网络技术与应用课程报告

第三次实验报告

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

通过编程获取IP地址与MAC地址的对应关系

要求如下：

1. 在IP数据报捕获与分析编程实验的基础上，学习WinPcap的数据包发送方法
2. 通过Npcap编程，获取IP地址与MAC地址的映射关系
3. 程序要具有输入IP地址，显示输入IP地址与获取的MAC地址对应关系界面。界面可以是命令行界面，也可以是图形界面，但应以简单明了的方式在屏幕上显示
4. 编写的程序应结构清晰，具有较好的可读性

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：

<table>
<tr><td colspan="2">硬件类型</td><td>协议类型</td></tr>
<tr><td>硬件长度</td><td>协议长度</td><td>操作码（请求为1，响应为2）</td></tr>
<tr><td colspan="3">源硬件地址</td></tr>
<tr><td colspan="3">源逻辑地址</td></tr>
<tr><td colspan="3">目的硬件地址</td></tr>
<tr><td colspan="3">目的逻辑地址</td></tr>
</table>

（图3）ARP报文格式

- 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。

三、实验过程

(1)项目设计思路

个人认为本次实验还是非常友善的，首先在实验指导书上，已经把大体思路都设计出来了，需要我们完成的是把这些部分组合起来，并实现我们想要实现的功能，即获得本地的IP地址与MAC地址的关系以及目的IP与MAC地址之间的关系。

> 注意：由于广播发送只能在相同网段的网卡内进行发送，而主机上各个网卡的IP可能并不在同一网段，例如子网掩码为255.255.255.0的两个IP192.168.174.1和192.168.120.1，不能接收到对方的广播消息。所以想要成功获取ARP应答必须 ***使用相同的网卡发出和响应ARP报文***

所以按照课本上的要求，依次完成：

1. 获取本机网络接口的MAC地址和IP地址
2. 向网络发送数据报

需要注意的是我们需要按照书上的介绍，将捕获的的结构体强制转化成我们所需要的格式——即标准数据报所具有的格式，因为其是按字节划分的，所以需要用到pack()函数，打包过程如以下代码：

```
// 字节对齐方式
#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;

// IP首部
typedef struct ARPFrame_t {
    // 用上面定义好的帧首部
    FrameHeader_t FrameHeader;
    // 硬件类型
    WORD HardwareType;
    // 协议类型（本次实验应该为ARP）
    WORD ProtocolType;
    // 硬件地址长度
    BYTE HLen;
    // 协议地址长度
    BYTE PLen;
    // 操作类型（比如ARP的请求或应答）
    WORD Operation;
    // 发送方MAC地址，即源MAC地址
    BYTE SendHa[6];
    // 发送方IP地址，即源IP地址
    DWORD SendIP;
    // 接收方MAC地址，即目的MAC地址
    BYTE RecvHa[6];
    // 接收方IP地址，即目的IP地址
    DWORD RecvIP;
}ARPFrame_t;

// 结束字节对齐方式
# pragma pack()
```

捕获完数据包后就可以根据以上结构体，输出我们想要的数据。

本次实验需要获取主机网卡中对应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

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，一开始需要引入头函数<pcap.h>,然后是获取设备列表，先定义接口指针以及将来会用到选设备时候的int型变量和一个错误信息缓冲区，并且还需要为即将用到的IP提前声明一个字符串，由于在编译软件中不能利用字符常量，所以自己写了一个error1并赋值给获取接口列表的第一个参数，然后就可以利用pcap_findalldevs_ex函数来获取计算机上的网络接口设备的列表，如果返回值为-1——即出现异常的话，则会显示异常信息并结束进程，具体代码如下：

```
    // 指向所有设备链表首部
    pcap_if_t* alldevs;
    // 之后回用到循环设备的变量
    pcap_if_t* d;
    // 设置错误信息缓冲区的大小
    char errbuf[PCAP_ERRBUF_SIZE];
    // 所有的接口个数
    int num = 0;
    // 用户会选择的设备序号
    int n;
    // 预存本机的IP
    char* ip = new char[20];
    // 由于字符串不能为常量，所以自己输入
    char error1[9];
    error1[0] = 'r';
    error1[1] = 'p';
    error1[2] = 'c';
    error1[3] = 'a';
    error1[4] = 'p';
    error1[5] = ':';
    error1[6] = '/';
    error1[7] = '/';
    error1[8] = '\0';

    // 获得本机的设备列表，如果错误直接返回
    if (pcap_findalldevs_ex(error1, NULL, &alldevs, errbuf) == -1)
    {
        cout << "无法获取本机设备" << endl;
        // 释放设备列表
        pcap_freealldevs(alldevs);
        return -1;
    }
```

然后是由用户选择想要嗅探的设备，首先将刚刚捕获到的接口设备信息打印在进程中，这一次需要打印相应的IP地址，便于用户选择，然后由用户输入想要嗅探的接口并对其选择的数字做合法性检测，并跳转到此设备出进行数据报的嗅探，如果嗅探成功则开始进行监听在，准备获取对应关系，如果失败则会显示错误信息并结束进程，具体代码如下所示：

```
    // 显示接口列表描述和对应ip地址
    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));
                }
                break;
            // IPV6类型地址
            case AF_INET6:
                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;
        return -1;
    }

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

```
    // 跳转到相应接口
    for (d = alldevs; num < (n - 1); num++)
    {
        d = d->next;
    }

    // 将设备的IP地址赋值给`ip`
    strcpy(ip, inet_ntoa(((struct sockaddr_in*)(d->addresses)->addr)->sin_addr));

    pcap_t* adhandle;
    adhandle = pcap_open(d->name, 65536, PCAP_OPENFLAG_PROMISCUOUS, 1000, NULL, errbuf);
    if (adhandle == NULL)
    {
        cout << "打开接口失败" << endl;
        pcap_freealldevs(alldevs);
        return -1;
    }
    else
    {
        cout << "开始监听: " << endl;
        pcap_freealldevs(alldevs);
    }
```

接下来发送ARP请求，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：

```
    // 一、接下来的程序是设置ARP帧的内容，并获取本机的MAC地址与IP地址的关系

    // 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;
    }
    // 请求方为本机的IP地址
    ARPFrame.RecvIP = inet_addr(ip);

    // 发送设置好的帧内容，如果发送失败直接退出
    if (pcap_sendpacket(adhandle, (u_char*)&ARPFrame, sizeof(ARPFrame_t))
!= 0)
    {
        cout << "发送失败" << endl;
        return -1;
    }

    // 声明即将捕获的ARP帧
    ARPFrame_t* IPPacket;

    // 捕获消息，可能会收到多个数据报
    while(true)
    {
        pcap_pkthdr* pkt_header;
        const u_char* pkt_data;
        int rtn = pcap_next_ex(adhandle, &pkt_header, &pkt_data);
        // 捕获到相应的信息
        if (rtn == 1)
        {
            IPPacket = (ARPFrame_t*)pkt_data;
            if ((ntohs(IPPacket->FrameHeader.FrameType) == 0x0806) &&
(ntohs(IPPacket->Operation) == 0x0002))//如果帧类型为ARP并且操作为ARP应答
            {
                // 打印本机的MAC地址和IP地址
                printf("%s\t%s\n", "IP地址:", ip));
                printf("Mac地址：\n");
                printf("%02x:%02x:%02x:%02x:%02x:%02x\n",
                    IPPacket->FrameHeader.SrcMAC[0],
                    IPPacket->FrameHeader.SrcMAC[1],
```

```
                    IPPacket->FrameHeader.SrcMAC[2],
                    IPPacket->FrameHeader.SrcMAC[3],
                    IPPacket->FrameHeader.SrcMAC[4],
                    IPPacket->FrameHeader.SrcMAC[5]
                );
                // 调试程序所用
                cout << 1 << endl;
                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，可以直接使用伪造的IP和MAC地址进行发送，虽然本机网卡发送时需用的是虚拟MAC和IP地址，但是网关接收到组建的ARP请求后会由网关发出一个ARP请求，找到本机发送网卡的真实IP和MAC地址，从而进一步获取远程主机的MAC。

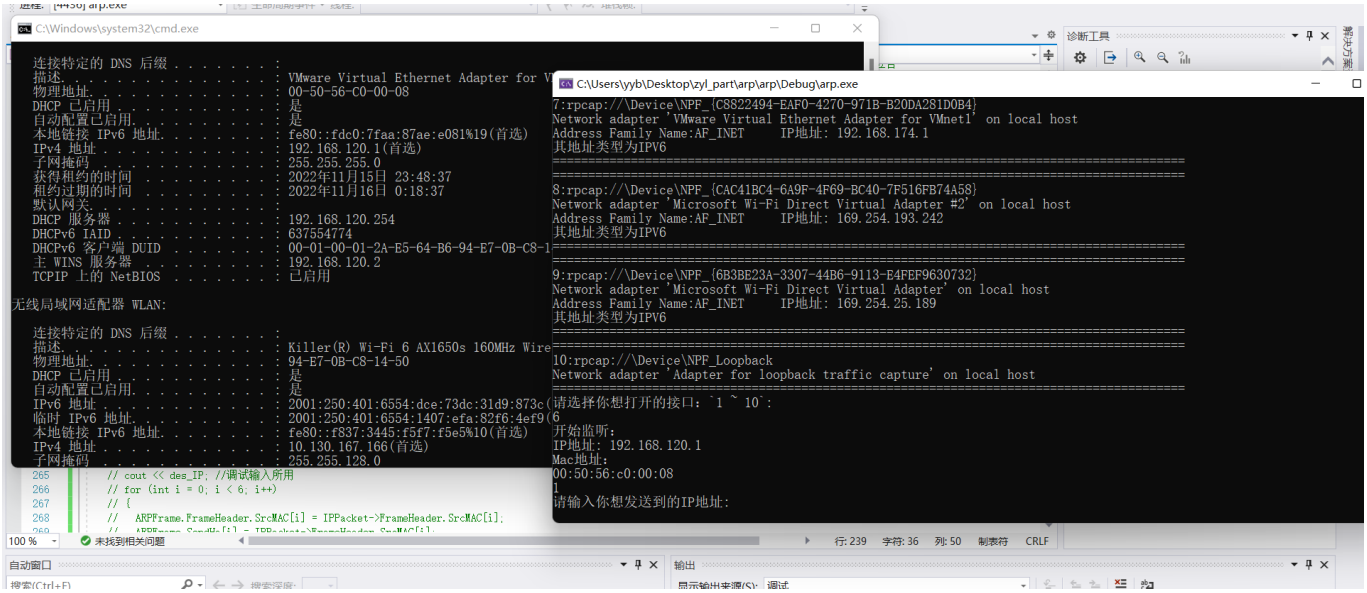
```
    // 二、接下来更改ARP帧的内容，获取想获得的目的IP地址和MAC地址的关系，但只能在同一个网段中

    // 目的ip
    char* des_IP = new char;

    cout << "请输入你想发送到的IP地址:" << endl;
    cin >> des_IP;

    // 发现网关回自动切换到这个网段，所以并不需要去改变原本的MAC地址以及IP地址
    // cout << des_IP; //调试输入所用
    // for (int i = 0; i < 6; i++)
    // {
    //   ARPFrame.FrameHeader.SrcMAC[i] = IPPacket->FrameHeader.SrcMAC[i];
    //   ARPFrame.SendHa[i] = IPPacket->FrameHeader.SrcMAC[i];
    // }
    // ARPFrame.SendIP = inet_addr(des_IP);

    // 接收方还是本机
    ARPFrame.RecvIP = inet_addr(des_IP);
    // 发送设置好的帧内容，如果发送失败直接退出
    if (pcap_sendpacket(adhandle, (u_char*)&ARPFrame, sizeof(ARPFrame_t)) != 0)
    {
        cout << "发送失败" << endl;
        return -1;
    }
    else
    {
        cout << "发送成功" << endl;
    }
    ARPFrame_t* IPPacketNew;

    // 捕获消息，可能会收到多个数据报，与前一次的捕获方法相同
```

```
        while(true)
        {
            pcap_pkthdr* pkt_headerNew;
            const u_char* pkt_dataNew;
            int rtnNew = pcap_next_ex(adhandle, &pkt_headerNew, &pkt_dataNew);
            if (rtnNew == 1)
            {
                IPPacketNew = (ARPFrame_t*)pkt_dataNew;
                if ((ntohs(IPPacketNew->FrameHeader.FrameType) == 0x0806) &&
    (ntohs(IPPacketNew->Operation) == 0x0002))//如果帧类型为ARP并且操作为ARP应答
                {
                    // 输出其对应的MAC地址
                    printf("Mac地址: \n");
                    printf("%02x:%02x:%02x:%02x:%02x:%02x\n",
                        IPPacketNew->FrameHeader.SrcMAC[0],
                        IPPacketNew->FrameHeader.SrcMAC[1],
                        IPPacketNew->FrameHeader.SrcMAC[2],
                        IPPacketNew->FrameHeader.SrcMAC[3],
                        IPPacketNew->FrameHeader.SrcMAC[4],
                        IPPacketNew->FrameHeader.SrcMAC[5]
                    );
                    // 调试程序所用
                    cout << 2 << endl;
                    break;
                }
            }
        }
```

最后需要释放设备列表，即可结束进程。

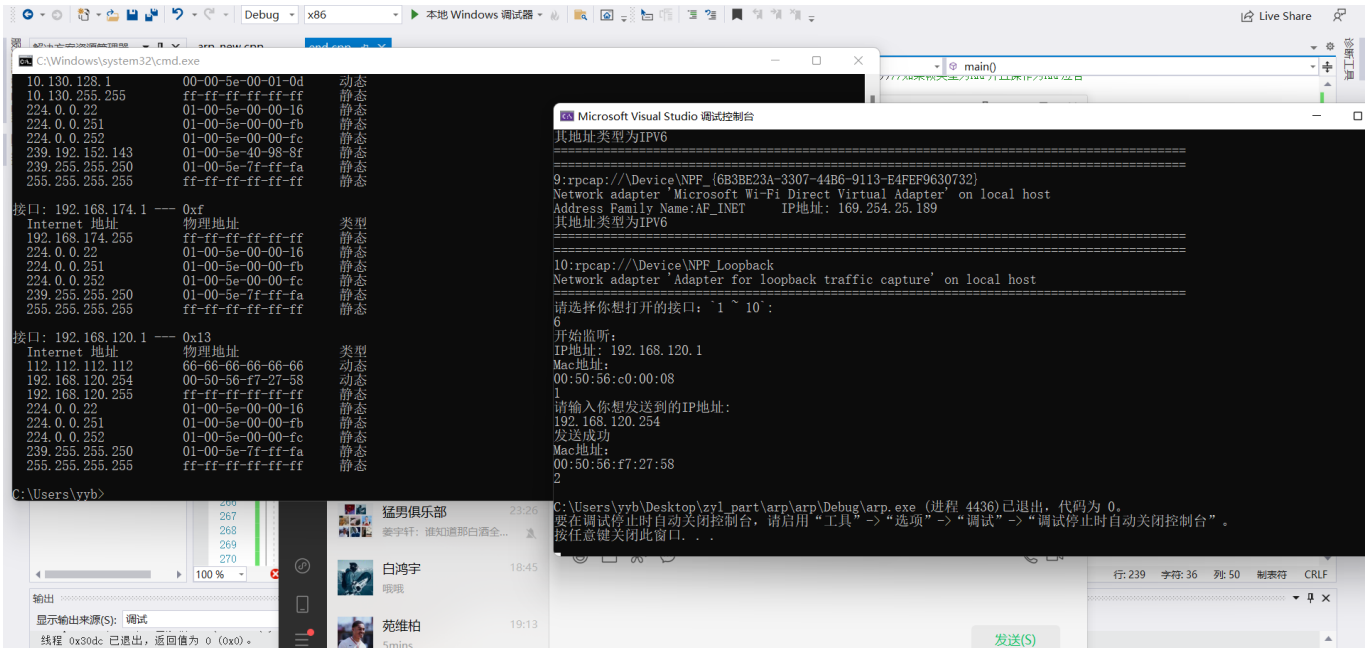
(3)结果展示

首先是获取本机网络接口的MAC地址和IP地址，如下图所示：



- 可以发现结果一致

然后是向网络（DHCP服务器）发送数据报，获取目的IP与MAC地址的关系，如下图所示：



- 可以发现结果一致

四、特殊现象分析

本次实验中，一开始用两台电脑连接同一个WLAN并在一个电脑上给另一台电脑发送数据报获取ARP关系，但是却收不到ARP响应消息，改了半天程序都没有起什么作用。

最后才知道由于广播发送只能在相同网段的网卡内进行发送，而主机上各个网卡的IP可能并不在同一网段，例如子网掩码为255.255.255.0的两个IP192.168.174.1和192.168.120.1，不能接收到对方的广播消息。所以想要成功获取ARP应答必须使用相同的网卡发出和响应ARP报文，最后改到一个网段后则接收成功，本次实验页到此结束。

五、总结与展望

(1)总结

本次实验是网络技术与应用的第三次实验，对数据报的发送与接收方面的知识有了更深层的认识，并了解了ARP报文及相应的构造方法，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，期望自己在这学期未来实验的更好的发展，心想事成、万事胜意。