网络技术与应用课程报告

第五次实验报告

学号：2013921
姓名：周延霖
年级：2020级
专业：信息安全

一、实验内容说明

简单路由器程序的设计

要求如下：

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、前期准备

(1)NPcap架构

Npcap是致力于采用`Microsoft Light-Weight Filter (NDIS 6 LWF)`技术和`Windows Filtering Platform (NDIS 6 WFP)`技术对当前最流行的WinPcap工具包进行改进的一个项目。Npcap项目是最初2013年由 Nmap网络扫描器项目（创始人Gordon Lyon）和 北京大学罗杨博士发起，由Google公司的Summer of Code计划赞助的一个开源项目，遵循MIT协议（与WinPcap一致）。

Npcap基于WinPcap 4.1.3源码基础上开发，支持32位和64位架构，在`Windows Vista`以上版本的系统中，采用NDIS 6技术的Npcap能够比原有的WinPcap数据包（NDIS 5）获得更好的抓包性能，并且稳定性更好。

NPcap提供两个不同的库：packet.dll与npcap.dll。第一个库提供一个底层的API，可用来直接访问驱动程序的函数，提供一个独立于微软的不同操作系统的编程接口。第二个库导出了更强大的、更高层的捕获函数接口，并提供与UNIX捕获库libpcap的兼容性。这些函数使得数据包的捕获能独立于底层网络硬件与操作系统。

大多数网络应用程序通过被广泛使用的操作系统元件来访问网络，比如 sockets——这是一种简单的实现方式，因为操作系统已经妥善处理了底层具体实现细节（比如协议处理，封装数据包等等工作），并且提供了一个与读写文件类似的，令人熟悉的接口；但是有些时候，这种“简单的实现方式”并不能满足需求，因为有些应用程序需要直接访问网络中的数据包：也就是说原始数据包——即没有被操作系统利用网络协议处理过的数据包。而 NpCap 则为 Win32 应用程序提供了这样的接口：

- 捕获原始数据包：无论它是发往某台机器的，还是在其他设备（共享媒介）上进行交换的
- 在数据包发送给某应用程序前，根据指定的规则过滤数据包
- 将原始数据包通过网络发送出去
- 收集并统计网络流量信息

(2)NPcap捕获数据包

> 设备列表获取方法——`pcap_findalldevs_ex`

NpCap 提供了 pcap_findalldevs_ex 和 pcap_findalldevs 函数来获取计算机上的网络接口设备的列表；此函数会为传入的 pcap_if_t 赋值——该类型是一个表示了设备列表的链表头；每一个这样的节点都包含了 name 和 description 域来描述设备。

除此之外，pcap_if_t 结构体还包含了一个 pcap_addr 结构体；后者包含了一个地址列表、一个掩码列表、一个广播地址列表和一个目的地址的列表；此外，pcap_findalldevs_ex 还能返回远程适配器信息和一个位于所给的本地文件夹的 pcap 文件列表。

> 网卡设备打开方法——`pcap_open`

用来打开一个适配器，实际调用的是 pcap_open_live；它接受五个参数：

- name：适配器的名称（GUID）
- snaplen：制定要捕获数据包中的哪些部分。在一些操作系统中 (比如 xBSD 和 Win32)，驱动可以被配置成只捕获数据包的初始化部分：这样可以减少应用程序间复制数据的量，从而提高捕获效率；本次实验中，将值定为 65535，比能遇到的最大的 MTU 还要大，因此总能收到完整的数据包。
- flags：主要的意义是其中包含的混杂模式开关；一般情况下，适配器只接收发给它自己的数据包， 而那些在其他机器之间通讯的数据包，将会被丢弃。但混杂模式将会捕获所有的数据包——因为我们需要捕获其他适配器的数据包，所以需要打开这个开关。
- to_ms：指定读取数据的超时时间，以毫秒计；在适配器上使用其他 API 进行读取操作的时候，这些函数会在这里设定的时间内响应——即使没有数据包或者捕获失败了；在统计模式下，to_ms 还可以用来定义统计的时间间隔：设置为 0 说明没有超时——如果没有数据包到达，则永远不返回；对应的还有 -1：读操作立刻返回。
- errbuf：用于存储错误信息字符串的缓冲区

该函数返回一个 pcap_t 类型的 handle。

> 数据包捕获方法——`pcap_loop`

虽然在课本上演示用的是 pcap_next_ex 函数，但是他并不会使用回调函数，不会把数据包传递给应用程序，所以在本次实验中我采取的是 pcap_loop 函数。

API 函数 pcap_loop 和 pcap_dispatch 都用来在打开的适配器中捕获数据包；但是前者会已知捕获直到捕获到的数据包数量达到要求数量，而后者在到达了前面 API 设定的超时时间之后就会返回（尽管这得不到保证）；前者会在一小段时间内阻塞网络的应用，故一般项目都会使用后者作为读取数据包的函数；虽然在本次实验中，使用前者就够了。

这两个函数都有一个回调函数；这个回调函数会在这两个函数捕获到数据包的时候被调用，用来处理捕获到的数据包；这个回调函数需要遵顼特定的格式。但是需要注意的是我们无法发现 CRC 冗余校验码——因为帧到达适配器之后，会经过校验确认的过程；这个过程成功，则适配器会删除 CRC；否则，大多数适配器会删除整个包，因此无法被 NpCap 确认到。

(3)ARP协议

静态映射

静态映射的意思是要手动创建一张ARP表，把逻辑IP地址和物理地址关联起来。这个ARP表储存在网络中的每一台机器上。例如，知道其机器的IP地址但不知道其物理地址的机器就可以通过查ARP表找出对应的物理地址。这样做有一定的局限性，因为物理地址可能发生变化：

1. 机器可能更换NIC（网络适配器），结果变成一个新的物理地址。
2. 在某些局域网中，每当计算机加电时，他的物理地址都要改变一次。
3. 移动电脑可以从一个物理网络转移到另一个物理网络，这样会时物理地址改变。

要避免这些问题出现，必须定期维护更新ARP表，此类比较麻烦而且会影响网络性能。

动态映射

动态映射时，每次只要机器知道另一台机器的逻辑（IP）地址，就可以使用协议找出相对应的物理地址。已经设计出的实现了动态映射协议的有ARP和RARP两种。ARP把逻辑（IP）地址映射为物理地址。RARP把物理地址映射为逻辑（IP）地址。

ARP请求

任何时候，当主机需要找出这个网络中的另一个主机的物理地址时，它就可以发送一个ARP请求报文，这个报文包好了发送方的MAC地址和IP地址以及接收方的IP地址。因为发送方不知道接收方的物理地址，所以这个查询分组会在网络层中进行广播。

ARP响应

局域网中的每一台主机都会接受并处理这个ARP请求报文，然后进行验证，查看接收方的IP地址是不是自己的地址，只有验证成功的主机才会返回一个ARP响应报文，这个响应报文包含接收方的IP地址和物理地址。这个报文利用收到的ARP请求报文中的请求方物理地址以单播的方式直接发送给ARP请求报文的请求方。

报文格式

- ARP报文格式大致如下图所示：

- 解释如下：

硬件类型： 16位字段，用来定义运行ARP的网络类型。每个局域网基于其类型被指派一个整数。例如：以太网的类型为1。ARP可用在任何物理网络上。

协议类型： 16位字段，用来定义使用的协议。例如：对IPv4协议这个字段是0800。ARP可用于任何高层协议

硬件长度： 8位字段，用来定义物理地址的长度，以字节为单位。例如：对于以太网的值为6。

协议长度： 8位字段，用来定义逻辑地址的长度，以字节为单位。例如：对于IPv4协议的值为4。

操作码： 16位字段，用来定义报文的类型。已定义的分组类型有两种：ARP请求（0x0001），ARP响应（0x0002）。

源硬件地址： 这是一个可变长度字段，用来定义发送方的物理地址。例如：对于以太网这个字段的长度是6字节。

源逻辑地址： 这是一个可变长度字段，用来定义发送方的逻辑（IP）地址。例如：对于IP协议这个字段的长度是4字节。

目的硬件地址： 这是一个可变长度字段，用来定义目标的物理地址，例如，对以太网来说这个字段位6字节。对于ARP请求报文，这个字段为全0，因为发送方并不知道目标的硬件地址。

目的逻辑地址： 这是一个可变长度字段，用来定义目标的逻辑（IP）地址，对于IPv4协议这个字段的长度为4个字节。

(4)路由器

路由器定义：

路由器（Router）是在网络层实现网络互连，可实现网络层、链路层和物理层协议转换。也就是说，路由器是一种利用协议转换技术将异种网进行互联的设备。

路由器的用途：

1. 局域网之间的互连

- 利用路由器也可以互连不同类型的局域网。

2. 局域网与广域网（WAN）之间的互连

- 局域网与WAN互连时，使用较多的互连设备是路由器。路由器能完成局域网与WAN低三层协议的转换。路由器的档次很多，其端口数从几个到几十个不等，所支持的通信协议也可多可少。
- 其实，局域网与广域网之间的互连主要为了实现是通过广域网对两个异地局域网进行互连。用于连接局域网的广域网可以是分组交换网、帧中继网或ATM网等。

3. WAN与WAN之间的互连

- 利用路由器互连WAN，要求两个WAN只是低三层协议不同。

路由器的功能：

1. 选择最佳传输路由

- 路由器涉及OSI-RM的低三层，当分组到达路由器，先在组合队列中排队，路由器依次从队列中取出分组，查看托组中的目的地址，然后再查路由表。
- 一般到达目的站点前可能有多条路由，路由器应按照某种路由选择策略，从中选出一条最佳路由，将分组转发出去。
- 当网络拓扑发生变化时，路由器还可自动调整路由表，并使所选择的路由仍然是最佳的。这一功能还可以很好地均衡网络中的信息流量，避免出现网络拥挤现象。

2. 实现IP、ICMP、TCP、UDP等互联网协议

- 作为IP网的核心设备，路由器应该可以实现IP、ICMP、TCP、UDP等互联网协议。

3. 流量控制和差错指示

- 在路由器中具有较大容量的缓冲区，能控制收发双方间的数据流量，使两者更加匹配。而且当分组出现差错时，路由器能够辨认差错并发送ICMP差错报文报告必要的差错信息。

4. 分段和重新组装功能

- 由路由器所连接的多个网络，它们所采取的分组大小可能不同，需要分段和重组。

5. 提供网络管理和系统支持机制

- 包括存储/上载配置、诊断、升级、状态报告、异常情况报告及控制。

三、实验过程

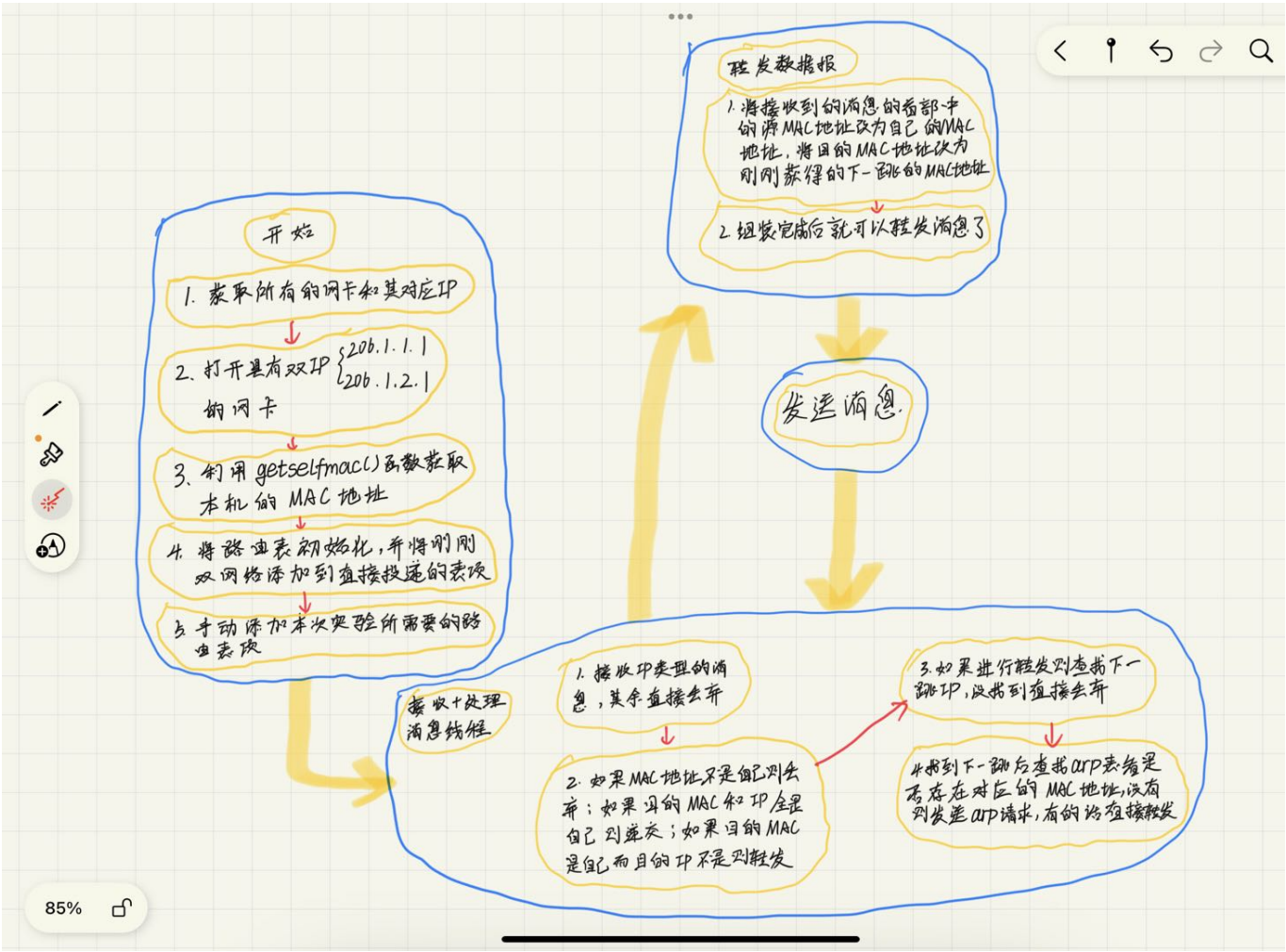
---

(1)项目设计思路

首先需要了解路由器的转发过程，按照IP路由选择算法，在利用WinPcap获取到需要转发的IP数据报之后，路由处理软件首先需要提取该报文的目的IP地址，并通过路由表为其进行路由选择。如果路由选择成功，则记录需要投递到的下一路由器地址；如果不成功，则简单地将该报文抛弃。

在将路由选择成功的IP数据报发送到相应的接口之前，首先需要利用ARP获取下一站路由器接口的MAC地址。一旦得到下一站路由器的MAC地址，就可以把IP数据报封装成数据帧并通过相应的接口发送出去。其实这个实验可以看作前两个实验的继续。

根据以上对这个大项目可以设计出如下的流程图：



本次实验也是需要获取本机网卡中对应IP的MAC地址，可以利用ARP请求方法，**过程如下：**

- 获取网络接口卡列表，选择需要捕获MAC地址的网卡A（或选择对应的IP）
- 伪造ARP请求报文S，内容要求如下：
  - ARP请求
  - 广播
  - 伪造源MAC地址和源IP地址
  - 目的IP地址为网卡A的IP地址
- 用网卡A发送报文S'
- 对网卡A进行流量监听，筛选其中的ARP报文（类型为0x806），捕获网卡A的ARP响应报文，在响应报文的帧首部源MAC地址部分可以看到发送该ARP响应的网卡对应的MAC地址

对应目的ip主机的MAC地址的获取与以上思路类似，只不过源地址变为自己即可。

以上即为主要思路，接下来进行关键代码分析。

(2)关键代码分析

按照项目设计思路开始编写代码，首先是各类头文件的定义：

> 注意：对其格式一定要对，一定要有#pragma pack(1)

- 帧首部

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

- ARP报文和IP报文

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

typedef struct IPHeader_t
{
    BYTE Ver_HLen;
    BYTE TOS;
    WORD TotalLen;
```

```
    WORD ID;
    WORD Flag_Segment;
    // (time to live)
    BYTE TTL;
    // 协议
    BYTE Protocol;
    // 校验和
    WORD Checksum;
    // 源IP
    ULONG SrcIP;
    // 目的IP
    ULONG DstIP;
}IPHeader_t;
```

- 路由器表项：包括掩码、目的网络等信息，方便之后对整体的操作

```
class routeitem
{
public:
    // 掩码
    DWORD mask;
    // 目的网络
    DWORD net;
    // 下一跳
    DWORD nextip;
    // 下一跳的MAC地址
    BYTE nextMAC[6];
    // 序号
    int index;
    // 0为直接相连 1为用户添加（直接相连 不可删除）
    int type;
    routeitem* nextitem;
    routeitem()
    {
        // 将其全部设置为零
        memset(this, 0, sizeof(*this));
    }
    // 打印掩码、目的网络、下一跳IP、类型
    void printitem();
};
```

- 路由表：采用的结构体是链表的形式

```
class routetable
{
public:
    // 最多可以又50个条目
    routeitem* head, * tail;
    // 目前存在的个数
```

```
    int num;
    routetable();
    // 添加 （直接投递在最前 接着最长匹配 长的在前）
    void add(routeitem* a);
    //删除 type=0（直接相连）不能删除
    void remove(int index);
    //路由表的打印`mask net next_ip type`
    void print();
    //查找 （最长前缀 返回下一跳的`ip`地址）
    DWORD lookup(DWORD ip);
};
```

- 然后定义arp表项和arp表，arp表采用的是数组的形式

```
class arpitem
{
public:
    DWORD ip;
    BYTE mac[6];
};
class ipitem
{
public:
    DWORD sip, dip;
    BYTE smac[6], dmac[6];
};

// arp表 存储已经得到的arp关系
class arptable
{
public:
    // IP地址
    DWORD ip;
    // MAC地址
    BYTE mac[6];
    // 表项数量
    static int num;
    // 插入表项
    static void insert(DWORD ip, BYTE mac[6]);
    // 删除表项
    static int lookup(DWORD ip, BYTE mac[6]);
}atable[50];
```

- 然后是日志的定义，方便将其转发信息写入日志

```
//日志类
class zyl_log
{
public:
```

```
    // 索引
    int index;

    // arp和ip
    char type[5];

    // 具体内容
    ipitem ip;
    arpitem arp;

    zyl_log();
    ~zyl_log();

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

    static void print();
};
```

- 主函数首先先获取设备列表，然后选择双网卡的设备进行打开，并将刚刚获取的双IP输出，然后利用arp协议获取本机的mac地址，然后打开一个接收数据报的线程，实现满足要求的数据报就进行转发，在主程序中运行一个循环，可以实现添加、删除以及显示路由表项的操作。

```
int main()
{
    // const char* 到char*的转换 解决vs中的报错问题
    pcap_src_if_string = new char[strlen(PCAP_SRC_IF_STRING)];
    strcpy(pcap_src_if_string, PCAP_SRC_IF_STRING);

    // 获取本机ip
    find_alldevs();

    //输出此时存储的IP地址与MAC地址
    for (int i = 0; i < 2; i++)
    {
        printf("%s\t", ip[i]);
        printf("%s\n", mask[i]);
    }

    cout << "本机MAC地址为:  ";
    getselfmac(inet_addr(ip[0]));
    getmac(selfmac);
    BYTE mac[6];
    int op;
    routetable zyl_table;
```

```
    hThread = CreateThread(NULL, NULL, handlerRequest, LPVOID(&zyl_table),
0, &dwThreadId);
    routeitem a;

    while (true)
    {
        // 进行简介
        cout <<
"================================================================================
===========" << endl;
        cout << "粥小霖的路由器，你可以输入以下数字进行相应操作：" << endl;
        cout << "1. 添加路由表项" << endl;
        cout << "2. 删除路由表项" << endl;
        cout << "3. 打印路由表：" << endl;
        cout << "4. 退出程序" << endl;
        cout <<
"================================================================================
===========" << endl;

        // 输入想要进行的操作
        scanf("%d", &op);
        if (op == 1)
        {
            routeitem a;
            char t[30];

            cout << "请输入网络掩码：" << endl;
            scanf("%s", &t);
            a.mask = inet_addr(t);

            cout << "请输入目的网络`ip`地址：" << endl;
            scanf("%s", &t);
            a.net = inet_addr(t);

            cout << "请输入下一跳`ip`地址：" << endl;
            scanf("%s", &t);
            a.nextip = inet_addr(t);

            // 手动添加的类型
            a.type = 1;
            zyl_table.add(&a);
        }
        else if (op == 2)
        {
            cout << "请输入你想要删除的表项编号：" << endl;
            int index;
            scanf("%d", &index);
            zyl_table.remove(index);
        }
        else if (op == 3)
        {
            zyl_table.print();
        }
        else if (op == 4)
```

```
        {
            break;
        }
        else {
            cout << "请输入正确的操作号! " << endl;
        }
    }
    return 0;
}
```

接下来分别讲一些重点函数

- 打开设备的函数与之前类似，先获取所有设备并展示在程序中，本次选择双网卡的ip接口打开，具体代码如下所示：

```
// 获取网卡上的IP
void find_alldevs()
{
    if (pcap_findalldevs_ex(pcap_src_if_string, NULL, &alldevs, errbuf) ==
-1)
    {
        cout << "无法获取本机设备" << endl;
        // 释放设备列表
        pcap_freealldevs(alldevs);
    }
    else
    {
        // 显示接口列表描述和对应ip地址
        int num = 0;
        int t = 0;
        for (d = alldevs; d != NULL; d = d->next)
        {
            cout <<
"================================================================================
================" << endl;
            // 设备数加一
            num++;
            // 获取网络接口设备名字
            cout << dec << num << ":" << d->name << endl;
            // 用d->description获取描述信息
            if (d->description != NULL)
            {
                cout << d->description << endl;
            }
            else
            {
                cout << "没有相关描述" << endl;
            }
            // 网络适配器的地址
            pcap_addr_t* a;
            for (a = d->addresses; a != NULL; a = a->next)
```

```
            {
                switch (a->addr->sa_family)
                {
                    // IPV4类型地址
                case AF_INET:
                    printf("Address Family Name:AF_INET\t");
                    if (a->addr != NULL)
                    {
                        // 打印IP地址
                        printf("%s\t%s\n", "IP地址:", inet_ntoa(((struct
sockaddr_in*)a->addr)->sin_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));
                    }
                    break;
                    // IPV6类型地址
                case AF_INET6:
                    //if (a->addr != NULL)
                    //{
                    //  // 打印IP地址
                    //  printf("%s\t%s\n", "IP地址:", inet_ntoa(((struct
sockkaddr_in*)a->addr)->sin_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));
                    //}
                    cout << "其地址类型为IPV6" << endl;
                    break;
                default:
                    break;
                }
            }
            cout <<
"================================================================================
================" << endl;
        }

        // 没有接口直接返回
        if (num == 0)
        {
            cout << "不可选择接口" << endl;
        }

        // 用户选择接口
        cout << "请选择你想打开的接口: " << "`1 ~ " << num << "`:" << endl;
        num = 0;
        int n;
        cin >> n;
```

```
            // 跳转到相应接口
            for (d = alldevs; num < (n - 1); num++)
            {
                d = d->next;
            }
            // n或n-1后续再调
            net[n - 1] = d;
            ahandle = open(d->name);
        }
        pcap_freealldevs(alldevs);
    }
```

然后是利用arp协议获取本机mac地址，与arp实验类似，请求本机网络接口上绑定的IP地址与MAC地址的对应关系：

1. 本地主机模拟一个远端主机，发送一个ARP请求报文，改请求报文请求本机网络接口上绑定的IP地址与MAC地址的对应关系
2. 在组装报文过程中，源MAC地址字段和源IP地址字段需要使用虚假的MAC地址和虚假的IP地址
3. 本次实验使用66-66-66-66-66-66作为源MAC地址，使用112.112.112.112作为源IP地址
4. 本地主机一旦获取该ARP请求，就会做出响应

实现的具体代码如下：

```
void getselfmac(DWORD ip)
{
    memset(selfmac, 0, sizeof(selfmac));
    ARPFrame_t ARPFrame;

    // 设置目的地址为广播地址
    for (int i = 0; i < 6; i++)
    {
        ARPFrame.FrameHeader.DesMAC[i] = 0xff;
    }
    // 设置虚拟MAC地址
    for (int i = 0; i < 6; i++)
    {
        ARPFrame.FrameHeader.SrcMAC[i] = 0x66;
    }
    // 帧类型为ARP
    ARPFrame.FrameHeader.FrameType = htons(0x0806);
    // 硬件类型为以太网
    ARPFrame.HardwareType = htons(0x0001);
    // 协议类型为IP
    ARPFrame.ProtocolType = htons(0x0800);
    // 硬件地址长度为6
    ARPFrame.HLen = 6;
    // 协议地址长为4
    ARPFrame.PLen = 4;
    // 操作为ARP请求
    ARPFrame.Operation = htons(0x0001);
    // 设置虚拟MAC地址
```

```
    for (int i = 0; i < 6; i++)
    {
        ARPFrame.SendHa[i] = 0x66;
    }
    // 设置虚拟IP地址
    ARPFrame.SendIP = inet_addr("112.112.112.112");
    // 设置未知的MAC地址
    for (int i = 0; i < 6; i++)
    {
        ARPFrame.RecvHa[i] = 0x00;
    }
    ARPFrame.RecvIP = ip;

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


    if (ahandle == nullptr)
    {
        cout << "网卡接口打开错误" << endl;
    }
    else
    {
        if (pcap_sendpacket(ahandle, (u_char*)&ARPFrame,
sizeof(ARPFrame_t)) == 0)
        {
            ARPFrame_t* IPPacket;
            while (true)
            {
                pcap_pkthdr* pkt_header;
                const u_char* pkt_data;
                int rtn = pcap_next_ex(ahandle, &pkt_header, &pkt_data);
                if (rtn == 1)
                {
                    // cout << 1 << endl;
                    IPPacket = (ARPFrame_t*)pkt_data;
                    if (ntohs(IPPacket->FrameHeader.FrameType) == 0x0806)
                    {
                        // cout << 2 << endl;
                        // 输出目的MAC地址
                        if (!compare(IPPacket->FrameHeader.SrcMAC,
ARPFrame.FrameHeader.SrcMAC) && compare(IPPacket->FrameHeader.DesMAC,
ARPFrame.FrameHeader.SrcMAC))
                            // if (ntohs(IPPacket->Operation) == 0x0002)
                        {
                            // 把获得的关系写入到日志表中
                            ltable.write2log_arp(IPPacket);
                            // cout << 3 << endl;
                            // 输出源MAC地址，源MAC地址即为所需MAC地址
                            for (int i = 0; i < 6; i++)
                            {
                                selfmac[i] = IPPacket-
>FrameHeader.SrcMAC[i];
                            }
```

```
                            break;
                        }
                    }
                }
            }
        }
    }
```

得到本机网络接口的MAC地址和其上绑定的IP地址后，应用程序就可以组装和发送ARP请求报文，请求以太网中其他主机的IP地址和MAC地址的对应关系，发送步骤与之前几乎是相同的，使用如下代码即可获取目的IP的MAC地址：

```
void getothermac(DWORD ip_, BYTE 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
    for (int i = 0; i < 6; i++)
    {
        ARPFrame.FrameHeader.SrcMAC[i] = selfmac[i];
        ARPFrame.SendHa[i] = selfmac[i];
    }

    // 帧类型为ARP
    ARPFrame.FrameHeader.FrameType = htons(0x0806);
    // 硬件类型为以太网
    ARPFrame.HardwareType = htons(0x0001);
    // 协议类型为IP
    ARPFrame.ProtocolType = htons(0x0800);
    // 硬件地址长度为6
    ARPFrame.HLen = 6;
    // 协议地址长为4
    ARPFrame.PLen = 4;
    // 操作为ARP请求
    ARPFrame.Operation = htons(0x0001);

    // 将ARPFrame.SendIP设置为本机网卡上绑定的IP地址
    ARPFrame.SendIP = inet_addr(ip[0]);

    // 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)
    {
        cout << "网卡接口打开失败" << endl;
    }
    else
    {
        if (pcap_sendpacket(ahandle, (u_char*)&ARPFrame,
sizeof(ARPFrame_t)) != 0)
        {
            cout << "发送失败" << endl;
        }
        else
        {
            while (true)
            {
                cout << "外部发送成功" << endl;
                pcap_pkthdr* pkt_header;
                const u_char* pkt_data;
                int rtn = pcap_next_ex(ahandle, &pkt_header, &pkt_data);
                if (rtn == 1)
                {
                    ARPFrame_t* IPPacket = (ARPFrame_t*)pkt_data;
                    if (ntohs(IPPacket->FrameHeader.FrameType) == 0x806)
                    {
                        // 输出目的MAC地址
                        if (!compare(IPPacket->FrameHeader.SrcMAC,
ARPFrame.FrameHeader.SrcMAC) && compare(IPPacket->FrameHeader.DesMAC,
ARPFrame.FrameHeader.SrcMAC) && IPPacket->SendIP == ip_)//&&ip==IPPacket-
>SendIP
                        {
                            // 把获得的关系写入到日志表中
                            ltable.write2log_arp(IPPacket);
                            // 输出源MAC地址
                            for (int i = 0; i < 6; i++)
                            {
                                mac[i] = IPPacket->FrameHeader.SrcMAC[i];
                            }
                            break;
                        }
                    }
                }
            }
        }
    }
}
```

接下来是路由表，初始化时先将两个默认ip存储进去，插入的函数如果不是直接投递的表项按照掩码长短插入，方便最长匹配，然后重新付给索引。打印路由表就是将掩码，目的网络，下一跳，类型都打印，按照链表来遍历，删除的话只能删除不是直接投递的表项，否则会报错。路由表的最后是转发时查找下一跳ip，如果可以找到直接返回即可。

```
// 路由表采用链表形式 并初始化直接跳转的网络
routetable::routetable()
{
    head = new routeitem;
    tail = new routeitem;
    head->nextitem = tail;
    num = 0;

    // 本次实验初始一定只有两个网络
    for (int i = 0; i < 2; i++)
    {
        routeitem* temp = new routeitem;

        // 本机网卡的ip和掩码进行按位与 所得为网络号
        temp->net = (inet_addr(ip[i])) & (inet_addr(mask[i]));
        temp->mask = inet_addr(mask[i]);
        temp->type = 0;

        // 将其初始化到链表中
        this->add(temp);
    }
}

// 插入表项
void routetable::add(routeitem* a)
{
    // 直接投递的表项
    if (!a->type)
    {
        a->nextitem = head->nextitem;
        head->nextitem = a;
        a->type = 0;
    }

    // 方便找到插入的位置
    routeitem* pointer;

    // 不是直接投递的表相：按照掩码由长至短找到合适的位置
    if (a->type)
    {
        // for (pointer = head->nextitem; pointer != tail && pointer-
>nextitem != tail; pointer = pointer->nextitem)
        for (pointer = head->nextitem; pointer != tail; pointer = pointer-
>nextitem)
        {
            if (a->mask < pointer->mask && a->mask >= pointer->nextitem-
```

```
>mask || pointer->nextitem == tail)
            {
                break;
            }
        }
        a->nextitem = pointer->nextitem;

        // 插入到合适位置
        pointer->nextitem = a;
    }

    routeitem* p = head->nextitem;
    for (int i = 0; p != tail; p = p->nextitem, i++)
    {
        p->index = i;
    }
    num++;
}

// 打印表项
void routeitem::printitem()
{
    // 打印的内容为：`掩码 目的网络 下一跳IP 类型`
    in_addr addr;

    // 多打印一个索引
    printf("%d   ", index);

    addr.s_addr = mask;
    char* pchar = inet_ntoa(addr);
    printf("%s\t", pchar);

    addr.s_addr = net;
    pchar = inet_ntoa(addr);
    printf("%s\t", pchar);

    addr.s_addr = nextip;
    pchar = inet_ntoa(addr);
    printf("%s\t\t", pchar);

    printf("%d\n", type);
}

// 打印路由表
void routetable::print()
{
    routeitem* p = head->nextitem;
    for (; p != tail; p = p->nextitem)
    {
        p->printitem();
    }
}

// 删除表项
```

```
void routetable::remove(int index)
{
    for (routeitem* t = head; t->nextitem != tail; t = t->nextitem)
    {
        if (t->nextitem->index == index)
        {
            // 直接投递的路由表项不可删除
            if (t->nextitem->type == 0)
            {
                cout << "该项无法删除" << endl;
                return;
            }
            else
            {
                t->nextitem = t->nextitem->nextitem;
                return;
            }
        }
    }
    cout << "查无此项! " << endl;
}

// 查找表项 并返回下一跳`ip`地址
DWORD routetable::lookup(DWORD ip)
{
    routeitem* t = head->nextitem;
    for (; t != tail; t = t->nextitem)
    {
        //调试相应的地址所用，当时一直没出来
        //cout << "-------------------------------------------------------
-----------" << endl;
        //cout << "mask: " << t->mask << endl;
        //cout << "ip: " << ip << endl;
        //cout << "mask&ip: " << (t->mask & ip) << endl;
        //cout << "net: " << t->net << endl;
        //cout << "nextip: " << t->nextip << endl;
        //cout << "-------------------------------------------------------
-----------" << endl;
        if ((t->mask & ip) == t->net)
        {
            return t->nextip;
        }
    }
    cout << "未找到对应跳转地址，退出程序" << endl;
    return -1;
}
```

转发函数根据两个mac地址修改即可，具体代码如下：

```
// 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;

        // 源MAC为本机MAC
        memcpy(temp->FrameHeader.SrcMAC, temp->FrameHeader.DesMAC, 6);

        // 目的MAC为下一跳MAC
        memcpy(temp->FrameHeader.DesMAC, dmac, 6);

        // 发送数据报
        int rtn = pcap_sendpacket(ahandle, (const u_char*)temp, 74);
        if (rtn == 0)
        {
            // 将其写入日志
            ltable.write2log_ip("转发", temp);
        }

    }
```

比较重要的是接收处理线程，首先是一个内部循环，每次接收到消息再开始下一步，判断目的mac为自己并且是ip数据报，用其目的地址查找下一跳，如果没有直接丢弃，如果有则将其打印到程序中。接下来检查校验和，并且不能是发向自己的和广播消息，随后如果是直接投递，查看arp表是否有mac地址的存储，没有则需要获得，不是直接投递的也是如此，具体代码如下所示：

```
DWORD WINAPI handlerRequest(LPVOID lparam)
{
    routetable zyl_table = *(routetable*)(LPVOID)lparam;
    while (true)
    {
        pcap_pkthdr* pkt_header;
        const u_char* pkt_data;
        while (true)
        {
            int rtn = pcap_next_ex(ahandle, &pkt_header, &pkt_data);
            // 接收到消息就跳出循环
            if (rtn)
            {
                break;
            }
        }
        FrameHeader_t* header = (FrameHeader_t*)pkt_data;
        // 目的`MAC`地址是本机
        if (compare(header->DesMAC, selfmac))
        {
            // IP格式的数据报
            if (ntohs(header->FrameType) == 0x0800)
            {
                Data_t* data = (Data_t*)pkt_data;
                // 写入日志
                ltable.print();
                ltable.write2log_ip("接收", data);
```

```
                DWORD ipzyl_ = data->IPHeader.SrcIP;
                DWORD ip1_ = data->IPHeader.DstIP;
                // 查找是否有对应表项
                DWORD ip_ = zyl_table.lookup(ip1_);

                // 如果没有则丢弃或递交至上层
                if (ip_ == -1)
                {
                    continue;
                }

                // 打印的内容为：`源IP 目的IP 下一跳IP`
                in_addr addr;

                cout << "-----------------------------------------------
----------------------------------------" << endl;

                cout << "转发数据报ing" << endl;

                cout << "源IP:  ";
                addr.s_addr = ipzyl_;
                char* pchar = inet_ntoa(addr);
                printf("%s\t", pchar);
                cout << endl;

                cout << "目的IP:  ";
                addr.s_addr = ip1_;
                pchar = inet_ntoa(addr);
                printf("%s\t", pchar);
                cout << endl;

                cout << "下一跳IP:  ";
                addr.s_addr = ip_;
                pchar = inet_ntoa(addr);
                printf("%s\t\t", pchar);
                cout << endl;

                cout << "-----------------------------------------------
----------------------------------------" << endl;

                // 如果校验和不正确，则直接丢弃不进行处理
                if (checkchecksum(data))
                {
                    // 不是自己网卡的消息
                    if (data->IPHeader.DstIP != inet_addr(ip[0]) && data-
>IPHeader.DstIP != inet_addr(ip[1]))
                    {
                        int t1 = compare(data->FrameHeader.DesMAC,
broadcast);
                        int t2 = compare(data->FrameHeader.SrcMAC,
broadcast);
                        // 不是广播消息
                        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)
                            {
                                // 则需要获取ARP
                                if (!arptable::lookup(ip1_, mac))
                                {
                                    arptable::insert(ip1_, mac);
                                }
                                // 转发数据报
                                resend(temp, mac);
                            }
                            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);
                                }
                                // 转发数据报
                                resend(temp, mac);
                            }
                        }
                    }
                }
            }
        }
    }
}
```

- arp表的插入是先获取目的虚拟的mac地址然后插入，查找则是按ip进行查找，具体代码如下所示：

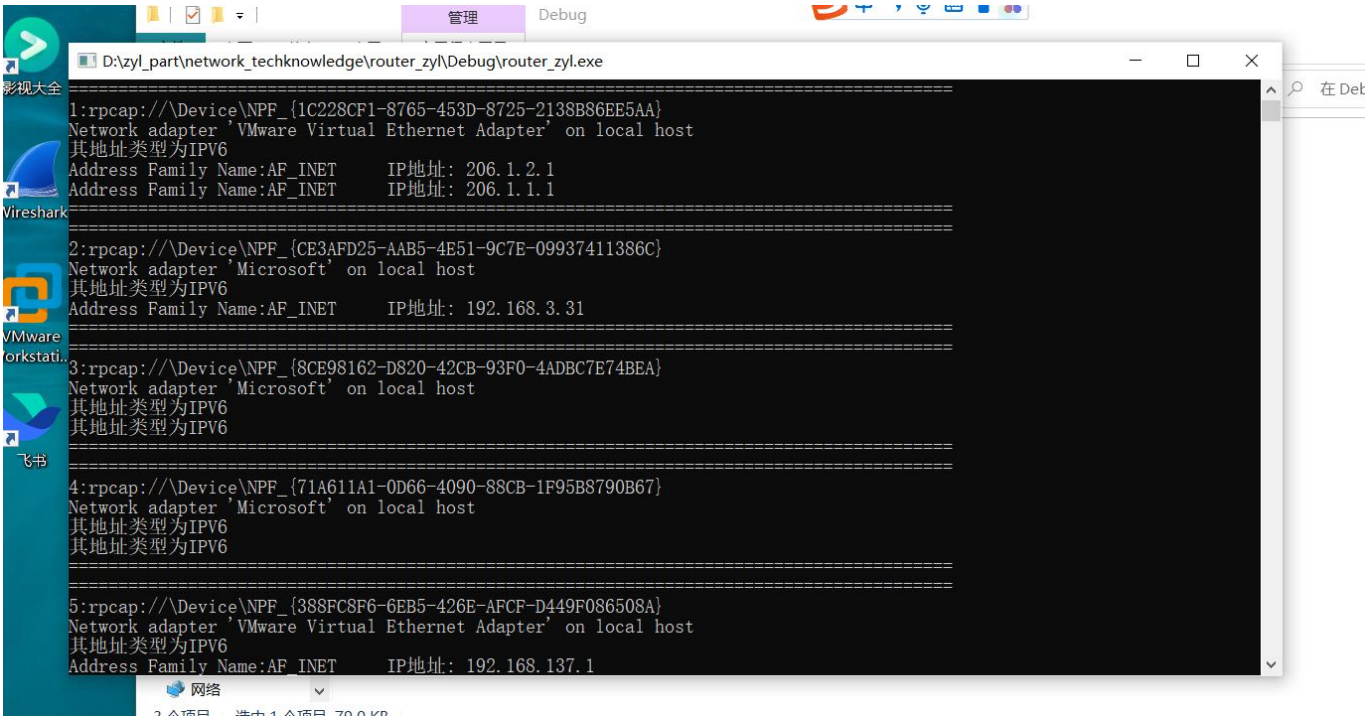
```
void arptable::insert(DWORD ip, BYTE mac[6])
{
    atable[num].ip = ip;
    getothermac(ip, atable[num].mac);
    memcpy(mac, atable[num].mac, 6);
    num++;
}
int arptable::lookup(DWORD ip, BYTE mac[6])
{
    memset(mac, 0, 6);
    for (int i = 0; i < num; i++)
    {
        if (ip == atable[i].ip)
        {
            memcpy(mac, atable[i].mac, 6);
            return 1;
```

```
            }
        }
        // 未知返回0
        return 0;
    }
```

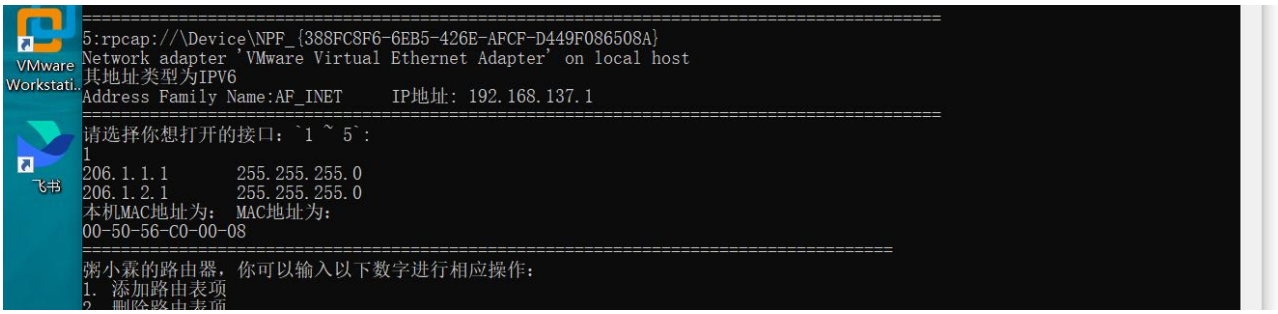
本次复杂又耗时（花费整整3周）的大项目终于到这里也接近尾声了！

(3)结果展示

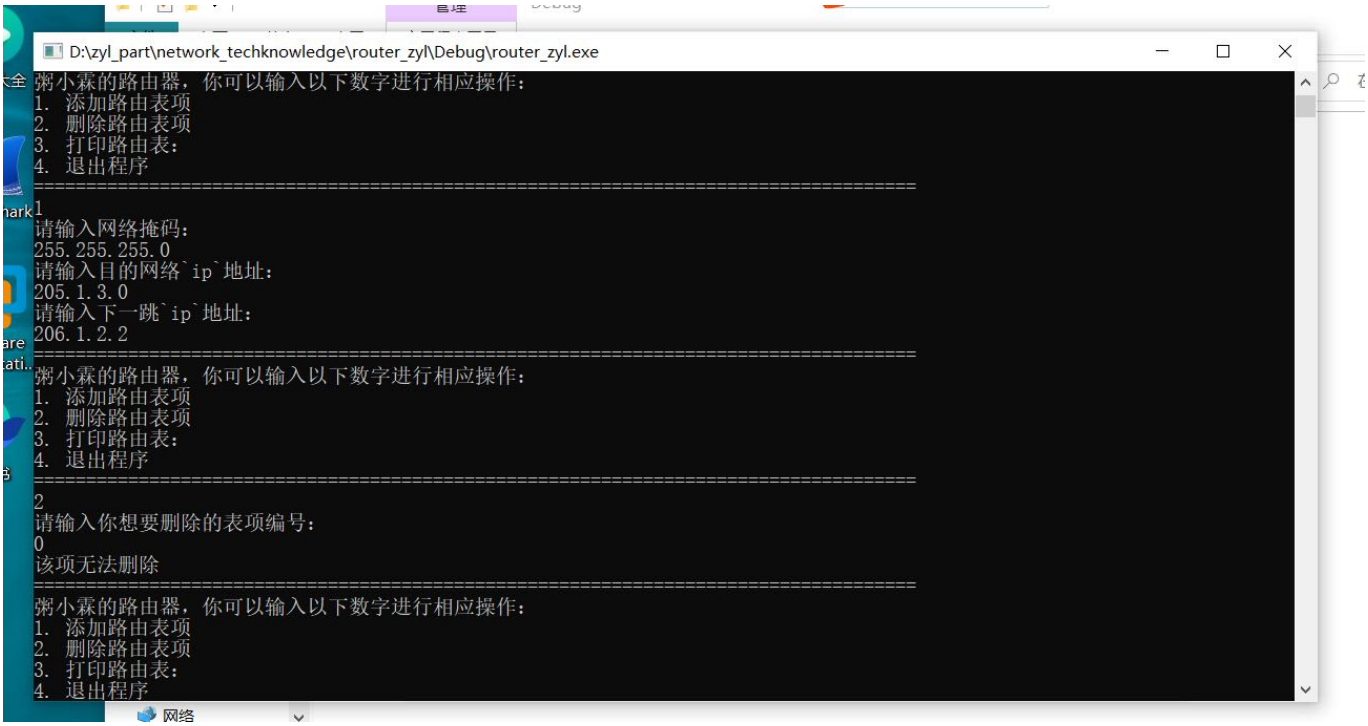
首先是打开接口列表：

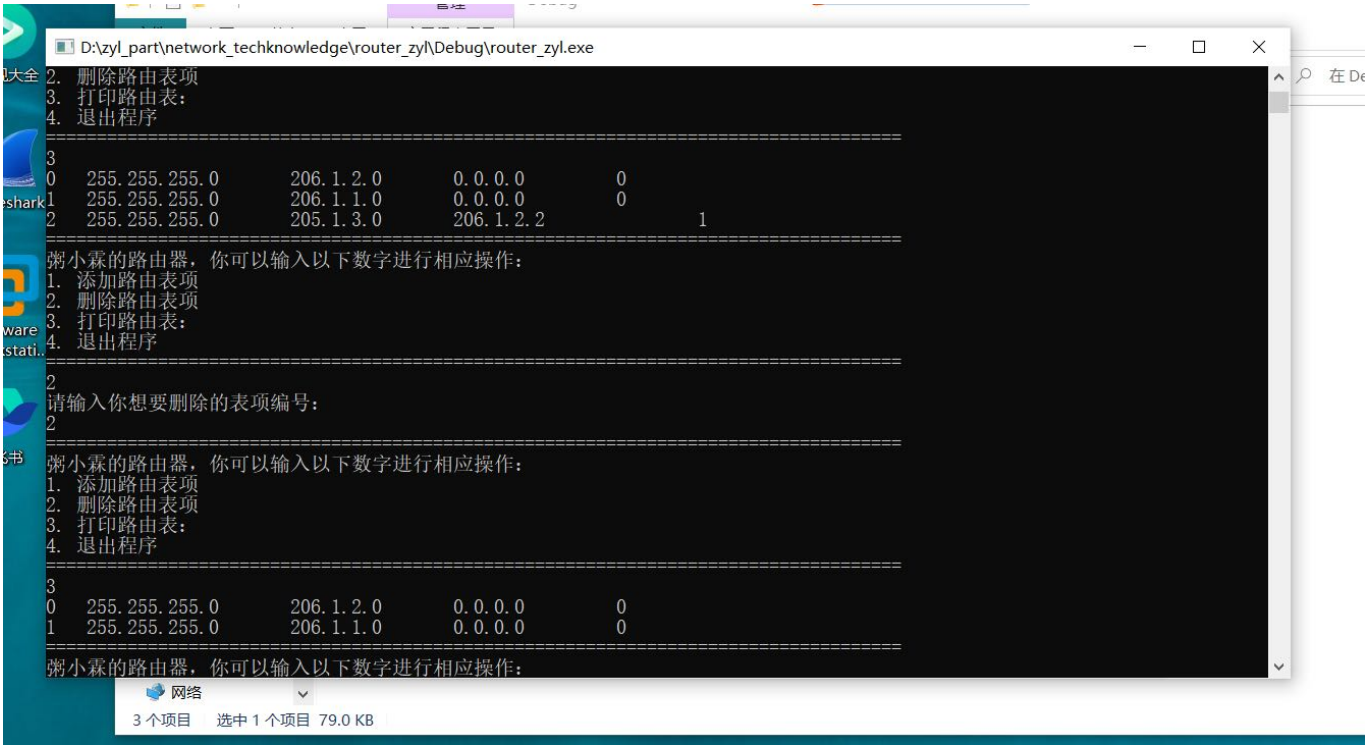


然后是打开双IP网卡，并带有mac地址的输出：

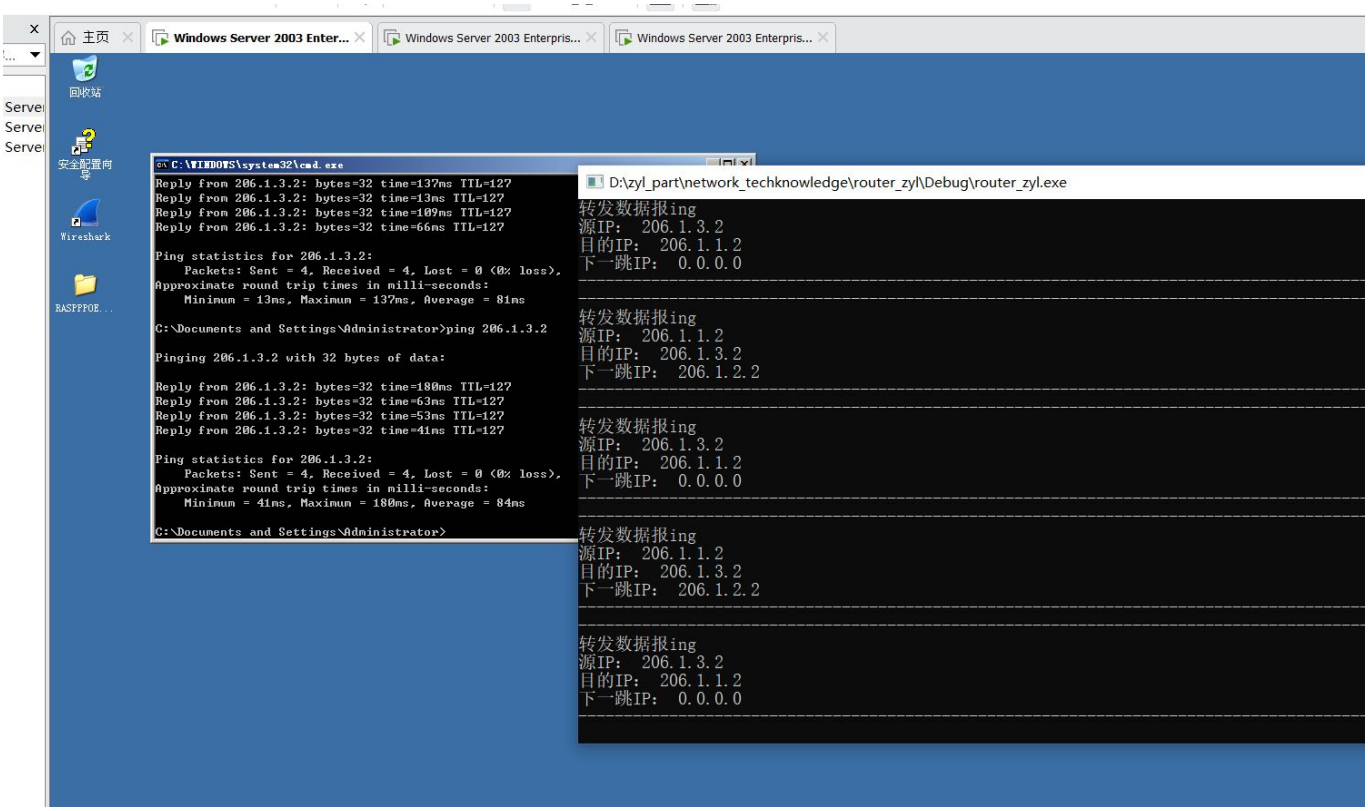


接下来是添加、删除、打印路由表项：


D:\zyl_part\network_techknowledge\router_zyl\Debug\router_zyl.exe
粥小霖的路由器，你可以输入以下数字进行相应操作：
1. 添加路由表项
2. 删除路由表项
3. 打印路由表：
4. 退出程序
1
请输入网络掩码：
255.255.255.0
请输入目的网络`ip`地址：
205.1.3.0
请输入下一跳`ip`地址：
206.1.2.2
粥小霖的路由器，你可以输入以下数字进行相应操作：
2
请输入你想要删除的表项编号：
0
该项无法删除
粥小霖的路由器，你可以输入以下数字进行相应操作：
网络


D:\zyl_part\network_techknowledge\router_zyl\Debug\router_zyl.exe
2. 删除路由表项
3. 打印路由表：
4. 退出程序
3
0 255.255.255.0 206.1.2.0 0.0.0.0 0
1 255.255.255.0 206.1.1.0 0.0.0.0 0
2 255.255.255.0 205.1.3.0 206.1.2.2 1
粥小霖的路由器，你可以输入以下数字进行相应操作：
2
请输入你想要删除的表项编号：
2
粥小霖的路由器，你可以输入以下数字进行相应操作：
3
0 255.255.255.0 206.1.2.0 0.0.0.0 0
1 255.255.255.0 206.1.1.0 0.0.0.0 0
粥小霖的路由器，你可以输入以下数字进行相应操作：
网络
3 个项目 选中 1 个项目 79.0 KB

最后是转发功能：



> 可以看到实现了转发功能

四、特殊现象分析

本次实验中，分别在程序和设备各遇到一处卡点，下面将分别进行介绍：

(1)设备

在这次实验中，原本是要用四台老师给定的虚拟机来模仿之前在机房实现的路由转发功能，但是在实验进行的时候发现虚拟机2（也就是原本要运行我们自己设计的路由器的虚拟机），由于缺少一些东西导致exe程序并不能在这台虚拟机上运行，然后按照老师的提示使用本机来运行程序，使主机成为路由器。

在按照之前的经验打开服务，配置完网卡以及关闭防火墙后以为就可以了，但是怎么都ping不通，甚至都不能用虚拟机1来ping通主机，在这里卡了好久，最终查阅相关资料发现需要将虚拟机的网络使用到自定义的虚拟网卡上，实现了此步骤后，发现终于能ping通了。

(2)程序

在程序中，也是边调试边实现，最后写完了将近1200行的代码，也算是继数据库大作业后的又一个大程序，其中本来是一切顺利的，但是到最后运行的时候确怎么也转发不了，在程序中加入了一些cout后发现是转发函数的问题，其一直不能找到对应的网络，但是我也确实将此路由表项添加到路由表中，并且路由表打印出来也是有的，这里也卡了很久，最后输出转发地址才发现问题，这个地址明显不对，由回头看了一眼头部的定义才发现头部少写了一个按一字节进行对齐，所以将其补上之后，整个程序就跑通了，大作业也到此结束。

五、总结与展望

(1)总结

本次实验是网络技术与应用的第五次实验，也是本学期的最后一次实验，通过本次实验对路由器编写，尽管此次编写的程序非常的复杂和耗时，但是对路由程序的编制使我深入了解了互联层的工作原理，对网络方面的知识有了更深层的认识，在网络方面的认知也更上一层楼。

(2)展望

本门课程是与计算机网络课相辅相成的一门课，通过上这门课使得对计算机网络课有些不理解的地方有了更多的感悟，对网络也有了更多的兴趣，期望未来更好的发展，心想事成、万事胜意、未来可期！