

简单路由器程序的设计报告

学号：2012482

姓名：董伊萌

专业：信息安全

简单路由器程序的设计报告

一. 实验内容说明

二. 实验设计思路

1. 实验环境拓扑图

2. 路由器工作流程

一个较为完整的路由器程序包含大致如下工作：

程序工作流程图如下：

三. 代码实现

1. 代码中的主要数据结构

- 帧首部：
- IP首部：
- ICMP首部：
- ARP数据帧：
- IP数据报：
- 路由表项：
- 路由表
- ARP映射表
- 转发的数据报结构
- 日志类

2. 路由表的相关操作

- 路由表的初始化
- 路由表项的添加
- 路由表项的删除
- 路由表的查询
- 路由表的打印

3. 日志的相关操作

- 下一跳相关信息
- ARP数据报信息

4. 主要函数部分

4.1. 路由程序初始化模块：

- 获取本机网卡的IP
- open打开网络接口
- 获取本机的MAC地址
- 初始化路由表

4.2. 主线程：静态路由表维护模块

4.3. 接收数据包的处理线程：数据包的捕获与处理模块

- 过滤捕获到的报文
- 数据包的捕获
- 收到IP数据包的处理
- 收到ARP数据包的处理（包括缓冲区超时删除）

4.4. 其他函数

校验和计算

校验和检验

四. 实验结果及分析

- 显示本机网卡的双IP地址并获取MAC地址
- 路由表的相关操作
- 主机之间ping
- 工作日志的显示

五. 实验总结与反思

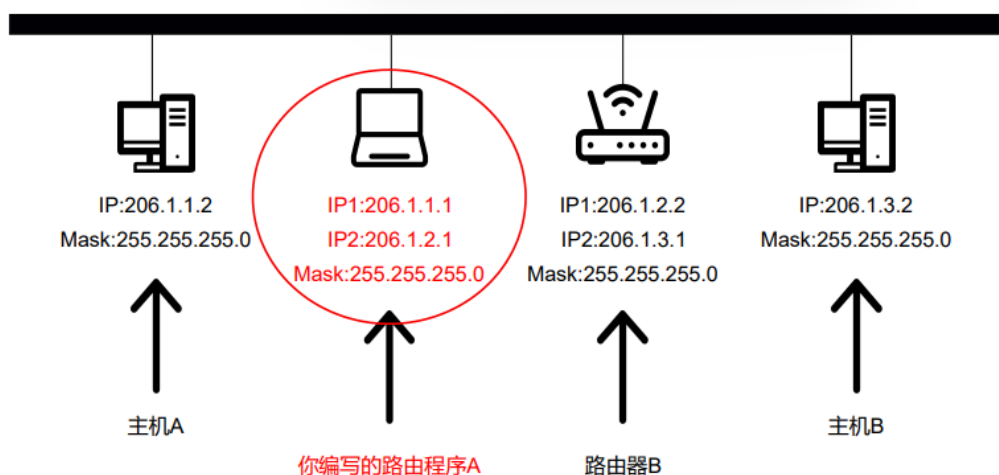
一. 实验内容说明

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。

二. 实验设计思路

1. 实验环境拓扑图



可以看到其中路由器A和B是连接不同子网的通用计算机，路由器B已经配置好，通过在路由器A上运行编写的路由程序，完成双IP的绑定，通过路由程序添加路由表，对接收的数据报进行分析和转发，实现处于不同网络中的主机A和主机B的相互通信。通信通过ping命令执行，也可以通过tracert命令追踪数据报在网络中的转发路径。

2. 路由器工作流程

一个较为完整的路由器程序包含大致如下工作：

① 为经过的IP数据报选择路由，是路由器的基本功能，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。

- 关于路由选择算法：

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以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哪里。

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1。

③ 重新计算IP数据报的头部校验和。

④ 生成和处理ICMP报头。

- 相关的ICMP报文涉及两种：

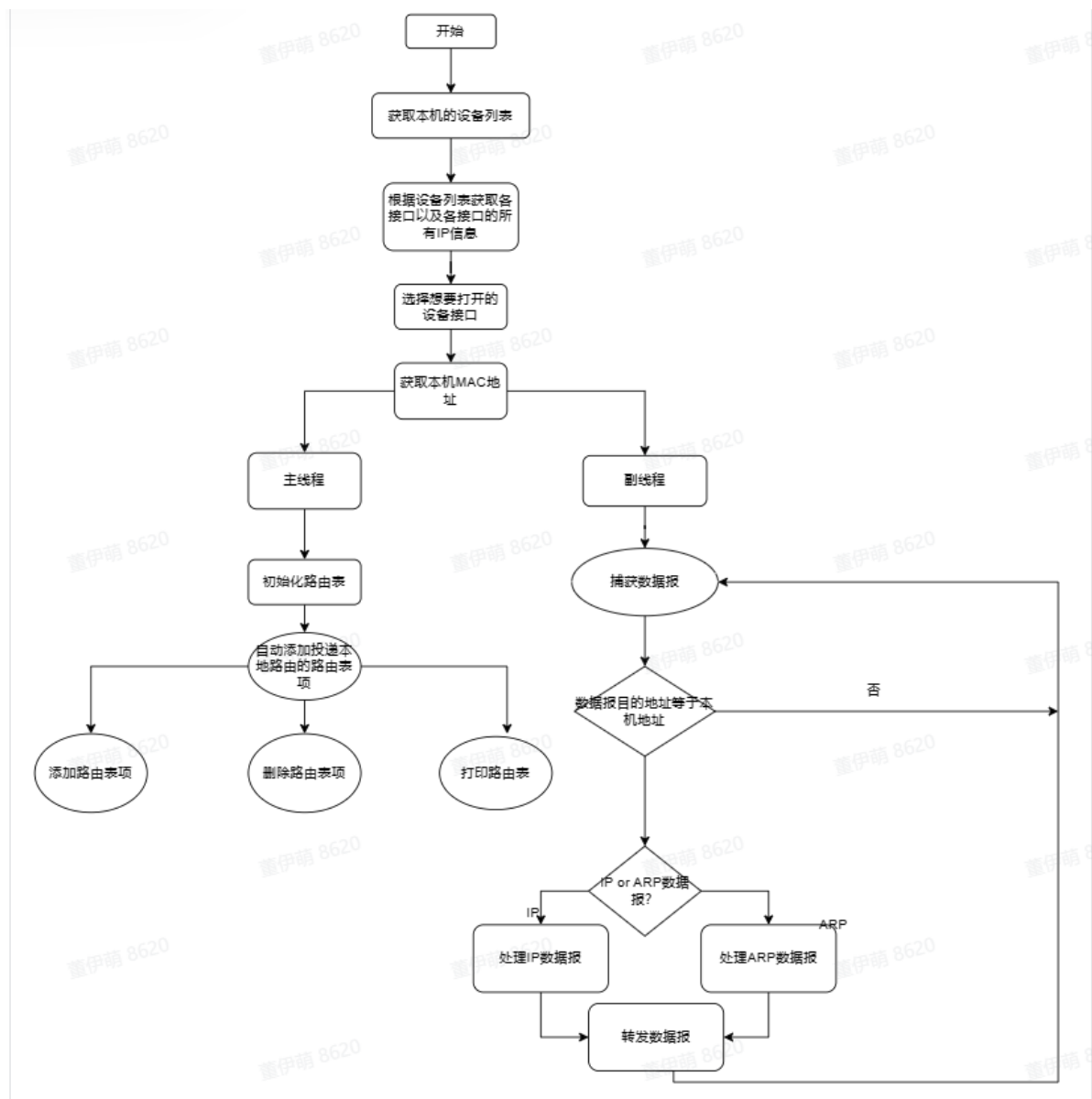
目的不可达报文：当路由器不能为相应的数据包找到路由器或者主机交付数据包时，就丢弃该数据报，向源主机发送ICMP目的不可达报文。

超时报文：路由器在转发数据包时，如果生存周期TTL值减一后为0，就丢弃这个数据包。当丢弃这个数据包的时候，路由器向源主机发出ICMP超时报文。

⑤ 实现路由协议，维护静态路由。

⑥ 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，路由程序需要获取到下一跳的物理地址，然后形成数据帧从选择的网络接口发送出去。

程序工作流程图如下：



三. 代码实现

1. 代码中的主要数据结构

• 帧首部:

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

• IP首部:

```
typedef struct IPHeader_t {
    BYTE Ver_HLen;
    BYTE TOS;
    WORD TotalLen;
    WORD ID;
    WORD Flag_Segment;
    BYTE TTL; //生命周期
    BYTE Protocol;
    WORD Checksum; //校验和
    ULONG SrcIP; //源IP
    ULONG DstIP; //目的IP
}IPHeader_t;
```

• ICMP首部:

```
typedef struct ICMPHeader_t {
    BYTE Type;
    BYTE Code;
    WORD Checksum;
    WORD Id;
    WORD Sequence;
}ICMPHeader_t;
```

• ARP数据帧:

```
typedef struct ARPFrame_t {
    FrameHeader_t FrameHeader; //帧首部
    WORD HardwareType; //硬件类型
    WORD ProtocolType; //协议长度
    BYTE HLen; //硬件地址长度
    BYTE PLen; //协议地址
    WORD Operation; //操作
    BYTE SendHa[6]; //发送方MAC
    DWORD SendIP; //发送方IP
    BYTE RecvHa[6]; //接收方MAC
    DWORD RecvIP; //接收方IP
}ARPFrame_t;
```

• IP数据报:

包含帧首部和IP首部的数据包

```
typedef struct IPFrame_t { //包含帧首部和IP首部的数据包
    FrameHeader_t FrameHeader;
    IPHeader_t IPHeader;
}IPFrame_t;
```

• 路由表项：

本次路由程序的设计对路由表项的存储采用链表结构，路由表项类包括该项在路由表中的索引，掩码，目的网络，下一跳的IP地址以及下一跳的MAC地址异界连接的类型（是否为直接相连）。

```
class RouteEntry
{
public:
    RouteEntry* next;
    int number; //索引
    DWORD mask; //掩码
    DWORD dst_net; //目的网络
    DWORD next_hop; //下一跳的IP地址
    BYTE nextMAC[6]; //下一跳的MAC地址
    int type; //0为直接连接（不可删除），1为用户添加
    RouteEntry() {
        next = NULL;
        number = 0;
        mask = 0;
        dst_net = 0;
        next_hop = 0;
        type = 0;
    }
    RouteEntry(int number, DWORD dstNetwork, DWORD mask, DWORD nextHop, int
access)
    {
        this->number = number;
        this->dst_net = dstNetwork;
        this->mask = mask;
        this->next_hop = nextHop;
        this->type = access;
    }
    void print();
};
```

• 路由表

对路由表项通过链表形式组合成路由表，按照最长匹配原则，为方便寻找最长前缀表项，在插入时对掩码进行排序，掩码越长越靠前，索引越小。同时类中包括添加路由表项、删除路由表项和查找目的IP地址的下一跳的ip地址以及路由表的输出等功能，之后将详细叙述实现方法。

```
class RouteTable {
public:
    RouteEntry* head, * tail; //头节点
    int num; //条数
    RouteTable(); //初始化，添加直接相连的网络
    void add(RouteEntry* entry); //添加路由表项，直接相连的在最前面，其余的按照最长匹配原则
    void erase(int number); //删除第i条路由表项，直接相连的不能删除
    void print();
    DWORD search(DWORD dstip); //根据最长匹配原则查找下一跳的ip地址
};
```

• ARP映射表

ARP映射表中包括了IP地址与MAC地址的对应关系，通过ARP映射表才能准确的转发数据报。同样包括插入和查找对应关系功能。

```
class ArpTable {
public:
    DWORD IP;
    BYTE MAC[6];
    static int num;
    static void add(DWORD IP, BYTE MAC[6]);
    static int search(DWORD IP, BYTE MAC[6]);
};
```

• 转发的数据报结构

```
class PacketBuffer
{
public:
    BYTE          pktData[MAX_SIZE]; // 数据缓存
    int           totalLen; // 数据包总长度
    ULONG         targetIP; // 目的IP地址
    bool          valid = 1; // 有效位: 如果已经被转发或者超时, 则置0
    clock_t       Time; // 超时判断
    PacketBuffer() {};
    PacketBuffer(const PacketBuffer& x) // 复制构造函数---->检查!!!
    {
        memcpy(this->pktData, x.pktData, x.totalLen);
        this->totalLen = x.totalLen;
        this->targetIP = x.targetIP;
        this->valid = x.valid;
        this->Time = x.Time;
    }
};
```

• 日志类

在日志类中，可以实现数据报获取和转发的过程（包括IP，ARP，ICMP数据报），展示了数据报的IP，目的IP，源MAC，目的MAC地址，同样对接收到的IP数据报展示了下一跳信息。

```
class Log
{
public:
    Log(); // 打开文件进行写入
    ~Log(); // 关闭文件!
    static FILE* my_fp;
    // 写入日志
    static void addInfo(const char* str /* 日志信息标识 */);
    static void addInfohop(const char* str /* 日志信息标识 */, DWORD hop);
    static void ARP_info(const char* str /* 日志信息标识 */, ARPFrame_t* p); // arp类型
    static void IP_info(const char* str, IPFrame_t* p); // ip类型
    static void ICMP_info(const char* str); // icmp类型
};
```

2. 路由表的相关操作

• 路由表的初始化

把得到的双IP相关信息添加到路由表（通过下面介绍的add函数）中，类型 `type` 设置为0，表示为直接相连，类型设置为0，为不可删除项

```
RouteTable::RouteTable()
{
    head = new RouteEntry;
    head->next = NULL;
    num = 0;
    //通过得到的双IP的掩码，在路由表中添加直接相连的网络，将类型设置为0，即不可删除项
    for (int i = 0; i < 2; i++)
    {
        RouteEntry* entry = new RouteEntry;
        //添加直接相连的网络
        entry->dst_net = (inet_addr(my_ip[i])) & (inet_addr(my_mask[i]));
        entry->mask = inet_addr(my_mask[i]);
        entry->type = 0;
        this->add(entry);
    }
}
```

• 路由表项的添加

由于采取的是最长匹配原则，所以对路由表项的存储采用按照掩码排序，掩码越大存的越靠前。

实现方法为：如果路由表为空，将表项 `RouteEntry* r` 直接添加到 `head` 后面，下一项设置为 `NULL`；如果路由表非空，从路由表中第一项开始循环比较要插入的表项的掩码与表中已有的掩码，找到合适的位置（掩码小于前一项而大于后一项）插进去。插入后需要从表头遍历，对表中的路由表项重新编号，并且把路由表中的路由表项个数 `num + 1`。

```
void RouteTable::add(RouteEntry* r) {

    RouteEntry* temp;

    if (num == 0) {
        head->next = r;
        r->next = NULL;
    }
    else {
        temp = head->next;
        while (temp != NULL) {

            if (temp->next == NULL || (r->mask < temp->mask && r->mask >= temp->next->mask)) {

                break;
            }
            temp = temp->next;
        }
        if (temp->next == NULL)
```



```

    {

        r->next = NULL;
        temp->next = r;
    }
    else {
        r->next = temp->next;
        temp->next = r;
    }
}
RouteEntry* p = head->next;
//重新编号
for (int i = 0; p != NULL; i++)
{

    p->number = i;
    p = p->next;
}
num++;
return;
}

```

• 路由表项的删除

输入的参数为索引，即根据路由表想要删除该索引的路由表项，首先从头遍历，查找是否有对应的索引，如果没有的话，直接报错："查无此项，请重新输入！\n"；如果索引为第一项，首先判断它的类型是否为直接相连，直接相连的路由表项无法删除，如果不是的话，只有一项，直接让 head 的下一项为空，有多项的话，先存下第一项 temp，之后令链表头的下一项直接为 temp 的下一项。如果是在中间位置，遍历到 index 处，寻找删除结点的前驱结点，存下要删除的点 rem，让 rem 的前一项的下一项为 rem 的下一项，完成删除后重新编号。

```

void RouteTable::erase(int index) {

    RouteEntry* temp = new RouteEntry;
    bool hav = false;
    for (RouteEntry* t = head; t->next != NULL; t = t->next)
    {
        if (t->next->number == index)hav = true;
    }
    if (!hav) {
        printf("查无此项，请重新输入！\n");
        return;
    }

    if (index == 0)//删除第一项
    {
        temp = head->next;
        if (temp->type == 0)
        {
            printf("没有权限删除该项！\n");
            return;
        }
        else
        {

```

```

        if (num == 1)
        {
            head->next = NULL;
        }
        else
        {
            temp = head->next;
            head->next = temp->next;
            printf("已成功删除指定项! \n");

        }

    }
}
else
{
    temp = head->next;
    for (int i = 0; i < index - 1; i++)//遍历到index处,寻找删除结点的前驱结点
    {
        temp = temp->next;
    }
    RouteEntry* rem = new RouteEntry;
    rem = temp->next;//要删除的结点x
    if (rem->next == NULL)//尾删
    {
        if (rem->type == 0)
        {
            printf("没有权限删除该项! \n");
            return ;
        }

        temp->next = NULL;

        // printf("已成功删除指定项! \n");

    }
    //中间删
    temp->next = rem->next;
    printf("已成功删除指定项! \n");

}
/*重新编号*/

RouteEntry* p = head->next;
//重新为表项编号
for (int i = 0; p != NULL; i++)
{
    p->number = i;
    p = p->next;
}
num--;
my_log.addInfo("路由表删除成功!");
return ;
}

```

• 路由表的查询

查找目的ip地址的下一跳的IP地址，遍历路由表，直到找到网络号匹配的一项（即目的IP地址和掩码相与得到的就是对应的网络号），因为路由表项的排列遵循最长匹配原则，所以找到的第一项就是。如果该项的类型是0，那么说明是直接相连的，直接投递即可，下一跳的地址即为查找的IP地址；如果类型为1，返回表中存储的下一条的地址即 `next_hop`。如果没有查询到返回-1，据此，之后可以发送ICMP目的不可达数据报。

```
DWORD RouteTable::search(DWORD dst_ip) {

    DWORD a = -1;
    RouteEntry* t = head->next;
    for (; t!= NULL; t = t->next) {

        if ((t->mask & dst_ip) == t->dst_net) {
            if (t->type == 0) //直接相连的网络
            {
                a= dst_ip;
            }
            else
                a = t->next_hop;
        }

    }

    return a;
}
```

• 路由表的打印

路由表项的打印

```
void RouteEntry::print()
{
    //打印序号
    printf("%d  ", number);

    in_addr addr;

    //打印掩码
    addr.s_addr = mask;
    //转成主机序
    char* str = inet_ntoa(addr);
    printf("%s\t", str);

    //打印目的主机
    addr.s_addr = dst_net;
    str = inet_ntoa(addr);
    printf("%s\t", str);

    //打印下一跳ip地址
    addr.s_addr = next_hop;
    str = inet_ntoa(addr);
    printf("%s\t", str);
}
```

```

//用户是否可操作

if (type == 0)
    printf("non_accessible\n");
else
    printf("accessible\n");

printf("\n");

}

```

之后通过链表遍历路由表，调用路由表项的输出函数

```

void RouteTable::print() {
    RouteEntry* temp = head->next;
    printf("-----路由表-----\n");
    int t = 0;
    while (temp!=NULL) {
        temp->print();
        temp = temp->next;
        t++;
    }
    printf("共有路由表项: ");
    printf("%d ", num);
    printf("条/n");
}

```

3. 日志的相关操作

• 下一跳相关信息

根据参数中的IP，输出下一条的信息，将DWORD类型通过 `inet_ntoa` 转成char*类型的主机序的IP地址便于观察。

```

void Log::addInfohop(const char* str/*日志信息标识*/, DWORD hop) {
    fprintf(my_fp, str);
    if (hop == -1)
        fprintf(my_fp, "%s \n", "-1");
    in_addr addr;
    addr.s_addr = hop;
    char* str1 = inet_ntoa(addr);
    fprintf(my_fp, "%s \n", str1);
}

```

• ARP数据报信息

根据参数中的ARP数据帧类型，对于IP地址，将DWORD类型通过 `inet_ntoa` 转成char*类型的主机序的IP地址；对于MAC地址，通过遍历数组，并转换为16进制输出，不足两位，前面补0输出。

```

void Log::ARP_info(const char* str, ARPFrame_t* p) {
    fprintf(my_fp, str);
}

```

```

        fprintf(my_fp, "-----ARP 数据包-----\n");

        in_addr addr;
        addr.s_addr = p->SendIP;
        char* str1 = inet_ntoa(addr);
        fprintf(my_fp, "源IP: ");
        fprintf(my_fp, "%s \n", str1);

        fprintf(my_fp, "源MAC: ");
        for (int i = 0; i < 5; i++)
            fprintf(my_fp, "%02X-", p->SendHa[i]);
        fprintf(my_fp, "%02X\n", p->SendHa[5]);

        in_addr addr2;
        addr2.s_addr = p->RecvIP;
        char* str2 = inet_ntoa(addr2);
        fprintf(my_fp, "目的IP: ");
        fprintf(my_fp, "%s \n", str2);

        fprintf(my_fp, "目的MAC: ");
        for (int i = 0; i < 5; i++)
            fprintf(my_fp, "%02X-", p->RecvHa[i]);
        fprintf(my_fp, "%02X\n", p->RecvHa[5]);
        fprintf(my_fp, "\n");
    }

```

IP类型的数据报实现方法类似，这里不再赘述；对于ICMP报文只需输出相关提示信息，是否超时，是否目的不可达等等。

4. 主要函数部分

4.1. 路由程序初始化模块：

● 获取本机网卡的IP

```
void Get_Two_IP()
```

首先通过函数 `pcap_findalldevs_ex` 获取网络接口的设备列表，然后通过指向链表首部的 `alldevs` 打印设备的相关信息，包括设备的名字和描述信息，并获得每个网络接口绑定的IP的相关信息，打印IP地址和网络掩码；接下来由用户指定要打开的网卡，打开后，将双IP地址存到 `my_ip` 数组中，把其对应的掩码存到 `my_mask` 中。

```

void Get_Two_IP() {

    pcap_if_t* dev;//用于遍历网卡信息链表
    pcap_addr_t* add;//用于遍历IP地址信息链表：一个网卡可能有多个IP地址
    if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, //获取本机的接口设备
        NULL, //无需认证
        &alldevs, //指向设备列表首部
        errbuf
    ) == -1)//返回-1表示出错
    {

```

```

    printf("There's Error in pcap_findalldevs_ex! Program exit.\n");
    exit(1);
}
//显示设备列表信息
int index = 0;
for (dev = alldevs; dev; dev = dev->next) {
    printf("%d. %s", ++index, dev->name);
    if (dev->description)
        printf("( %s )\n", dev->description);
    else
        printf("(No description )\n");

    //获取该网络接口设备的IP地址信息
    for (add = dev->addresses; add != NULL; add = add->next) {
        if (add->addr->sa_family == AF_INET) { //判断该地址是否IP地址
            //输出相关的信息
            char str[INET_ADDRSTRLEN];
            //通过 inet_ntoa将一个网络字节序的IP地址转化为点分十进制的IP地址（字符串）
            strcpy(str, inet_ntoa(((struct sockaddr_in*)add->addr)->sin_addr));

            printf("IP地址: %s\n", str);
            strcpy(str, inet_ntoa(((struct sockaddr_in*)add->netmask)->sin_addr));
            printf("网络掩码: %s\n", str);

        }
    }
}
if (index == 0)
{
    printf("\nNo interfaces found!\n");
}
printf("-----\n\n");
dev = alldevs;
int num;
printf("请选择您要打开的网卡: ");
scanf("%d", &num);

//遍历寻找要打开的网络
for (int i = 0; i < num - 1; i++) {
    dev = dev->next;
}

int t = 0;
for (add = dev->addresses; add != NULL && t < 10; add = add->next) {
    if (add->addr->sa_family == AF_INET) { //是IP类型的
        strcpy(my_ip[t], inet_ntoa(((struct sockaddr_in*)add->addr)->sin_addr));
        strcpy(my_mask[t], inet_ntoa(((struct sockaddr_in*)add->netmask)->sin_addr));
        t++;
    }
}
open_dev = open(dev->name);

```

```

    if (open_dev == NULL) {
        pcap_freealldevs(alldevs);

    }

    pcap_freealldevs(alldevs);

}

```

• open打开网络接口

利用 `pcap_open` 打开对应名字的网络接口

```

pcap_t* open(char* name) {

    pcap_t* temp = pcap_open(name, 65536/*最大值*/, PCAP_OPENFLAG_PROMISCUOUS,
1000, NULL, errbuf);
    if (temp == NULL)
        printf("无法打开设备\n");
    return temp;
}

```

• 获取本机的MAC地址

首先利用自定义函数 `SET_ARP_Frame_HOST` 构造要发送的ARP数据包：DesMac设置为广播地址；SrcMAC、SendHa、SendIP随意设置；RecvHa设置为0，在请求报文里表示目的地址未知；RecvIP设置为刚刚获取的本机IP地址；帧类型为ARP；硬件类型为以太网；协议类型为IP；硬件地址长度为6；协议地址长度为4；操作为ARP请求。

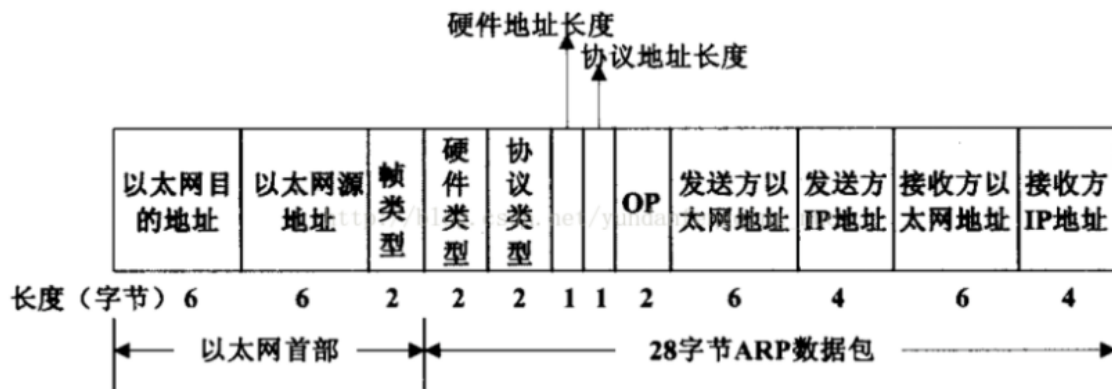
```

void SET_ARP_Frame_HOST(ARPFrame_t& ARPFrame1, DWORD ip) {
    for (int i = 0; i < 6; i++) {
        ARPFrame1.FrameHeader.DesMAC[i] = 0xff;
        ARPFrame1.FrameHeader.SrcMAC[i] = 0x0f;
        ARPFrame1.SendHa[i] = 0x0f;
        ARPFrame1.RecvHa[i] = 0x00;
    }

    ARPFrame1.FrameHeader.FrameType = htons(0x0806);
    ARPFrame1.HardwareType = htons(0x0001);
    ARPFrame1.ProtocolType = htons(0x0800);
    ARPFrame1.HLen = 6;
    ARPFrame1.PLen = 4;
    ARPFrame1.Operation = htons(0x0001);
    ARPFrame1.SendIP = inet_addr("10.10.10.10");
    ARPFrame1.RecvIP = ip;
}

```

在循环中，用 `pcap_sendpacket` 向指定的网卡发送构造好的数据包，利用 `pcap_next_ex` 捕获数据包，设置过滤条件为：帧类型为ARP，且操作类型为ARP响应；捕获到符合条件的ARP数据包后，通过 `my_log.ARP_info` 展示一下包的内容。将报文中的MAC地址存入 `my_mac` 中，根据下图中的报文格式可知，`pkt_data + 22`指向的是发送方的MAC地址，长度为6。



代码实现如下：

```
void Get_Host_Mac(DWORD ip) {
    memset(my_mac, 0, sizeof(my_mac));
    ARPFrame_t ARPFrame_Send;
    SET_ARP_Frame_HOST(ARPFrame_Send, ip);
    struct pcap_pkthdr* pkt_header;
    const u_char* pkt_data;
    struct pcap_pkthdr* header = new pcap_pkthdr;
    int k;
    my_log.addInfo("-----获取本机MAC地址-----");
    -----\n");
    my_log.ARP_info("发送ARP请求包", &ARPFrame_Send);
    //发送构造好的数据包
    //用pcap_next_ex()捕获数据包, pkt_data指向捕获到的网络数据包

    //这里可能后续需要修改
    while ((k = pcap_next_ex(open_dev, &pkt_header, &pkt_data)) >= 0) {
        //发送数据包
        /* if (pcap_sendpacket(choosed_dev, (u_char*)&ARPFrame_Send,
        sizeof(ARPFrame_t)) != 0) {
            cout << "Error in pcap_sendpacket";
            pcap_freealldevs(alldevs);
            return 0;
        }
        */
        pcap_sendpacket(open_dev, (u_char*)&ARPFrame_Send, sizeof(ARPFrame_t));
        struct ARPFrame_t* arp_message;
        arp_message = (struct ARPFrame_t*)(pkt_data);
        if (k == 0) continue;
        else
        {
            //帧类型为ARP, 且操作类型为ARP响应

            if (arp_message->FrameHeader.FrameType == htons(0x0806) &&
            arp_message->Operation == htons(0x0002)) {

                //展示一下包的内容
                my_log.ARP_info("收到ARP应答包", arp_message);
                ShowArp(arp_message);
                //用my_mac记录本机的MAC地址, 用于后续构造ARP数据包
                for (int i = 0; i < 6; i++) {
                    my_mac[i] = *(unsigned char*)(pkt_data + 22 + i);
                }
            }
        }
    }
}
```



```

        }
        printf("成功获取本机MAC地址\n");
        break;
    }
}
}
my_log.addInfo("-----成功获取本机MAC地址-----
-----\n");
}

```

• 初始化路由表

调用上述解释过的路由表初始化函数 `RouteTable::RouteTable()`，初始化路由表，在路由表中添加直接相连的双IP地址对应的相关信息，并手动添加路由表项

```

掩码: 255.255.255.0
目的网络: 206.1.3.0
下一跳的IP地址: 206.1.2.2

```

此时整个实验拓扑图已经完成连接。接下来将进行静态路由表维护模块，也是本程序的主线程部分。

4.2. 主线程：静态路由表维护模块

这一部分主要由路由表的添加、删除、打印组成，根据用户输入想要进行的操作调用对应的路由表的添加、删除、打印函数。这些函数的具体实现方法上述报告已经提及，这里不再赘述。

```

int op;
while (1)
{
    printf("\n\n请选择要进行的操作: \n1. 添加路由表项\n2. 删除路由表项\n3. 打印路由表\n");
    scanf("%d", &op);
    RouteEntry* entry = new RouteEntry;

    switch (op) {
    case 1:
        char t[30];
        printf("请输入掩码: ");
        scanf("%s", &t);
        entry->mask = inet_addr(t);
        printf("请输入目的网络: ");
        scanf("%s", &t);
        entry->dst_net= inet_addr(t);
        printf("请输入下一跳的IP地址: ");
        scanf("%s", &t);
        entry->next_hop = inet_addr(t);
        entry->type = 1;
        entry->print();
        my_route.add(entry);
        break;
    case 2:
        my_route.print();
    }
}

```

```

        printf("请输入要删除的表项的索引: ");
        int i;
        scanf("%d", &i);
        my_route.erase(i);
        break;
    case 3:
        my_route.print();
        break;
    default:
        printf("无效操作, 请重新输入\n");
        break;
}
}

```

4.3. 接收数据包的处理线程：数据包的捕获与处理模块

数据包的捕获与处理模块用于捕获流经本路由器的数据包，在线程 `Recv_Handle` 中按照协议路由处理接收到的数据包。

• 过滤捕获到的报文

设置捕获报文的过滤条件，通过绑定过滤器，设置只捕获IP和ARP数据包。主要通过调用 `pcap_compile` 和 `pcap_setfilter` 函数。

```

if (pcap_compile(open_dev, &fcode, "ip or arp", 1, bpf_u_int32(my_mask[0])) < 0)
{
    fprintf(stderr, "\nError filter: wrong syntax.\n");
    system("pause");
    return -1;
}

//根据编译好的过滤码设置过滤条件
if (pcap_setfilter(open_dev, &fcode) < 0)
{
    fprintf(stderr, "\nError setting the filter\n");
    system("pause");
    return -1;
}

```

• 数据包的捕获

调用 `pcap_next_ex` 捕获数据包，格式化收到的包为帧首部，以获取目的MAC地址和帧类型，并且只处理目的MAC是本机MAC的数据包

```

pcap_pkthdr* pkt_header = NULL;
const u_char* pkt_data = NULL;
while (1)
{
    int ret = pcap_next_ex(open_dev, &pkt_header, &pkt_data); //抓包
    if (ret) break; //接收到消息
}

//格式化收到的包为帧首部，以获取目的MAC地址和帧类型
FrameHeader_t* recv_header = (FrameHeader_t*)pkt_data;
//只处理目的mac是自己的包

```

```

    if (Compare(recv_header->DesMAC, my_mac)){
        //处理IP数据包
        //。 。 。 。 。
        //处理ARP数据包
        //。 。 。 。 。

    }

```

● 收到IP数据包的处理

处理的主要思路如下：

- ① 查看报文中的TTL值，如果小于等于0的话，说明超时，发送ICMP超时报文，继续接收下一条报文。
- ② 查询路由表，找路由，找到对应目的地址的下一跳IP，如果查询结果为-1，发送ICMP目的不可达报文，继续接收下一条报文。
- ③ 计算报文中的IP校验和并进行检验，如果校验和错误直接丢弃数据包不处理，继续接收下一条数据包。
- ④ 重新封装数据包，将源地址设置为本机mac地址，TTL-1，将IP头部的校验和设置为0，重新计算IP头部校验和
- ⑤ 查看目的地址是否在ARP表（IP-MAC映射表）中，如果在，则直接转发。
- ⑥ IP-MAC地址映射表中不存在该映射关系，先缓存IP数据报入缓存队列 `my_buffer` 中，并设置定时器，发送ARP请求获取目的IP的MAC地址。设置缓冲队列最大为 `MAX_BUFFER_COUNT`，如果缓冲区已经满了，则不做处理，提示信息："缓冲区已满，丢弃该IP数据包"。

关于发送ARP请求报文获得目的IP的MAC地址，首先要构造ARP数据报：DesMAC设置为广播地址；SrcMAC和SendHa为本机的MAC地址，用获得的my_mac数组填充；RecvHa设置为0；SendIP为本机的IP地址，用上面得到的ip设置；RecvIP设置为目的主机的IP地址，从参数中获取。

```

void SET_ARP_Frame_DEST(ARPFrame_t& ARPFrame, char ip[20], unsigned char* mac) {
    for (int i = 0; i < 6; i++) {
        ARPFrame.FrameHeader.DesMAC[i] = 0xff; //将ARPFrame.FrameHeader.DesMAC设置为广播地址
        ARPFrame.RecvHa[i] = 0x00;
        ARPFrame.FrameHeader.SrcMAC[i] = mac[i]; //设置为本机网卡的MAC地址
        ARPFrame.SendHa[i] = mac[i]; //设置为本机网卡的MAC地址
    }
    ARPFrame.FrameHeader.FrameType = htons(0x0806); //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
    ARPFrame.SendIP = inet_addr(ip); //将ARPFrame->SendIP设置为本机网卡上绑定的IP地址
}

```

然后调用 `pcap_sendpacket` 发送构造好的数据报

```

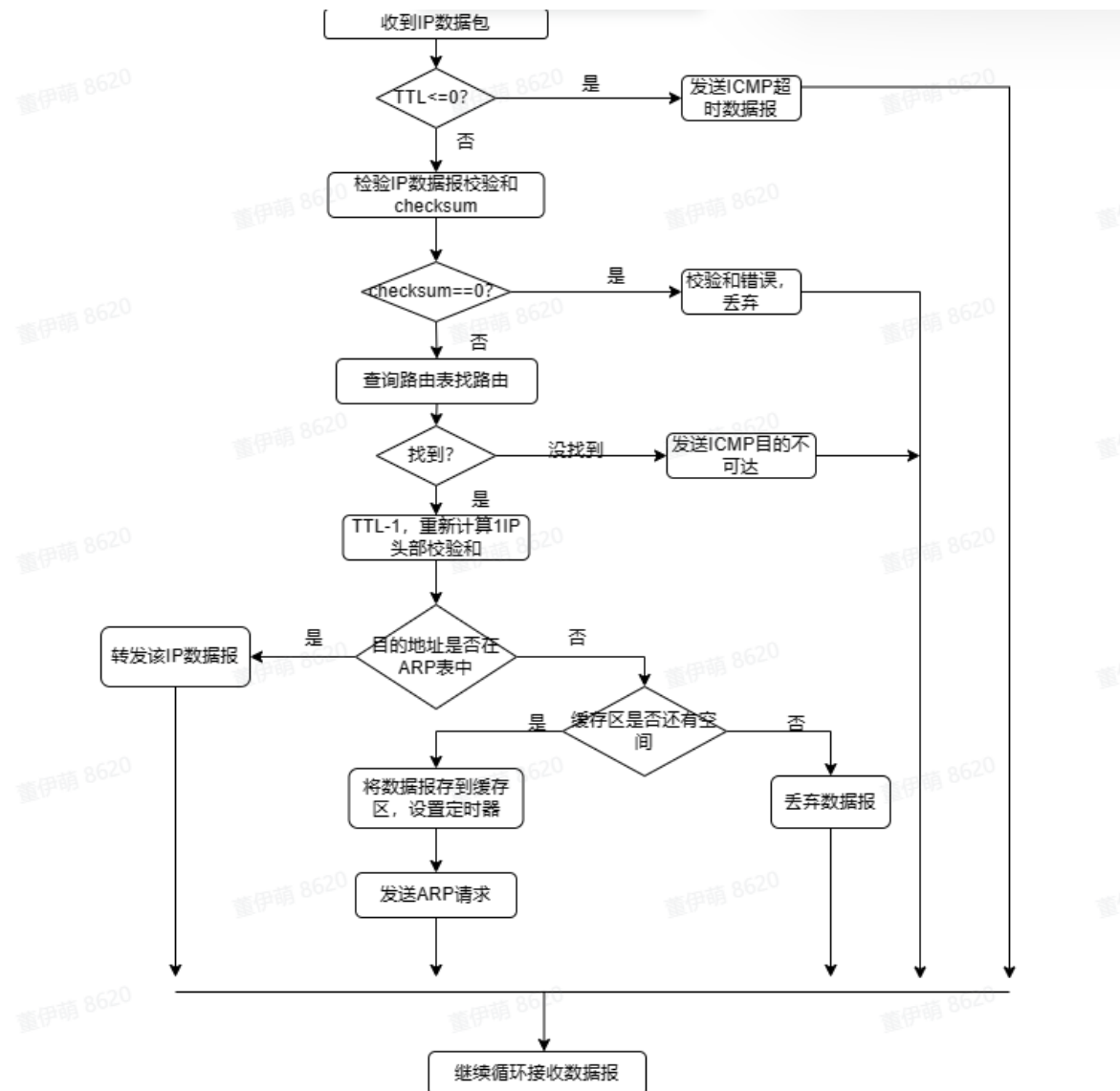
void Get_Other_Mac(DWORD ip_) {
    /*这里只发送ARP请求!!!*/

    ARPFrame_t ARPFrame;
    SET_ARP_Frame_DEST(ARPFrame, my_ip[0], my_mac);
    ARPFrame.RecvIP = ip_;

    my_log.addInfo("-----获取远程主机MAC地址-----\n");
    pcap_sendpacket(open_dev, (u_char*)&ARPFrame, sizeof(ARPFrame_t));
    my_log.ARP_info("发送ARP请求包", &ARPFrame);
}

```

主要流程图如下：



```

//收到IP数据报
if (ntohs(recv_header->FrameType) == 0x800)
{
    //格式化收到的包为帧首部+IP首部类型
    IPFrame_t* data = (IPFrame_t*)pkt_data;
}

```

```

my_log.IP_info("接收IP数据报\n", data);
//获取目的IP地址并在路由表中查找, 并获取下一跳ip地址

// ICMP超时
if (data->IPHeader.TTL <= 0)
{
    //发送超时报文
    Send_ICMP_Pac(11, 0, pkt_data);
    my_log.addInfo("发送ICMP超时数据包!");
    continue;
}

IPHeader_t* IpHeader = &(data->IPHeader);
// 数据报损坏或是出错
if (check_checksum(data) == 0)
{
    my_log.IP_info("校验和错误, 丢弃", data);
    continue;
}
DWORD dstip = data->IPHeader.DstIP;

DWORD nexthop = router_table.search(dstip);

my_log.addInfohop("接收到的IP数据报目的为", dstip);
my_log.addInfohop("接收到的IP数据报下一跳为", nexthop);
//无匹配项

if (nexthop == -1)
{
    Send_ICMP_Pac(3, 0, pkt_data); // ICMP目的不可达
    my_log.addInfo("发送ICMP目的不可达数据报!");
    continue;
}
else
{
    PacketBuffer packet;
    packet.targetIP = nexthop;

    //重新封装MAC地址(源MAC地址变为my_mac)
    for (int t = 0; t < 6; t++)
    {
        data->FrameHeader.SrcMAC[t] = my_mac[t];
    }

    data->IPHeader.TTL -= 1; // TTL减1
    // 设IP头中的校验和为0
    data->IPHeader.Checksum = 0;
    unsigned short buff[sizeof(IPHeader_t)];

    memset(buff, 0, sizeof(IPHeader_t));
    IPHeader_t* header = &(data->IPHeader);

```

```

memcpy(buff, header, sizeof(IPHeader_t));

// 计算IP头部校验和
// data->IPHeader.Checksum = cal_checksum(check_buff,
sizeof(IPHeader_t));
data->IPHeader.Checksum = calChecksum1(header);

// IP-MAC地址映射表中存在该映射关系
//根据nexthop取ARP映射表中查看是否存在映射

if (my_arptable->search(nexthop, data->FrameHeader.DesMAC))
{
    //查找到了数据报可以直接转发
    memcpy(packet.pktData, pkt_data, pkt_header->len);
    packet.totalLen = pkt_header->len;
    if (pcap_sendpacket(open_dev, (u_char*)packet.pktData,
packet.totalLen) != 0)
    {
        // 错误处理
        continue;
    }
    my_log.addInfo("-----转发
发数据包-----");
    my_log.IP_info("转发", data);
}

// IP-MAC地址映射表中不存在该映射关系,获取
//先缓存IP数据报
else
{
    //最多存50条
    if (pktNum < MAX_BUFFER_COUNT)    // 存入缓存队列
    {
        packet.totalLen = pkt_header->len;
        // 将需要转发的数据报存入缓冲区
        memcpy(packet.pktData, pkt_data, pkt_header->len);

        my_buffer[pktNum++] = packet;

        packet.Time = clock();

        my_log.IP_info("缓存IP数据报\n", data);
        // 发送ARP请求
        Get_Other_Mac(packet.targetIP);
    }
    else
    {
        my_log.addInfo("缓冲区已满, 丢弃该IP数据包");
        my_log.IP_info("缓冲区溢出, 丢弃", data);
    }
}
}
}
}

```

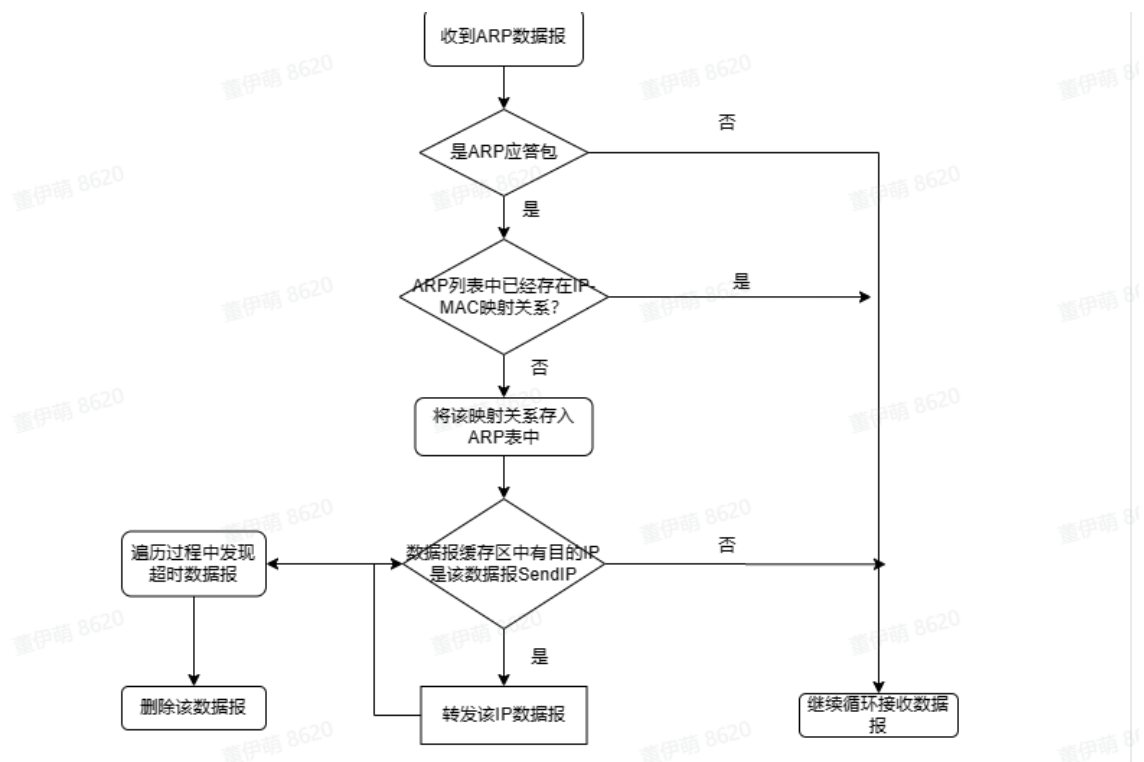
● 收到ARP数据包的处理（包括缓冲区超时删除）

处理思路如下：

- ① 接收到ARP数据报后，先判断是否为响应报文，如果不是的话，什么也不做，直接继续接收新的报文。
- ② 在IP-MAC映射表 `my_arptable` 中查找是否已经存在对应的映射关系，如果存在的话，直接继续接收新的报文。
- ③ 在IP-MAC映射表 `my_arptable` 中不存在对应的映射关系，通过 `my_arptable->add(data->SendIP, data->SendHa)` 将映射关系存入，然后遍历缓冲区中是否有目的IP是该MAC对应的IP地址的数据报，有则进行转发。
- ④ 在遍历缓冲区的时候，如果发现了超时的数据报，则将其从缓冲区中删除，调用自定义函数 `Delete_Buffer`，将已经占用缓冲区的总数据包个数 `pktNum` -1。

```
void Delete_Buffer(PacketBuffer my_buffer1[50],int target) {  
    int j = 0;  
    for (int i = 0; i < pktNum; i++) {  
        if (i != target) {  
            my_buffer1[j] = my_buffer1[i];  
            j++;  
        }  
    }  
}
```

处理流程图如下：



代码如下：

```
if (ntohs(recv_header->FrameType) == 0x806)
```

```

{
    ARPFrame_t* data = (ARPFrame_t*)pkt_data; //格式化收到的包为帧首部
+ARP首部类型

    my_log.ARP_info("接收ARP响应包", data);
    //收到ARP响应包
    //处理响应报文
    if (data->Operation == ntohs(0x0002)) {
        BYTE tmp_mac[6] = { 1 };

        if (my_arptable->search(data->SendIP, tmp_mac)) { //该映射关系已
经存到路由表中，不做处理
        }
        else {

            DWORD tmp_ip = data->SendIP;
            for (int i = 0; i < 6; i++) {
                tmp_mac[i] = data->SendHa[i];
            }

            //存表
            my_arptable->add(data->SendIP, data->SendHa);

            //遍历缓冲区，看是否有可以转发的包
            for (int i = 0; i < pktNum; i++)
            {
                PacketBuffer packet = my_buffer[i];
                if (packet.valid == 0) continue;
                if (clock() - packet.Time >= 6000) { //超时
                    // packet.valid = 0;
                    // my_buffer[i].valid = 0;
                    Delete_Buffer(my_buffer, i);
                    pktNum -= 1;
                    continue;
                }
                //往此IP地址转发
                if (packet.targetIP == data->SendIP)
                {
                    IPFrame_t* ipframe = (IPFrame_t*)packet.pktData;
                    //重新封装IP包
                    for (int i = 0; i < 6; i++) {
                        ipframe->FrameHeader.SrcMAC[i] = my_mac[i];
                        ipframe->FrameHeader.DesMAC[i] = data->
+SendHa[i];

                    }
                    // 发送IP数据包
                    pcap_sendpacket(open_dev,
(u_char*)packet.pktData, packet.totalLen);

                    my_buffer[i].valid = 0;
                    my_log.addInfo("-----");
                    my_log.IP_info("转发", ipframe);
                    my_log.addInfo("-----");
                    -----该数据包转发成功-----");
                }
            }
        }
    }
}

```



```

    }

    }

    }
    else if (data->Operation == ntohs(0x0002)) {}//请求报文什么也不做
}

```

4.4. 其他函数

程序中还有很多辅助程序完整实现的函数，比如比较两个MAC地址是否相同、展示ARP数据帧、打印MAC地址、以及关于校验和计算与检验，报告中仅详细解释校验和部分的实现设计思路。

校验和计算

校验和计算的主要思路为：

- ① 将校验和字段设置为0（传进来的是已经为0的）
- ② 按顺序对每16位（1个字=2字节）进行加法运算；
- ③ 如果溢出就将进位加到最低位
- ④ 对累加的结果取反——就是头部校验和值

```

unsigned short calChecksum2(unsigned short* pBuffer, int nSize) {

    unsigned long ulChecksum = 0;
    while (nSize > 1)
    {
        ulChecksum += *pBuffer++;
        nSize -= sizeof(unsigned short);//每16位一组
    }
    if (nSize)
    {
        ulChecksum += *(unsigned short*)pBuffer;
    }

    ulChecksum = (ulChecksum >> 16) + (ulChecksum & 0xffff);
    ulChecksum += (ulChecksum >> 16);
    return (unsigned short)(~ulChecksum);
}

```

校验和检验

```

bool check_checksum(IPFrame_t* temp) {
    unsigned int sum = 0;
    WORD* t = (WORD*)&temp->IPHeader;//每16位为一组
    for (int i = 0; i < sizeof(IPHeader_t) / 2; i++) {
        sum += t[i];
        while (sum >= 0x10000) {
            //包含原校验和一起进行相加
            int s = sum >> 16;
            sum -= 0x10000;
            sum += s;
        }
    }
}

```

```

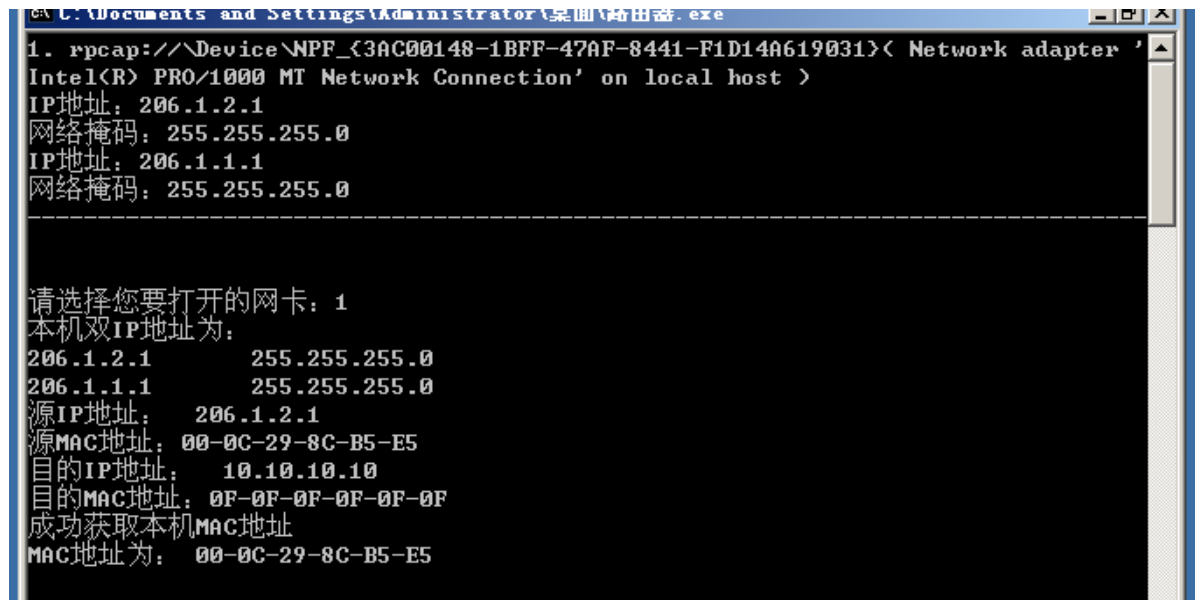
}

if (sum == 65535) return 1; //全1, 校验和正确
return 0;
}

```

四. 实验结果及分析

- 显示本机网卡的双IP地址并获取MAC地址



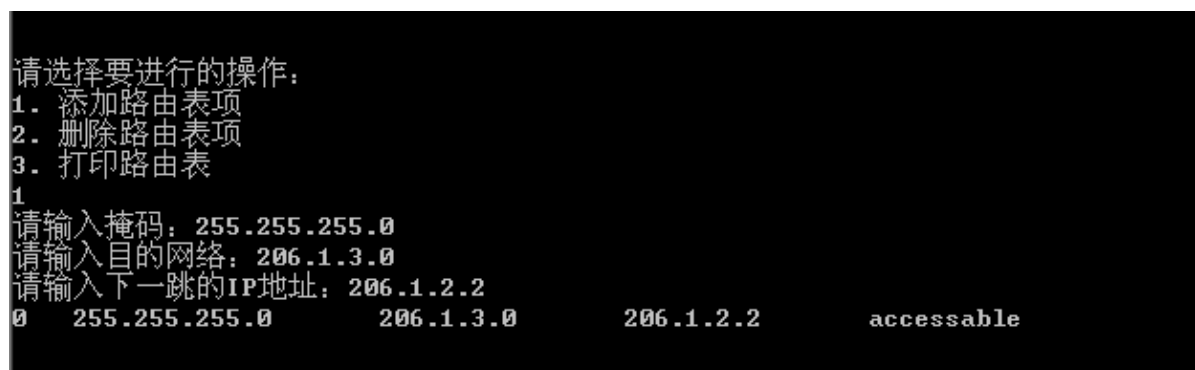
```

C:\Documents and Settings\Administrator\桌面\路由器.exe
1. pcap://Device\NPF_{3AC00148-1BFF-47AF-8441-F1D14A619031}\{ Network adapter '
Intel(R) PRO/1000 MT Network Connection' on local host >
IP地址: 206.1.2.1
网络掩码: 255.255.255.0
IP地址: 206.1.1.1
网络掩码: 255.255.255.0
-----
请选择您要打开的网卡: 1
本机双IP地址为:
206.1.2.1      255.255.255.0
206.1.1.1      255.255.255.0
源IP地址: 206.1.2.1
源MAC地址: 00-0C-29-8C-B5-E5
目的IP地址: 10.10.10.10
目的MAC地址: 0F-0F-0F-0F-0F-0F
成功获取本机MAC地址
MAC地址为: 00-0C-29-8C-B5-E5

```

- 路由表的相关操作

路由表项的添加:

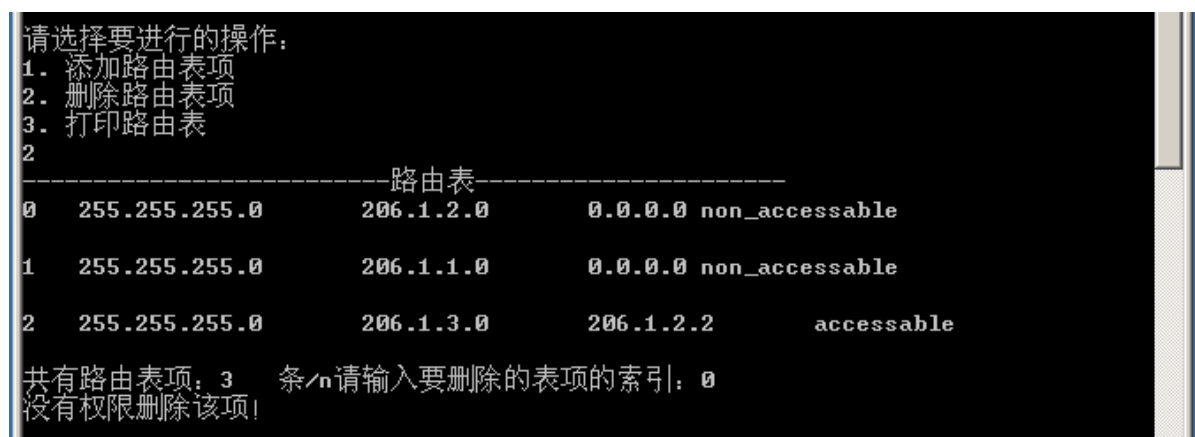


```

请选择要进行的操作:
1. 添加路由表项
2. 删除路由表项
3. 打印路由表
1
请输入掩码: 255.255.255.0
请输入目的网络: 206.1.3.0
请输入下一跳的IP地址: 206.1.2.2
0  255.255.255.0      206.1.3.0      206.1.2.2      accessible

```

添加成功后:



```

请选择要进行的操作:
1. 添加路由表项
2. 删除路由表项
3. 打印路由表
2
-----路由表-----
0  255.255.255.0      206.1.2.0      0.0.0.0 non_accessible
1  255.255.255.0      206.1.1.0      0.0.0.0 non_accessible
2  255.255.255.0      206.1.3.0      206.1.2.2      accessible

共有路由表项: 3 条
请输入要删除的表项的索引: 0
没有权限删除该项!

```

路由表项的删除:

对于直接相连的默认路由，无法删除

```
请选择要进行的操作：
1. 添加路由表项
2. 删除路由表项
3. 打印路由表
2
-----路由表-----
0   255.255.255.0      206.1.2.0      0.0.0.0 non_accessible
1   255.255.255.0      206.1.1.0      0.0.0.0 non_accessible
2   255.255.255.0      206.1.3.0      206.1.2.2      accessible

共有路由表项：3 条\n请输入要删除的表项的索引：0
没有权限删除该项！
```

其他路由，可以成功删除：

```
请选择要进行的操作：
1. 添加路由表项
2. 删除路由表项
3. 打印路由表
2
-----路由表-----
0   255.255.255.0      206.1.2.0      0.0.0.0 non_accessible
1   255.255.255.0      206.1.1.0      0.0.0.0 non_accessible
2   255.255.255.0      206.1.3.0      206.1.2.2      accessible

共有路由表项：3 条\n请输入要删除的表项的索引：2
已成功删除指定项！

请选择要进行的操作：
1. 添加路由表项
2. 删除路由表项
3. 打印路由表
3
-----路由表-----
0   255.255.255.0      206.1.2.0      0.0.0.0 non_accessible
1   255.255.255.0      206.1.1.0      0.0.0.0 non_accessible

共有路由表项：2 条\n
```

• 主机之间ping

用主机A ping 主机B，ping的TTL正确

```
C:\ 命令提示符
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:

Request timed out.
Request timed out.
Reply from 206.1.3.2: bytes=32 time=1429ms TTL=126
Reply from 206.1.3.2: bytes=32 time=1999ms TTL=126

Ping statistics for 206.1.3.2:
    Packets: Sent = 4, Received = 2, Lost = 2 (50% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 1429ms, Maximum = 1999ms, Average = 1714ms

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=1487ms TTL=126
Reply from 206.1.3.2: bytes=32 time=1996ms TTL=126
Reply from 206.1.3.2: bytes=32 time=1998ms TTL=126
Reply from 206.1.3.2: bytes=32 time=2000ms 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 = 1487ms, Maximum = 2000ms, Average = 1870ms

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

tracert正确显示三跳路径

```
C:\ 命令提示符

C:\Documents and Settings\Administrator>ping 206.5.6.3

Pinging 206.5.6.3 with 32 bytes of data:

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

Ping statistics for 206.5.6.3:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),

C:\Documents and Settings\Administrator>tracert 206.1.3.2

Tracing route to NANKAI-F3F03CED [206.1.3.2]
over a maximum of 30 hops:

    1  1857 ms  1998 ms  1998 ms  NANKAI-F3F03CED [206.1.2.2]
    2  1967 ms  2001 ms  1996 ms  NANKAI-F3F03CED [206.1.2.2]
    3  1966 ms  1999 ms  1999 ms  NANKAI-F3F03CED [206.1.3.2]

Trace complete.

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

• 工作日志的显示

```
-----获取本机MAC地址-----
发送ARP请求包-----ARP 数据包-----
源IP: 10.10.10.10
源MAC: 0F-0F-0F-0F-0F-0F
目的IP: 206.1.2.1
目的MAC: 00-00-00-00-00-00
收到ARP应答包-----ARP 数据包-----
源IP: 206.1.2.1
源MAC: 00-0C-29-8C-B5-E5
目的IP: 10.10.10.10
目的MAC: 0F-0F-0F-0F-0F-0F
-----成功获取本机MAC地址-----

接收IP数据报-----IP 数据包-----
源IP: 206.1.1.2
目的IP: 206.1.3.2
源MAC: 00-0C-29-9E-47-53
目的MAC: 00-0C-29-8C-B5-E5
接收到的IP数据报目的为206.1.3.2
接收到的IP数据报下一跳为206.1.2.2
缓存IP数据报-----IP 数据包-----
源IP: 206.1.1.2
目的IP: 206.1.3.2
源MAC: 00-0C-29-8C-B5-E5
目的MAC: 00-00-00-00-00-00
-----获取远程主机MAC地址-----
发送ARP请求包-----ARP 数据包-----
源IP: 206.1.2.1
源MAC: 00-0C-29-8C-B5-E5
目的IP: 206.1.2.2
目的MAC: 00-00-00-00-00-00
接收ARP响应包-----ARP 数据包-----
源IP: 206.1.2.2
源MAC: 00-0C-29-87-3F-15
```

```
目的MAC: 00-00-00-00-00-00
-----获取远程主机MAC地址-----
发送ARP请求包-----ARP 数据包-----
源IP: 206.1.2.1
源MAC: 00-0C-29-8C-B5-E5
目的IP: 206.1.2.2
目的MAC: 00-00-00-00-00-00

接收ARP响应包-----ARP 数据包-----
源IP: 206.1.2.2
源MAC: 00-0C-29-87-3F-15
目的IP: 206.1.2.1
目的MAC: 00-0C-29-8C-B5-E5

接收IP数据报-----IP 数据包-----
源IP: 206.1.1.2
目的IP: 206.1.3.2
源MAC: 00-0C-29-9E-47-53
目的MAC: 00-0C-29-8C-B5-E5

接收到的IP数据报目的为206.1.3.2
接收到的IP数据报下一跳为206.1.2.2-----转发数据包-----
转发-----IP 数据包-----
源IP: 206.1.1.2
目的IP: 206.1.3.2
源MAC: 00-0C-29-8C-B5-E5
目的MAC: 00-0C-29-87-3F-15

接收IP数据报-----IP 数据包-----
源IP: 206.1.3.2
目的IP: 206.1.1.2
源MAC: 00-0C-29-87-3F-15
目的MAC: 00-0C-29-8C-B5-E5

接收到的IP数据报目的为206.1.1.2
接收到的IP数据报下一跳为206.1.1.2
缓存IP数据报-----IP 数据包-----
目的MAC: 00-0C-29-87-3F-15

接收IP数据报-----IP 数据包-----
源IP: 206.1.3.2
目的IP: 206.1.1.2
源MAC: 00-0C-29-87-3F-15
目的MAC: 00-0C-29-8C-B5-E5

接收到的IP数据报目的为206.1.1.2
接收到的IP数据报下一跳为206.1.1.2
缓存IP数据报-----IP 数据包-----
源IP: 206.1.3.2
目的IP: 206.1.1.2
源MAC: 00-0C-29-8C-B5-E5
目的MAC: 00-00-00-00-00-00
-----获取远程主机MAC地址-----
发送ARP请求包-----ARP 数据包-----
源IP: 206.1.2.1
源MAC: 00-0C-29-8C-B5-E5
目的IP: 206.1.1.2
目的MAC: 00-00-00-00-00-00

接收ARP响应包-----ARP 数据包-----
源IP: 206.1.1.2
源MAC: 00-0C-29-9E-47-53
目的IP: 206.1.2.1
目的MAC: 00-0C-29-8C-B5-E5

接收IP数据报-----IP 数据包-----
源IP: 206.1.1.2
目的IP: 206.1.3.2
源MAC: 00-0C-29-9E-47-53
目的MAC: 00-0C-29-8C-B5-E5

接收到的IP数据报目的为206.1.3.2
接收到的IP数据报下一跳为206.1.2.2-----转发数据包-----
转发-----IP 数据包-----
```

可以看到从主机A连接主机B，在路由器A上的数据报接收转发完整过程。观察路由程序的日志可以发现，程序首先发送ARP请求数据报然后收到ARP应答数据报，从而成功的获得了本机MAC地址；然后收到了由主机A传送来的IP数据报，同时获得下一跳的IP地址为第二个相连路由器的IP地址，因为它的MAC地址，所以先进行缓存，在发送ARP数据报获取远程主机的MAC地址，获取成功后进行数据包的转发，之后接收到主机B传回来的IP数据报，在通过发送ARP请求获得主机A的MAC地址，成功后进行数据报的转发，完成了一次完整的主机A连接主机B的过程。

```
ay_log.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

-----获取本机MAC地址-----

发送ARP请求包-----ARP 数据包-----
源IP: 10.10.10.10
源MAC: 0F-0F-0F-0F-0F-0F
目的IP: 206.1.2.1
目的MAC: 00-00-00-00-00-00

收到ARP应答包-----ARP 数据包-----
源IP: 206.1.2.1
源MAC: 00-0C-29-8C-B5-E5
目的IP: 10.10.10.10
目的MAC: 0F-0F-0F-0F-0F-0F

-----成功获取本机MAC地址-----

接收IP数据报-----IP 数据包-----
源IP: 206.1.1.2
目的IP: 206.1.5.4
源MAC: 00-0C-29-9E-47-53
目的MAC: 00-0C-29-8C-B5-E5

接收到的IP数据报目的为206.1.5.4
接收到的IP数据报下一跳为-1
255.255.255.255
发送ICMP目的不可达数据包-->

发送ICMP目的不可达数据报!
接收IP数据报-----IP 数据包-----
源IP: 206.1.1.2
目的IP: 206.1.5.4
源MAC: 00-0C-29-9E-47-53
目的MAC: 00-0C-29-8C-B5-E5

接收到的IP数据报目的为206.1.5.4
接收到的IP数据报下一跳为-1
255.255.255.255
发送ICMP目的不可达数据包-->

发送ICMP目的不可达数据报!
接收IP数据报
```

如果我们尝试连接拓扑图上没有的网络IP地址，日志中会显示发送ICMP目的不可达数据报。

五. 实验总结与反思

本次实验基于网络技术与应用课程的前两次编程作业：捕获数据报程序和获取ARP的程序组建的路由程序，通过捕获数据报和arp获取mac地址实现数据报的转发。

本次“简单的路由程序设计”实验使我对路由表的结构和IP，ARP,ICMP数据报的报文格式有了更清晰的认识，同时对网络传输协议，数据报的转发，以及路由协议和算法有了更深刻的理解，学习了路由转发的机制，各种协议在不同情况下的使用，对网络中两台远程主机的联通过程消息传输过程有了了解。随着这个程序的完成，网络技术与应用课程也结束了，在这门课上，我收获很多，感谢老师和助教学长平常的帮助！