //获取该网络接口设备的ip地址信息
        for (pcap_addr* a = d->addresses; a != nullptr; a = a->next)
        {
            if (((struct sockaddr_in*)a->addr)->sin_family == AF_INET && a->addr)
            { //打印ip地址
                //打印相关信息
                //inet_ntoa将ip地址转成字符串格式
                printf("%d\n", dev_nums);
                printf("%s\t\t%s\n%s\t\t%s\n", "网卡名:", d->name, "描述信息:",
d->description);
                printf("%s\t\t%s\n", "IP地址: ", inet_ntoa(((struct
sockaddr_in*)a->addr)->sin_addr));
                printf("%s\t\t%s\n", "子网掩码: ", inet_ntoa(((struct
sockaddr_in*)a->netmask)->sin_addr));
                printf("-----\n");
                strcpy(ipList[dev_nums], inet_ntoa(((struct
sockaddr_in*)a->addr)->sin_addr));
                strcpy(maskList[dev_nums++], inet_ntoa(((struct
sockaddr_in*)a->netmask)->sin_addr));
            }
        }
    }
}
```

路由表的数据结构及初始化：

```
class RouteTable
{
public:
    RouteEntry* head;
    int routeNum; //条数
    //初始化, 添加直接连接的网络
    void initRouteTable() {
        head = NULL;
        routeNum = 0;
    }
}
```



```

        for (int i = 0; i < 2; i++) {
            RouteEntry* newEntry = new RouteEntry();
            newEntry->destIP = (inet_addr(ipList[i])) & (inet_addr(maskList[i])); //本
机网卡的ip和掩码进行按位与即为所在网络
            newEntry->mask = inet_addr(maskList[i]);
            newEntry->fault = 1; //0表示直接投递的网络，不可删除
            this->addEntry(newEntry); //添加表项
        }
    }
}

```

获取本机 MAC 地址：

//获取本机MAC地址

```

void getLocalMAC(pcap_if_t* device) {
    int index = 0;
    for (pcap_addr* a = device->addresses; a != nullptr; a = a->next) {
        if (((struct sockaddr_in*)a->addr)->sin_family == AF_INET && a->addr) {
            //打印本机IP地址
            printf("%s\t%s\n", "本地IP地址:", inet_ntoa(((struct
sockaddr_in*)a->addr)->sin_addr));
            //在当前网卡上伪造一个包
            ARPFrame_t ARPFrame = MakeARP();
            ARPFrame.SendIP = inet_addr("0.0.0.0");
            ARPFrame.RecvIP = inet_addr(ipList[index]);
            //打开该网卡的网络接口
            pcap_t* adhandle = pcap_open(device->name, 655340,
PCAP_OPENFLAG_PROMISCUOUS, 1000, 0, 0);
            if (adhandle == NULL) { printf("打开接口失败!\n"); return; }
            //发送ARP包
            int res = pcap_sendpacket(adhandle, (u_char*)&ARPFrame,
sizeof(ARPFrame_t));
            //捕获数据包
            ARPFrame_t* RecPacket;
            struct pcap_pkthdr* pkt_header;
            const u_char* pkt_data;
            while ((res = pcap_next_ex(adhandle, &pkt_header, &pkt_data)) >= 0) {
                RecPacket = (ARPFrame_t*)pkt_data;
                if (!CompareMAC(RecPacket->FrameHeader.SrcMAC,
ARPFrame.FrameHeader.SrcMAC)
                    && CompareMAC(RecPacket->FrameHeader.DesMAC,
ARPFrame.FrameHeader.SrcMAC)
                    && RecPacket->SendIP == ARPFrame.RecvIP
                ) { //过滤完毕
                    CopyMAC(RecPacket->FrameHeader.SrcMAC, MyMAC); //存储本机MAC
                    //打印获取的MAC地址
                }
            }
        }
    }
}

```



```

        cur->nextEntry = newEntry;
        routeNum++;
        return;
    }
}

```

在路由表结构中加入删除路由表项的函数：

```

bool deleteEntry(DWORD IP) {
    if (IP == head->destIP && !head->fault) {
        delete head;
        head = NULL;
        return true;
    }
    RouteEntry* cur = head;
    while (cur->nextEntry) {
        if (cur->nextEntry->destIP == IP) {
            RouteEntry* temp = cur->nextEntry;
            if (temp->fault) {
                printf("删除错误路由表项失败!\n");
                return false;
            }
            cur->nextEntry = temp->nextEntry;
            delete temp;
            printf("成功删除!\n");
            return true;
        }
        cur = cur->nextEntry;
    }
    return false;
}

```

同时我们在路由表结构中加入打印路由表的函数：

```

//路由表的打印
void printTable() {
    printf("\n-----路由表-----\n");
    RouteEntry* cur = head;
    while (cur) {
        cur->printEntry();
        cur = cur->nextEntry;
    }
    printf("\n\n");
    return;
}

```

为了方便查找路由以及日志的输出，在路由表结构中加入查找的函数：

//查找，最长前缀，返回下一跳的ip

```
DWORD lookup(DWORD ip) {
    RouteEntry* cur = head;
    while (cur != NULL) {
        if ((cur->mask & ip) == (cur->mask & cur->destIP)) {
            if (cur->fault) {
                return 0;
            }
            return cur->nextHop;
        }
        cur = cur->nextEntry;
    }
    return -1;
}
```

### 3. 数据包捕获与处理模块

数据包捕获与处理模块主要有数据包的捕获、对 IP 数据包的处理以及对 ARP 数据包的处理 3 部分。

数据包捕获的代码实现：

//捕获IP数据报

```
ICMP CapturePacket(pcap_if_t* device) {
    pcap_t* adhandle = pcap_open(device->name, 655340, PCAP_OPENFLAG_PROMISCUOUS, 1000,
0, 0);
    pcap_pkthdr* pkt_header;
    const u_char* pkt_data;
    printf("开始监听...\n");
    while (1) {
        int res = pcap_next_ex(adhandle, &pkt_header, &pkt_data);
        if (res > 0) {
            FrameHeader_t* header = (FrameHeader_t*)pkt_data;
            if (CompareMAC(header->DesMAC, MyMAC)) { //发给本机
                if (ntohs(header->FrameType) == 0x800) { //IP格式的数据报
                    Data_t* data = (Data_t*)pkt_data;
                    if (!checkChecksum(data)) {
                        printf("校验错误!\n");
                        continue;
                    }
                }
                //打印日志
                outputLog(data, 1);
                //提取IP数据报中的目的IP
                DWORD DstIP = data->IPHeader.DstIP;
            }
        }
    }
}
```

```

unsigned char* pIP = (unsigned char*)&DstIP;
//到路由表中查找
DWORD routeFind = routeTable.lookup(DstIP);
printf("找到路由地址: %u.%u.%u.%u \n", *pIP, *(pIP + 1), *(pIP +
2), *(pIP + 3));

pIP = (unsigned char*)&routeFind;
printf("下一跳: %u.%u.%u.%u \n", *pIP, *(pIP + 1), *(pIP + 2),
*(pIP + 3));

if (routeFind == -1) { //没有该路由条目
    printf("没有该路由条目\n");
    continue;
}

printf("成功找到路由表\n");
//检查是否为广播消息
BYTE broadcast[6] = "fffff";
if (!CompareMAC(data->FrameHeader.DesMAC, broadcast)
    && !CompareMAC(data->FrameHeader.SrcMAC, broadcast)
    )
{
    //ICMP报文包含IP数据包报头和其它内容
    ICMP_t* sendPacket_t = (ICMP_t*)pkt_data;
    ICMP_t sendPacket = *sendPacket_t;
    BYTE mac[6];
    if (routeFind == 0) {
        //默认路由项 直接投递 到ARP表中查询DstIP
        if (arpTable.lookup(DstIP, mac) == -1) {
            //ARP表中无该条目 调用getRemoteMAC函数获取目的IP的
MAC地址

            printf("获取远端MAC地址!\n");
            getRemoteMAC(device, DstIP);
            //打印更新后的ARP表
            arpTable.printTable();
            if (arpTable.lookup(DstIP, mac) == -1) {
                //仍旧未找到
                printf("无法获得远端MAC地址!");
                continue;
            }
        }
        printf("目的MAC地址:");
    }
    else { //非默认路由 获取下一跳的IP地址 到ARP表中查询nextHop的

```

MAC地址

```
        if (arpTable.lookup(routeFind, mac) == -1) {
            printf("获取远端MAC地址!\n");
            getRemoteMAC(device, routeFind);
            arpTable.printTable();
            if (arpTable.lookup(routeFind, mac) == -1) {
                printf("无法获得远端MAC地址!");
                continue;
            }
        }
        printf("下一跳MAC地址:");

    }
    printMAC(mac);
    printf("\n");
    //更新MAC地址并转发IP数据报
    routeForward(adhandle, sendPacket, mac);
}
}
}
}
}
```

IP 数据包处理的代码实现即路由转发函数:

```
void routeForward(pcap_t* adhandle, ICMP_t data, BYTE DstMAC[]) {
    Data_t* sendPacket = (Data_t*)&data;
    memcpy(sendPacket->FrameHeader.SrcMAC, sendPacket->FrameHeader.DesMAC, 6); //源MAC
    //为本机MAC
    memcpy(sendPacket->FrameHeader.DesMAC, DstMAC, 6); //目的MAC为下一跳MAC
    sendPacket->IPHeader.TTL -= 1; //更新TTL
    if (sendPacket->IPHeader.TTL < 0) {
        printf("TTL失效!\n");
        return;
    }
    setChecksum(sendPacket); //重新设置校验和
    int res = pcap_sendpacket(adhandle, (const u_char*)sendPacket, 74); //发送数据报
    if (res == 0) {
        //发送成功 打印日志
        outputLog(sendPacket, 0);
        printf("发送数据报到: ");
        printMAC(DstMAC);
        printf("\n");
    }
}
```

其中捕获 IP 数据包时包括的 MAC 处理函数和首部校验和检验函数：

//MAC操作

```
bool CompareMAC(BYTE* MAC_1, BYTE* MAC_2) {  
    for (int i = 0; i < 6; i++) {  
        if (MAC_1[i] != MAC_2[i]) {  
            return false;  
        }  
    }  
    return true;  
}
```

```
void CopyMAC(BYTE* MAC_1, BYTE* MAC_2) {  
    for (int i = 0; i < 6; i++) {  
        MAC_2[i] = MAC_1[i];  
    }  
}
```

//设置校验和

```
void setChecksum(Data_t* dataPacket) {  
    dataPacket->IPHeader.Checksum = 0;  
    unsigned int sum = 0;  
    WORD* t = (WORD*)&dataPacket->IPHeader;//16bit一组  
    for (int i = 0; i < sizeof(IPHeader_t) / 2; i++) {  
        sum += t[i];  
        while (sum >= 0x10000) {  
            int s = sum >> 16;  
            sum -= 0x10000;  
            sum += s;  
        }  
    }  
    dataPacket->IPHeader.Checksum = ~sum;//取反  
}
```

//校验

```
bool checkChecksum(Data_t* dataPacket) {  
    unsigned int sum = 0;  
    WORD* t = (WORD*)&dataPacket->IPHeader;  
    for (int i = 0; i < sizeof(IPHeader_t) / 2; i++) {  
        sum += t[i];  
        while (sum >= 0x10000) {  
            int s = sum >> 16;  
            sum -= 0x10000;  
            sum += s;  
        }  
    }  
    if (sum == 65535)  
        return 1;//校验正确
```

```

return 0;
}

```

为了获取目的 MAC 地址而需要构造 ARP 包的实现代码:

//构造ARP包

```

ARPFrame_t MakeARP() {
    ARPFrame_t ARPFrame;
    for (int i = 0; i < 6; i++)
        ARPFrame.FrameHeader.DesMAC[i] = 0xff; //表示广播
    //将ARPFrame.FrameHeader.SrcMAC设置为本机网卡的MAC地址
    for (int i = 0; i < 6; i++)
        ARPFrame.FrameHeader.SrcMAC[i] = 0x0f;
    //CopyMAC(ARPFrame.FrameHeader.SrcMAC, MyMAC);
    ARPFrame.FrameHeader.FrameType = htons(0x806); //帧类型为ARP
    ARPFrame.HardwareType = htons(0x0001); //硬件类型为以太网
    ARPFrame.ProtocolType = htons(0x0800); //协议类型为IP
    ARPFrame.HLen = 6; //硬件地址长度为6
    ARPFrame.PLen = 4; //协议地址长为4
    ARPFrame.Operation = htons(0x0001); //操作为ARP请求

    //将ARPFrame.SendHa设置为本机网卡的MAC地址
    for (int i = 0; i < 6; i++)
        ARPFrame.SendHa[i] = 0x0f;
    //将ARPFrame.SendIP设置为本机网卡上绑定的IP地址

    //将ARPFrame.RecvHa设置为0
    for (int i = 0; i < 6; i++)
        ARPFrame.RecvHa[i] = 0; //表示目的地址未知
    //将ARPFrame.RecvIP设置为请求的IP地址
    return ARPFrame;
}

```

获取目的 MAC 地址代码实现:

```

void getRemoteMAC(pcap_if_t* device, DWORD DstIP) {
    int index = 0;
    for (pcap_addr* a = device->addresses; a != nullptr; a = a->next) {
        if (((struct sockaddr_in*)a->addr)->sin_family == AF_INET && a->addr) {
            DWORD devIP = inet_addr(inet_ntoa(((struct
sockaddr_in*)a->addr)->sin_addr));
            //使用同网段的IP来获取远端MAC地址
            if ((devIP & inet_addr(maskList[index])) != (DstIP &
inet_addr(maskList[index]))) {
                continue;
            }
        }
    }
}

```



```

//打印远端IP和本机IP地址
unsigned char* pIP = (unsigned char*)&DstIP;
printf("远端IP地址: %u.%u.%u.%u \n", *pIP, *(pIP + 1), *(pIP + 2), *(pIP +
3));

pIP = (unsigned char*)&devIP;
printf("本地IP地址: %u.%u.%u.%u \n", *pIP, *(pIP + 1), *(pIP + 2), *(pIP +
3));

//在当前网卡上伪造一个包
ARPFrame_t ARPFrame = MakeARP();
//更新伪造的ARP包 将本机的MAC填入
CopyMAC(MyMAC, ARPFrame.FrameHeader.SrcMAC);
CopyMAC(MyMAC, ARPFrame.SendHa);
ARPFrame.SendIP = inet_addr(inet_ntoa(((struct
sockaddr_in*)a->addr)->sin_addr));
ARPFrame.RecvIP = DstIP;
//打开该网卡的网络接口
pcap_t* adhandle = pcap_open(device->name, 655340,
PCAP_OPENFLAG_PROMISCUOUS, 1000, 0, 0);
if (adhandle == NULL) { printf("打开接口失败!\n"); return; }
//发送ARP包
int res = pcap_sendpacket(adhandle, (u_char*)&ARPFrame,
sizeof(ARPFrame_t));
//捕获数据包
ARPFrame_t* RecPacket;
struct pcap_pkthdr* pkt_header;
const u_char* pkt_data;
while ((res = pcap_next_ex(adhandle, &pkt_header, &pkt_data)) >= 0) {
    RecPacket = (ARPFrame_t*)pkt_data;
    if (!CompareMAC(RecPacket->FrameHeader.SrcMAC,
ARPFrame.FrameHeader.SrcMAC)
        && CompareMAC(RecPacket->FrameHeader.DesMAC,
ARPFrame.FrameHeader.SrcMAC)
        && RecPacket->SendIP == ARPFrame.RecvIP
    ) { //过滤成功
        //插入ARP表
        arpTable.insert(DstIP, RecPacket->FrameHeader.SrcMAC);
        break;
    }
}
}
index++;
//}
}
}

```

#### 4. 路由器的工作日志

因为本次实验需要给出路由器的工作日志，显示数据报获取和转发过程，则设计日志输出的代码如下：

//日志输出

```
void outputLog(Data_t* dataPacket, bool receive) {
    FILE* fp = fopen("myRouter.log", "a");
    if (fp == NULL) {
        printf("打开文件失败! \n");
        return;
    }
    DWORD nexthop = routeTable.lookup((DWORD) dataPacket->IPHeader.DstIP);
    unsigned char* SrcIP = (unsigned char*)&dataPacket->IPHeader.SrcIP;
    unsigned char* DstIP = (unsigned char*)&dataPacket->IPHeader.DstIP;
    unsigned char* nexthop = (unsigned char*)&nexthop;
    BYTE* SrcMAC = dataPacket->FrameHeader.SrcMAC;
    BYTE* DstMAC = dataPacket->FrameHeader.DesMAC;

    if (receive) {
        fprintf(fp, "[receive IP] 源IP地址:%u.%u.%u.%u 目的IP地址:%u.%u.%u.%u 源MAC地
址:%02X-%02X-%02X-%02X-%02X-%02X 目的MAC地址:%02X-%02X-%02X-%02X-%02X-%02X\n",
            *SrcIP, *(SrcIP + 1), *(SrcIP + 2), *(SrcIP + 3),
            *DstIP, *(DstIP + 1), *(DstIP + 2), *(DstIP + 3),
            SrcMAC[0], SrcMAC[1], SrcMAC[2], SrcMAC[3], SrcMAC[4], SrcMAC[5],
            DstMAC[0], DstMAC[1], DstMAC[2], DstMAC[3], DstMAC[4], DstMAC[5]);
    }
    else {
        fprintf(fp, "[forward IP] 下一跳:%u.%u.%u.%u 源MAC地
址:%02X-%02X-%02X-%02X-%02X-%02X 目的MAC地址:%02X-%02X-%02X-%02X-%02X-%02X\n",
            *nexthop, *(nexthop + 1), *(nexthop + 2), *(nexthop + 3),
            SrcMAC[0], SrcMAC[1], SrcMAC[2], SrcMAC[3], SrcMAC[4], SrcMAC[5],
            DstMAC[0], DstMAC[1], DstMAC[2], DstMAC[3], DstMAC[4], DstMAC[5]);
    }
}
```

#### 5. 主函数实现

路由器程序最终实现的主函数代码如下：

//输入命令

```
char control[6];
char option[6];
char ipAddress[32];
char mask[32];
char nexthop[32];
```

```

printf("> 输入指令,例如:\n");
printf(">  route [option] [ipAddress] [mask] [nextHop]\n");
printf(">  arp      [option]\n");
printf(">  exit\n");
while (1) {
    printf("> ");
    scanf("%s", control);
    if (!strcmp(control, "route") || !strcmp(control, "ROUTE"))//操作[option]包括
add, delete, print, start, help
    {
        scanf("%s", option);
        if (!strcmp(option, "add") || !strcmp(option, "ADD")) {
            scanf("%s%s%s", ipAddress, mask, nextHop);
            RouteEntry* newEntry = new RouteEntry;
            newEntry->destIP = inet_addr(ipAddress);
            newEntry->mask = inet_addr(mask);
            newEntry->nextHop = inet_addr(nextHop);
            routeTable.addEntry(newEntry);
            continue;
        }
        else if (!strcmp(option, "delete") || !strcmp(option, "DELETE")) {
            scanf("%s", ipAddress);
            routeTable.deleteEntry(inet_addr(ipAddress));
            continue;
        }
        else if (!strcmp(option, "print") || !strcmp(option, "PRINT")) {
            routeTable.printTable();
            continue;
        }
        else if (!strcmp(option, "start") || !strcmp(option, "START")) {
            CapturePacket(alldevs);
            continue;
        }
        else if (!strcmp(option, "help") || !strcmp(option, "HELP")) {
            printf("例如:\n");
            printf("> route add ip mask nexthop\n");
            printf("> route delete ip\n");
            printf("> route start\n");
            printf("> route print\n");
            continue;
        }
    }
}
else if (!strcmp(control, "arp") || !strcmp(control, "ARP")) {

```

```

scanf("%s", option);
if (!strcmp(option, "show") || !strcmp(option, "SHOW")) {
    arpTable.printTable();
    continue;
}
}

else if (!strcmp(control, "exit") || !strcmp(control, "EXIT")) {
    break;
}

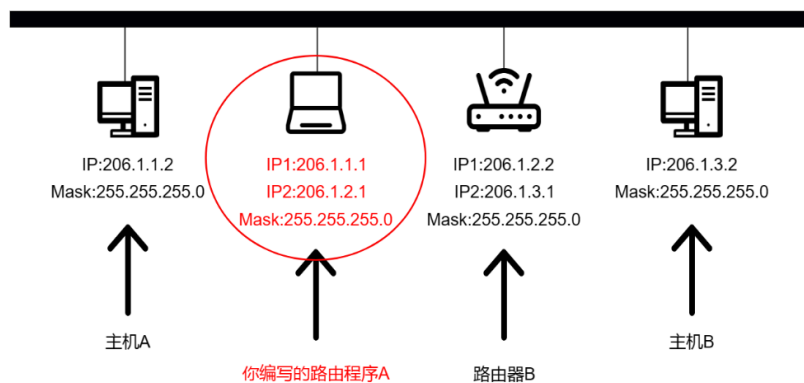
printf("错误指令! \n");
}

system("pause");

```

### (三) 程序运行界面展示

实验环境拓扑图：



路由表手动添加路由表项及启动：

```

C:\Documents and Settings\Administrator\桌面\Router.exe
本地IP地址: 206.1.2.1
本地MAC地址: 00-0c-29-ed-79-fe
成功插入ARP表项!
本地IP地址: 206.1.1.1
本地MAC地址: 00-0c-29-ed-79-fe
成功插入ARP表项!

-----ARP表-----
IP地址: 206.1.2.1      MAC地址: 00-0c-29-ed-79-fe
IP地址: 206.1.1.1      MAC地址: 00-0c-29-ed-79-fe

> 输入指令,例如:
> route [option] [ipAddress] [mask] [nextHop]
> arp [option]
> exit
> route add 206.1.3.0 255.255.255.0 206.1.2.2
> route start
开始监听...
找到路由地址: 206.1.1.2
下一跳: 0.0.0.0
成功找到路由表
在ARP表中查询IP: 206.1.1.2
ARP表中无该项!

```

主机 B 向主机 A 发送 ping 数据包:

```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Administrator>ping 206.1.1.2

Pinging 206.1.1.2 with 32 bytes of data:

Request timed out.
Request timed out.
Reply from 206.1.1.2: bytes=32 time=3523ms TTL=126
Reply from 206.1.1.2: bytes=32 time=3500ms TTL=126

Ping statistics for 206.1.1.2:
    Packets: Sent = 4, Received = 2, Lost = 2 (50% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 3500ms, Maximum = 3523ms, Average = 3511ms

C:\Documents and Settings\Administrator>
```

路由表程序:

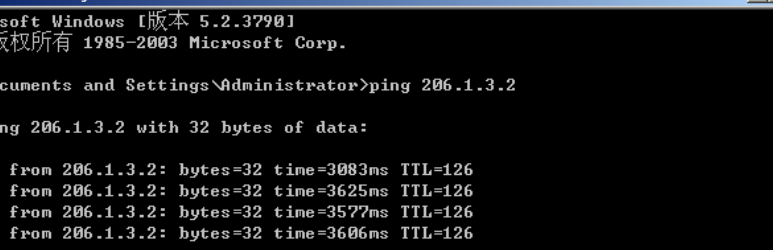
```
开始监听...
找到路由地址: 206.1.1.2
下一跳: 0.0.0.0
成功找到路由表
在ARP表中查询IP : 206.1.1.2
ARP表中无该项!
获取远端MAC地址!
远端IP地址: 206.1.1.2
本地IP地址: 206.1.1.1
成功插入ARP表项!

-----ARP表-----
IP地址: 206.1.2.1      MAC地址: 00-0c-29-ed-79-fe
IP地址: 206.1.1.1      MAC地址: 00-0c-29-ed-79-fe
IP地址: 206.1.1.2      MAC地址: 00-0c-29-75-de-e5
-----

在ARP表中查询IP : 206.1.1.2
目的MAC地址:00-0c-29-75-de-e5
发送数据报到: 00-0c-29-75-de-e5
找到路由地址: 206.1.3.2
下一跳: 206.1.2.2
成功找到路由表
在ARP表中查询IP : 206.1.2.2
ARP表中无该项!
获取远端MAC地址!
远端IP地址: 206.1.2.2
本地IP地址: 206.1.2.1
成功插入ARP表项!

-----ARP表-----
IP地址: 206.1.2.1      MAC地址: 00-0c-29-ed-79-fe
IP地址: 206.1.1.1      MAC地址: 00-0c-29-ed-79-fe
IP地址: 206.1.1.2      MAC地址: 00-0c-29-75-de-e5
IP地址: 206.1.2.2      MAC地址: 00-0c-29-66-05-1a
-----

在ARP表中查询IP : 206.1.2.2
下一跳MAC地址:00-0c-29-66-05-1a
发送数据报到: 00-0c-29-66-05-1a
```



The screenshot shows a Windows command prompt window with the title bar "C:\WINDOWS\system32\cmd.exe". The window contains the following text:

```
Microsoft Windows [版本 5.2.3790]  
(C) 版权所有 1985-2003 Microsoft Corp.  
  
C:\Documents and Settings\Administrator>ping 206.1.3.2  
  
Pinging 206.1.3.2 with 32 bytes of data:  
  
Reply from 206.1.3.2: bytes=32 time=3083ms TTL=126  
Reply from 206.1.3.2: bytes=32 time=3625ms TTL=126  
Reply from 206.1.3.2: bytes=32 time=3577ms TTL=126  
Reply from 206.1.3.2: bytes=32 time=3606ms TTL=126  
  
Ping statistics for 206.1.3.2:  
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),  
    Approximate round trip times in milli-seconds:  
        Minimum = 3083ms, Maximum = 3625ms, Average = 3472ms  
  
C:\Documents and Settings\Administrator>
```

[illegible]

[https://github.com/zhayao99/Net\\_Tech](https://github.com/zhayao99/Net_Tech)