设计性大作业（1） 简单路由器程序的设计

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。

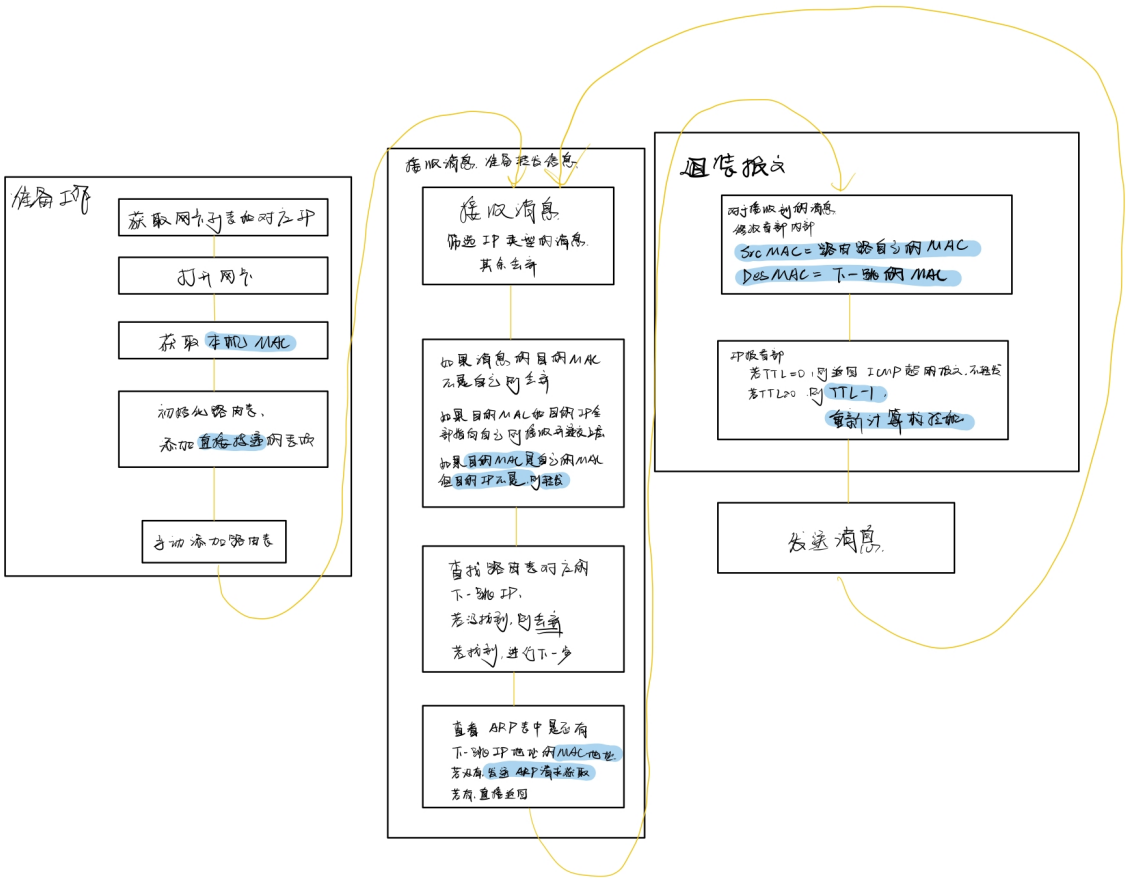
（4）需要给出路由器的工作日志，显示数据报获取和转发过程。

（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。

第1节 实验步骤

实验设计思路

路由器工作流程



具体工作流程详见下一部分的代码解析

关键代码分析

报文格式

```
#pragma pack(1)//以1byte方式对齐
#pragma pack()//恢复4bytes对齐
```

报文首部

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

ARP报文格式

```
typedef struct ARPFrame_t {
    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
    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报文首部

```
typedef struct IPHeader_t {//IP首部
    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;
```

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

ICMP报文格式

由于本次实验只对ICMP数据报进行简单解析，不需要复杂操作，所以只用char*表示即可

```
typedef struct ICMP {//包含帧首部和IP首部的数据包
    FrameHeader_t FrameHeader;
    IPHeader_t IPHeader;
    char buf[0x80];
}ICMP_t;
```

存储结构

路由表表项

```
class routeitem
{
public:
    DWORD mask;//掩码
    DWORD net;//目的网络
    DWORD nextip;//下一跳
    int index;//第几条
    int type;//0为直接连接，1为用户添加，1不可删除
    routeitem* nextitem;//采用链表形式存储
    routeitem()
    {
        memset(this, 0, sizeof(*this));//初始化为全0
    }
    void printitem();//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）
};
```

路由表

```
class routetable
{
public:
    routeitem* head, * tail;//支持最多添加50转发表
    int num;//条数
    routetable();//初始化，添加直接连接的网络

    //路由表的添加，直接投递在最前，前缀长的在前面
    void add(routeitem* a);

    //删除，type=0不能删除
    void remove(int index);
    //路由表的打印 mask net next type
    void print();
    //查找，最长前缀,返回下一跳的ip
    DWORD lookup(DWORD ip);

};
```

```
routetable::routetable()//初始化，添加直接连接的网络
{
    head = new routeitem;
    tail = new routeitem;
    head->nextitem = tail;
    num = 0;
    for (int i = 0; i < 2; i++)
```

```
    {
        routeitem* temp = new routeitem;
        temp->net = (inet_addr(ip[i])) & (inet_addr(mask[i]));//本机网卡的ip和掩码进行按位与即为所在网络
        temp->mask = inet_addr(mask[i]);
        temp->type = 0;//0表示直接投递的网络，不可删除
        this->add(temp);//添加表项
    }
}

```

ARP表

为减少发送ARP请求的次数，将IP和MAC的对应关系存储在一张表里，在实际情况中需要设置表项的生命周期，防止一段时间后个别表项IP和MAC出现不对应的情况，本次实验较为简单，故没有设置

```
class arptable
{
public:
    DWORD ip;//IP
    BYTE mac[6];//MAC
    static int num;//表项个数
    static void insert(DWORD ip,BYTE mac[6]);//插入
    static int lookup(DWORD ip, BYTE mac[6]);//查询
}atable[50];
```

日志

本次实验需要输出路由器的工作日志

```
class log
{
public:
    int index;//索引
    char type[5];//arp和ip
    //具体内容
    ipitem ip; arpitem arp;

    log();//打开文件进行写入
    ~log();//关闭文件

    static int num;//数量
    static log diary[50];//日志
    static FILE* fp;
    //写入日志
    static void write2log_arp(ARPFrame_t*);//arp类型
    static void write2log_ip(const char* a,Data_t*);//ip类型

    static void print();//打印日志
};
```

```
class arpitem//arp日志为ip和对应MAC
{
public:
    DWORD ip;
    BYTE mac[6];
};

class ipitem
{
public:
    DWORD sip, dip;//ip日志为源和目的的IP和mac
    BYTE smac[6], dmac[6];
};
```

主要函数

```
//获取自己的IP
void find_alldevs();    //获取本机的设备列表，将两个ip存入ip数组中,获取IP、mask，计算所在网段
DWORD getnet(DWORD ip, DWORD mask);//根据ip和掩码计算所在网络
//打开网络接口
pcap_t* open(char* name);
//获取自己的MAC
void getselfmac(DWORD ip);
//获取直接连接的网卡mac
void getothermac(DWORD mask_, DWORD ip, BYTE mac[]);
//显示基本信息 本机ip，mac
void printbasicinfo();

//数据报转发,修改源mac和目的mac
void resend(ICMP_t, BYTE dmac[]);

//打印mac
void getmac(BYTE MAC[]);

//线程函数
DWORD WINAPI handlerRequest(LPVOID lparam);

void ipprint(DWORD ip);

//检查和设置校验和
bool checkchecksum(Data_t*);
void setchecksum(Data_t*);
```

find_alldevs获取本机网卡的IP

```
void find_alldevs() //获取网卡上的IP
{
    //获取网卡列表
    if (pcap_findalldevs_ex(pcap_src_if_string, NULL, &alldevs, errbuf) == -1)
    {
        printf("%s", "error");
    }
    else
```

```
    {
        int i = 0;
        d = alldevs;
        //获取该网络接口设备的ip地址信息
        for (; d != NULL; d = d->next)
        {
            if (i == index)
            {
                net[i] = d;
                int t = 0;
                for (a = d->addresses; a != nullptr; a = a->next)
                {
                    if (((struct sockaddr_in*)a->addr)->sin_family == AF_INET && a->addr)
                    {
                        //将对应第index块网卡的内容存入全局数组
                        strcpy(ip[t], inet_ntoa(((struct sockaddr_in*)a->addr)->sin_addr));
                        strcpy(mask[t++], inet_ntoa(((struct sockaddr_in*)a->netmask)->sin_addr));
                    }
                }
                ahandle = open(d->name);//打开该网卡
            }
            i++;
        }
    }
    pcap_freealldevs(alldevs);
}
```

open打开网络接口

```
pcap_t* open(char* name)//打开网络接口，返回网卡指针
{
    pcap_t* temp = pcap_open(name, 65536, PCAP_OPENFLAG_PROMISCUOUS, 100, NULL, errbuf);
    if (temp == NULL)
        printf("error");
    return temp;
}
```

getselfmac和getothermac获取MAC地址

由于获取本机的MAC和其它机器的MAC地址逻辑相似，这里只详细说明一个

```
void getothermac(DWORD ip_, BYTE mac[])//获取ip对应的mac
{
    memset(mac, 0, sizeof(mac));
    ARPFrame_t ARPFrame;
    //将APRFrame.FrameHeader.DesMAC设置为广播地址
    for (int i = 0; i < 6; i++)
        ARPFrame.FrameHeader.DesMAC[i] = 0xff;
    //将APRFrame.FrameHeader.SrcMAC设置为本机网卡的MAC地址，若获取本机MAC，则伪造一个即可
    for (int i = 0; i < 6; i++)
    {
        ARPFrame.FrameHeader.SrcMAC[i] = selfmac[i];
```

```
        ARPFrame.SendHa[i] = selfmac[i];

    }

    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.SendIP设置为本机网卡上绑定的IP地址，若获取本机MAC，则伪造发送IP和MAC
    ARPFrame.SendIP = inet_addr(ip[0]);
    //ipprint(ARPFrame.SendIP);
    //将ARPFrame.RecvHa设置为0
    for (int i = 0; i < 6; i++)
        ARPFrame.RecvHa[i] = 0;
    //将ARPFrame.RecvIP设置为请求的IP地址
    ARPFrame.RecvIP = ip_;

    u_char* h = (u_char*)&ARPFrame;
    int len = sizeof(ARPFrame_t);

    if (ahandle == nullptr) printf("网卡接口打开错误\n");
    else
    {
        if (pcap_sendpacket(ahandle, (u_char*)&ARPFrame,
sizeof(ARPFrame_t)) != 0)
        {
            //发送错误处理
            printf("senderror\n");
        }
        else
        {
            //发送成功
            while (1)
            {
                pcap_pkthdr* pkt_header;
                const u_char* pkt_data;
                int rtn = pcap_next_ex(ahandle, &pkt_header, &pkt_data);//捕
获数据报
                if (rtn == 1)//捕获到数据报
                {
                    ARPFrame_t* IPPacket = (ARPFrame_t*)pkt_data;
                    if (ntohs(IPPacket->FrameHeader.FrameType) == 0x806)//筛
选ARP类型的消息进行处理
                    {
                        if (!compare(IPPacket->FrameHeader.SrcMAC,
ARPFrame.FrameHeader.SrcMAC) && compare(IPPacket->FrameHeader.DesMAC,
ARPFrame.FrameHeader.SrcMAC) && IPPacket->SendIP == ip_)//消息筛选
                        {
                            ltable.write2log_arp(IPPacket);
                            //源MAC地址即为所需MAC地址
                            for (int i = 0; i < 6; i++)
                            {
                                mac[i] = IPPacket->FrameHeader.SrcMAC[i];
                            }
                            break;//已经捕获到MAC，可以退出函数
```

```
                        }

                    }
                }
            }
        }
    }
}

```

setchecksum和checkchecesum设置校验和和检验校验和

```
void setchecksum(Data_t* temp)//设置校验和
{
    temp->IPHeader.Checksum = 0;
    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)//如果溢出，则进行回卷
        {
            int s = sum >> 16;
            sum -= 0x10000;
            sum += s;
        }
    }
    temp->IPHeader.Checksum = ~sum;//结果取反
}
```

```
bool checkchecksum(Data_t* temp)//检验
{
    unsigned int sum = 0;
    WORD* t = (WORD*)&temp->IPHeader;
    for (int i = 0; i < sizeof(IPHeader_t) / 2; i++)
    {
        sum += t[i];
        while (sum >= 0x10000)//包含原有校验和一起进行相加
        {
            int s = sum >> 16;
            sum -= 0x10000;
            sum += s;
        }
    }
    if (sum == 65535)//源码+反码-》全1
        return 1;//校验和正确
    return 0;
}
```

接收和处理线程函数

为使消息转发和路由表添加、删除、打印等操作可以同时进行，使用线程函数进行消息内容处理

```
DWORD WINAPI handlerRequest(LPVOID lparam)
{
    routetable rtable = *(routetable*)(LPVOID)lparam;
```

```
    while (1)
    {
        pcap_pkthdr* pkt_header; const u_char* pkt_data;
        while (1)
        {
            int rtn = pcap_next_ex(ahandle, &pkt_header, &pkt_data);
            if (rtn)break;//接收到消息
        }
        FrameHeader_t* header = (FrameHeader_t*)pkt_data;
        if (compare(header->DesMAC, selfmac))//目的mac是自己的mac
        {
            else if (ntohs(header->FrameType) == 0x800)//IP格式的数据报
            {
                Data_t* data = (Data_t*)pkt_data;
                ltable.write2log_ip("接收", data);//将接收内容写入日志

                DWORD ip1_ = data->IPHeader.DstIP;
                DWORD ip_ = rtable.lookup(ip1_);//查找路由表中是否有对应表项
                if(ip_==-1)continue;//如果没有则直接丢弃或直接递交至上层
                if (checkchecksum(data))//如果校验和不正确，则直接丢弃不进行处理
                {
                    if (data->IPHeader.DstIP != inet_addr(ip[0]) && data->IPHeader.DstIP != inet_addr(ip[1]))
                    {
                        //不是广播消息
                        int t1 = compare(data->FrameHeader.DesMAC, broadcast);
                        int t2 = compare(data->FrameHeader.SrcMAC, broadcast);
                        if (!t1 && !t2)
                        {
                            //ICMP报文包含IP数据包报头和其它内容
                            ICMP_t* temp_ = (ICMP_t*)pkt_data;
                            ICMP_t temp = *temp_;
                            BYTE mac[6];
                            if(ip_==0)//直接投递，查找目的IP的MAC
                            {
                                //如果ARP表中没有所需内容，则需要获取ARP
                                if (!arptable::lookup(ip1_, mac))
                                    arptable::insert(ip1_, mac);
                                resend(temp, mac);//转发
                            }

                            else if (ip_ != -1)//非直接投递，查找下一条IP的MAC
                            {
                                if (!arptable::lookup(ip_, mac))
                                    arptable::insert(ip_, mac);
                                resend(temp, mac);
                            }
                        }

                    }
                }
            }
        }
    }
}
```

resend消息转发

```
//数据报转发,修改源mac和目的mac
void resend(ICMP_t data, BYTE dmac[])
{
    Data_t* temp=(Data_t*)&data;
    memcpy(temp->FrameHeader.SrcMAC, temp->FrameHeader.DesMAC, 6);//源MAC为本机MAC
    memcpy(temp->FrameHeader.DesMAC, dmac, 6);//目的MAC为下一跳MAC
    temp->IPHeader.TTL -= 1;//TTL-1
    if (temp->IPHeader.TTL < 0)return;//丢弃
    setchecksum(temp);//重新设置校验和
    int rtn = pcap_sendpacket(ahandle, (const u_char*)temp, 74);//发送数据报
    if (rtn == 0)
        ltable.write2log_ip("转发",temp);//写入日志
}
```

ip打印

```
void ipprint(DWORD ip)
{
    in_addr addr;
    addr.s_addr = ip;
    char* pchar = inet_ntoa(addr);
    printf("%s\t", pchar);
    printf("\n");
}
```

getmac打印MAC

```
void getmac(BYTE MAC[])//打印mac
{
    printf("MAC地址为:  ");
    for (int i = 0; i < 5; i++)
        printf("%02X-", MAC[i]);
    printf("%02X\n", MAC[5]);
}
```

路由表添加

为方便寻找**最长前缀表项**，在插入时对掩码进行排序

```
void routetable::add(routeitem* a)
{
    routeitem* pointer;
    //找到合适的地方
    //默认路由,一定是最开始的时候添加
    if (!a->type)//直接投递
    {
        a->nextitem = head->nextitem;
        head->nextitem = a;
        a->type = 0;
    }

    //其它，按照掩码由长至短找到合适的位置
```

```
    else
    {
        for (pointer = head->nextitem; pointer != tail && pointer->nextitem
!= tail; pointer = pointer->nextitem)//head有内容，tail没有
        {
            if (a->mask < pointer->mask && a->mask >= pointer->nextitem-
>mask || pointer->nextitem == tail)
                break;
        }
        a->nextitem = pointer->nextitem;
        pointer->nextitem = a;//插入到合适位置
        //a->type = 1;
    }
    routeitem* p = head->nextitem;
    for (int i = 0; p != tail; p = p->nextitem, i++)
    {
        p->index = i;
    }
    num++;
}
```

路由表删除

```
void routetable::remove(int index)
{
    for (routeitem* t = head; t->nextitem != tail; t = t->nextitem)
    {
        if (t->nextitem->index == index)
        {
            if (t->nextitem->type == 0)
            {
                printf("该项不可删除\n");
                return;
            }
            else
            {
                t->nextitem = t->nextitem->nextitem;
                return;
            }
        }
    }
    printf("无该表项\n");
}
```

实验结果

已经给助教检查过，不再附结果截图

遇到的问题

- 收到ping数据报但是不回复

    - 可能使校验和设置错误，消息被接收方丢弃了
- 第一条能ping通，后面三条显示超时
    - 处理了大量其它的消息，占用CPU资源，将筛选提前
- ARP请求到的MAC是其它电脑的MAC
    - 注意筛选收到的ARP请求的发送方IP
- ping发送方显示无法访问目标网络
    - 打开运行路由器的主机中的routing and remote服务