

网络技术与应用第三次实验报告

网络空间安全学院 物联网工程 2111673 岳志鑫

一、实验要求

通过编程获取 IP 地址与 MAC 地址的对应关系实验，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 Npcap 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、实验内容

1.环境配置

在官网 (<https://npcap.com/#download>) 下载 Npcap 和 Npcap-SDK

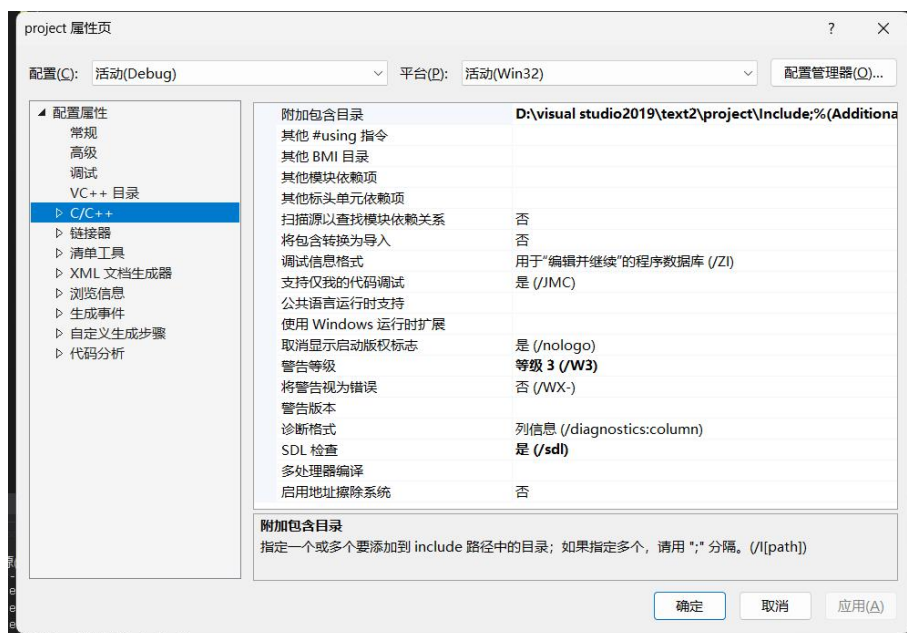
Downloading and Installing Npcap Free Edition

The free version of Npcap may be used (but not externally redistributed) on up to 5 systems ([free license details](#)). It may also be used on unlimited systems where it is only used with [Nmap](#), [Wireshark](#), and/or [Microsoft Defender for Identity](#). Simply run the executable installer. The full source code for each release is available, and developers can build their apps against the SDK. The improvements for each release are documented in the [Npcap Changelog](#).

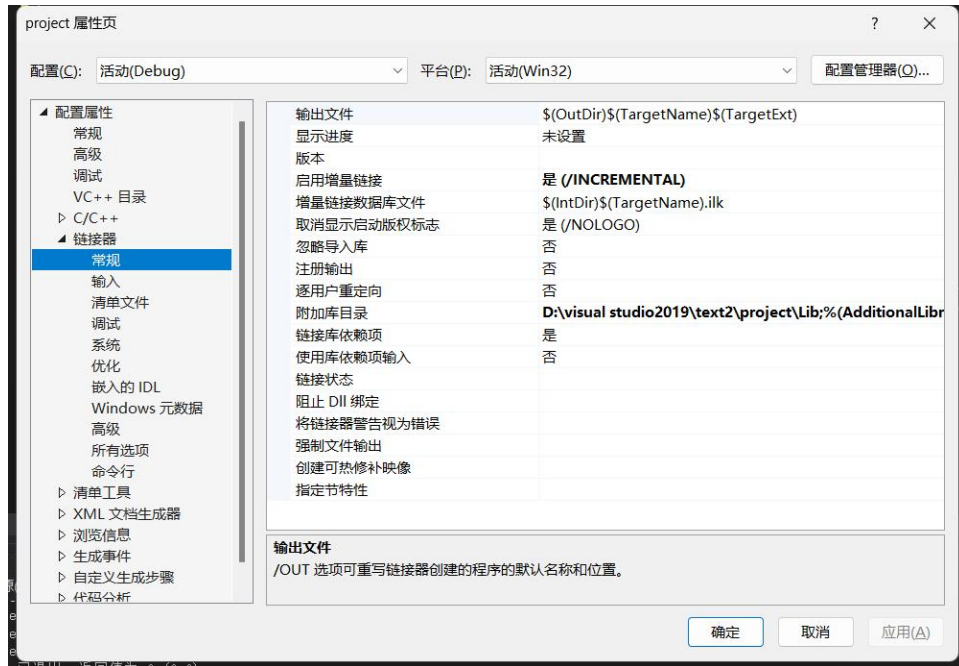
- [Npcap 1.77 installer](#) for Windows 7/2008R2, 8/2012, 8.1/2012R2, 10/2016, 2019, 11 (x86, x64, and ARM64).
- [Npcap SDK 1.13 \(ZIP\)](#).
- [Npcap 1.77 debug symbols \(ZIP\)](#).
- [Npcap 1.77 source code \(ZIP\)](#).

The latest development source is in our [Github source repository](#). Windows XP and earlier are not supported; you can use [WinPcap](#) for these versions.

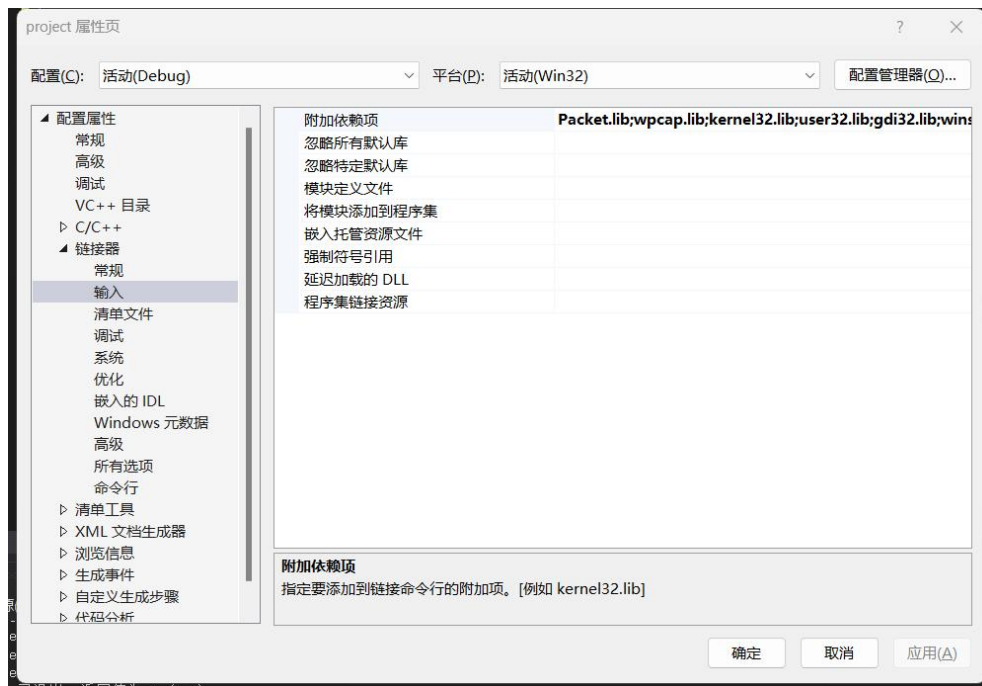
项目→属性→C/C++→常规→附加包含目录：添加 sdk 中的 Include 目录



项目→属性→链接器→常规→附加库目录：添加 sdk 中的 Lib 目录



项目→属性→链接器→输入→附加依赖项：添加 Packet.lib;wpcap.lib;



输入官网的测试代码，若成功检测则环境配置成功

```
#include "pcap.h"

void main()
{
    pcap_if_t *alldevs;
    pcap_if_t *d;
    int i = 0;
```

```

char errbuf[PCAP_ERRBUF_SIZE];

/* Retrieve the device list from the local machine */
if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL /* auth is not needed */, &alldevs, errbuf)
== -1)
{
    fprintf(stderr, "Error in pcap_findalldevs_ex: %s\n", errbuf);
    exit(1);
}

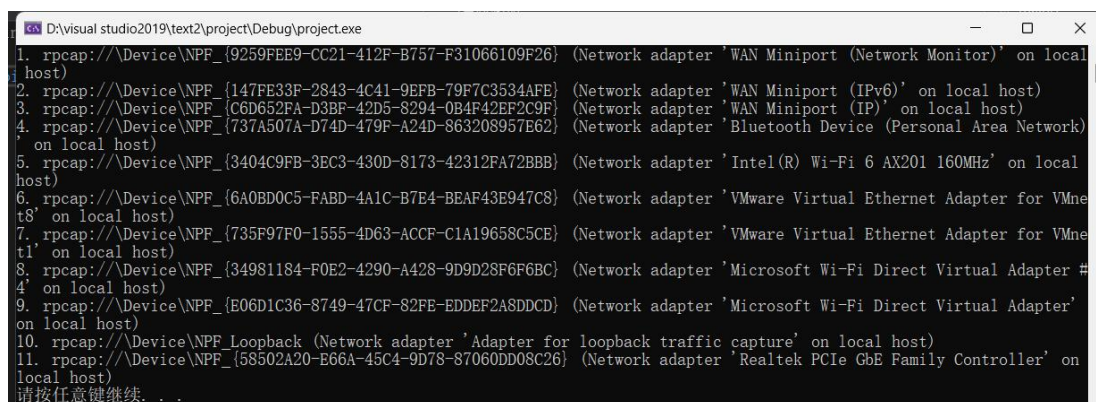
/* Print the list */
for (d = alldevs; d != NULL; d = d->next)
{
    printf("%d. %s", ++i, d->name);
    if (d->description)
        printf(" (%s)\n", d->description);
    else
        printf(" (No description available)\n");
}

if (i == 0)
{
    printf("\nNo interfaces found! Make sure Npcap is installed.\n");
    return;
}

/* We don't need any more the device list. Free it */
pcap_freealldevs(alldevs);

system("pause");
}

```



```

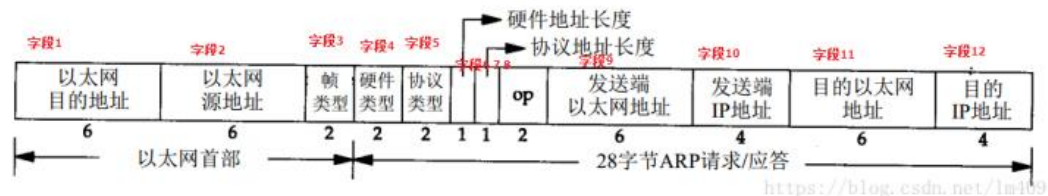
D:\visual studio2019\text2\project(Debug)\project.exe
1. rpcap://\Device\NPF_{9259FEE9-CC21-412F-B757-F31066109F26} (Network adapter 'WAN Miniport (Network Monitor)' on local host)
2. rpcap://\Device\NPF_{147FE33F-2843-4C41-9EFB-79F7C3534AFE} (Network adapter 'WAN Miniport (IPv6)' on local host)
3. rpcap://\Device\NPF_{C6D652FA-D3BF-42D5-S294-0B4F42EF2C9F} (Network adapter 'WAN Miniport (IP)' on local host)
4. rpcap://\Device\NPF_{737A507A-D74D-479F-A24D-863208957E62} (Network adapter 'Bluetooth Device (Personal Area Network)' on local host)
5. rpcap://\Device\NPF_{3404C9FB-3EC3-430D-8173-42312FA72BBB} (Network adapter 'Intel(R) Wi-Fi 6 AX201 160MHz' on local host)
6. rpcap://\Device\NPF_{6A0BD0C5-FABD-4A1C-B7E4-BEAF43E947C8} (Network adapter 'VMware Virtual Ethernet Adapter for VMnet8' on local host)
7. rpcap://\Device\NPF_{735F97F0-1555-4D63-ACCF-C1A19658C5CE} (Network adapter 'VMware Virtual Ethernet Adapter for VMnet1' on local host)
8. rpcap://\Device\NPF_{34981184-F0E2-4290-A428-9D9D28F6F6BC} (Network adapter 'Microsoft Wi-Fi Direct Virtual Adapter #4' on local host)
9. rpcap://\Device\NPF_{E06D1C36-8749-47CF-82FE-EDDEF2A8DDCD} (Network adapter 'Microsoft Wi-Fi Direct Virtual Adapter' on local host)
10. rpcap:///Device\NPF_Loopback (Network adapter 'Adapter for loopback traffic capture' on local host)
11. rpcap://\Device\NPF_{58502A20-E66A-45C4-9D78-87060DD08C26} (Network adapter 'Realtek PCIe GbE Family Controller' on local host)
请按任意键继续. . .

```

2. ARP 数据包

地址解析协议（Address Resolution Protocol），其基本功能为透过目标设备的 IP 地址，查询目标设备的 MAC 地址，以保证通信的顺利进行。它是 IPv4 中网络层必不可少的协议，不过在 IPv6 中已不再适用，并被邻居发现协议（NDP）所替代。

ARP 报文格式:



- (1). 字段 1 是 ARP 请求的目的以太网地址，全 1 时代表广播地址。
- (2). 字段 2 是发送 ARP 请求的以太网地址。
- (3). 字段 3 以太网帧类型表示的是后面的数据类型，ARP 请求和 ARP 应答这个值为 0x0806。
- (4). 字段 4 表示硬件地址的类型，硬件地址不只以太网一种，是以太网类型时此值为 1。
- (5). 字段 5 表示要映射的协议地址的类型，要对 IP 地址进行映射，此值为 0x0800。
- (6). 字段 6 和 7 表示硬件地址长度和协议地址长度，MAC 地址占 6 字节，IP 地址占 4 字节。
- (7). 字段 8 是操作类型字段，值为 1，表示进行 ARP 请求；值为 2，表示进行 ARP 应答；值为 3，表示进行 RARP 请求；值为 4，表示进行 RARP 应答。
- (8). 字段 9 是发送端 ARP 请求或应答的硬件地址，这里是以太网地址，和字段 2 相同。
- (9). 字段 10 是发送 ARP 请求或应答的 IP 地址。
- (10). 字段 11 和 12 是目的端的硬件地址和协议地址

3. 主要函数分析

(1). 输出设备和描述信息

利用 pcap 包的指针和 pcap_findalldevs 函数可以查找当前所有网卡设备，并输出查找到的所有设备信息

```
//打印设备信息
//打印设备列表中设备信息
pcap_if_t* count; //遍历用的指针
char srcip[INET_ADDRSTRLEN]; //本机 ip
//输出设备名和描述信息
for (count = devices; count; count = count->next) //借助 count 指针从第一个设备开始访问到最后一个设备
{
    cout << ++i << ". " << count->name; //输出设备信息和描述
    if (count->description) {
        cout << "描述: (" << count->description << ")" << endl;
    }
}
```

```

    }
    for (a = count->addresses; a != NULL; a = a->next) {
        if (a->addr->sa_family == AF_INET) {
            char str[INET_ADDRSTRLEN];
            inet_ntop(AF_INET, getaddress((struct sockaddr*)a->addr), str, sizeof(str));//
            将 a->addr 强制转换为 struct sockaddr_in 类型的指针，并访问 sin_addr 成员，其中包含了 IPv4 地址。
            cout << "IP 地址: " << str << endl;
            inet_ntop(AF_INET, getaddress((struct sockaddr*)a->netmask), str, sizeof(str));
            //将 a->netmask 强制转换为 struct sockaddr_in 类型的指针，从 a->netmask 这个结构中提取子网掩码。
            cout << "子网掩码: " << str << endl;
            inet_ntop(AF_INET, getaddress((struct sockaddr*)a->broadaddr), str,
            sizeof(str));//将 a->netmask 强制转换为 struct sockaddr_in 类型的指针，从 a->broadaddr 这个结构中提
            取广播地址。
            cout << "广播地址: " << str << endl;
        }
    }
}
//设备数量为 0
if (i == 0) {
    cout << endl << "存在错误！无查找设备！" << endl;
    return 0;
}
}

```

(2). 组装报文，利用 ARP_Frame 结构体定义 ARP 数据报内部的所有信息，例如目标 IP 地址和源 IP 地址等信息

```

//组装报文
unsigned char mac[48];
for (int i = 0; i < 6; i++) {
    send_ARPFrame.FrameHeader.DesMAC[i] = 0xFF; //DesMAC 设置为广播地址
    send_ARPFrame.DesMAC[i] = 0x00; //DesMAC 设置为 0
    //SrcMAC 用不到可以不设置
}

send_ARPFrame.FrameHeader.FrameType = htons(0x0806); //帧类型为 ARP
send_ARPFrame.HardwareType = htons(0x0001); //硬件类型为以太网
send_ARPFrame.ProtocolType = htons(0x0800); //协议类型为 IP
send_ARPFrame.HLen = 6; //硬件地址长度为 6
send_ARPFrame.PLen = 4; //协议地址长度为 4
send_ARPFrame.op = htons(0x0001); //操作为 ARP 请求
send_ARPFrame.DesIP = inet_addr(srcip); //设置为本机 IP 地址

```

其中 ARP 数据结构体和数据帧头部定义如下

```

typedef struct Frame_Header//帧首部
{
    BYTE DesMAC[6]; //目的地址
    BYTE SrcMAC[6]; //源地址

```

```

        WORD FrameType; //帧类型
};

typedef struct ARP_Frame//ARP 数据
{
    Frame_Header FrameHeader;
    WORD HardwareType; //硬件类型
    WORD ProtocolType; //协议类型
    BYTE HLen; //硬件长度
    BYTE PLen; //协议长度
    WORD op; //操作类型
    BYTE SrcMAC[6]; //源 MAC 地址
    DWORD SrcIP; //源 IP 地址
    BYTE DesMAC[6]; //目的 MAC 地址
    DWORD DesIP; //目的 IP 地址
};

```

(3) 发送构造的数据包后需要对反馈的数据包进行抓捕并拆包分析，参考 ARP 数据包的结构按字节进行拆分

```

if (*(unsigned short*)(packetData + 12) == htons(0x0806) //帧类型为 ARP (htons(0x0806))
    && *(unsigned short*)(packetData + 20) == htons(0x0002)) //操作类型为 ARP 响应
(htons(0x0002))
{
    cout << endl;
    cout << "-----" << endl;
    cout << "ARP 数据包内容: " << endl;
    //打印数据包
    cout << "源 IP 地址:\t ";
    // 提取 ip 地址 (28-32 字节)
    for (int i = 28; i < 32; ++i) {
        printf("%d", packetData[i]);
        if (i < 31) cout << ".";
    }
    cout << endl;
    // 提取 MAC 地址 (6-12 字节)
    cout << "源 MAC 地址:\t ";
    for (int i = 6; i < 12; ++i) {
        printf("%02X", packetData[i]);
        if (i < 11) cout << "-";
    }
    cout << endl;
    //用 mac 数组记录本机的 MAC 地址
    for (int i = 0; i < 6; i++)
    {
        mac[i] = *(unsigned char*)(packetData + 22 + i);
    }
}

```



```

    }

    cout << "获取 MAC 地址成功, MAC 地址为: ";

    for (int i = 6; i < 12; ++i) {

        printf("%02X", packetData[i]);

        if (i < 11) cout << "-";

    }

    cout << endl;

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

    break;

}

```

三、实验结果

1.打印设备信息如图所示: 将所有通信网卡设备的名称及 IP 地址等相关信息打印在屏幕上

```

D:\visual studio2019\text2\project\Debug\project.exe
1. \Device\NPF_{9259FEE9-CC21-412F-B757-F31066109F26} 描述: (WAN Miniport (Network Monitor))
2. \Device\NPF_{147FE33F-2843-4C41-9EFB-79F7C3534AFE} 描述: (WAN Miniport (IPv6))
3. \Device\NPF_{C6D652FA-D3BF-42D5-8294-0B4F42EF2C9F} 描述: (WAN Miniport (IP))
4. \Device\NPF_{737A507A-D74D-479F-A24D-863208957E62} 描述: (Bluetooth Device (Personal Area Network))
IP地址: 169.254.49.246
子网掩码: 255.255.0.0
广播地址: 169.254.255.255
5. \Device\NPF_{3404C9FB-3EC3-430D-8173-42312FA72BBB} 描述: (Intel(R) Wi-Fi 6 AX201 160MHz)
IP地址: 192.168.43.26
子网掩码: 255.255.255.0
广播地址: 192.168.43.255
6. \Device\NPF_{6A0BD0C5-FABD-4A1C-B7E4-BEAF43E947C8} 描述: (VMware Virtual Ethernet Adapter for VMnet8)
IP地址: 192.168.188.1
子网掩码: 255.255.255.0
广播地址: 192.168.188.255
7. \Device\NPF_{735F97F0-1555-4D63-ACCF-C1A19658C5CE} 描述: (VMware Virtual Ethernet Adapter for VMnet1)
IP地址: 192.168.6.1
子网掩码: 255.255.255.0
广播地址: 192.168.6.255
8. \Device\NPF_{34981184-F0E2-4290-A428-9D9D28F6F6BC} 描述: (Microsoft Wi-Fi Direct Virtual Adapter #4)
IP地址: 169.254.120.126
子网掩码: 255.255.0.0
广播地址: 169.254.255.255
9. \Device\NPF_{Loopback} 描述: (Adapter for loopback traffic capture)
10. \Device\NPF_{58502A20-E66A-45C4-9D78-87060DD08C26} 描述: (Realtek PCIe GbE Family Controller)
IP地址: 192.168.1.88
子网掩码: 255.255.255.0
广播地址: 192.168.1.255
IP地址: 169.254.185.177
子网掩码: 255.255.0.0
广播地址: 169.254.255.255
-----
输入当前要连接的网卡序号:

```

2. 输入网卡的序号, 这里输入 vmware 虚拟机的网卡序号 6, 显示相对应网卡的信息以及 IP 地址和 Mac 地址

```

-----
输入当前要连接的网卡序号: 6
当前网络设备接口卡IP为: 192.168.188.1
当前网络设备接口卡名字为: \Device\NPF_{6A0BD0C5-FABD-4A1C-B7E4-BEAF43E947C8}
打开当前网络接口成功!!
加载中... 加载中... 加载中...
-----
ARP数据包内容:
源IP地址: 192.168.188.1
源MAC地址: 00-50-56-C0-00-08
获取MAC地址成功, MAC地址为: 00-50-56-C0-00-08
-----
请输入目的IP地址:

```

用 windows 的命令行 ipconfig/all 查询自己的 IP 地址和 mac 地址发现与抓捕到的 mac 地址相同

```
以太网适配器 VMware Network Adapter VMnet8:

   连接特定的 DNS 后缀 . . . . . : 
   描述. . . . . : VMware Virtual Ethernet Adapter for VMnet8
   物理地址. . . . . : 00-50-56-C0-00-08
   DHCP 已启用 . . . . . : 否
   自动配置已启用. . . . . : 是
   本地链接 IPv6 地址. . . . . : fe80::67db:4437:a57c:461f%10(首选)
   IPv4 地址. . . . . : 192.168.188.1(首选)
   子网掩码. . . . . : 255.255.255.0
   默认网关. . . . . : 
   DHCPv6 IAID. . . . . : 738218070
   DHCPv6 客户端 DUID . . . . . : 00-01-00-01-27-C2-D7-FE-7C-D3-0A-96-DE-0B
   TCP/IP 上的 NetBIOS . . . . . : 已启用
```

3. 输入目的 IP 地址为 192.168.188.131，这是我在虚拟机中搭建的网站服务器的 IP 地址，可以用来检测是否能正确获取到 mac 地址

```
-----
请输入目的IP地址: 192.168.188.131
加载中...加载中...加载中...加载中...加载中...加载中...加载中...加载中...加载中...加载中...加载中...加载中...加载中...
加载中...加载中...
-----
ARP数据包内容:
源IP地址: 192.168.188.1
源MAC地址: 00-50-56-C0-00-08
目的IP地址: 192.168.188.131
目的MAC地址: 00-0C-29-9B-0D-CC
获取MAC地址成功, MAC地址为: 00-0C-29-9B-0D-CC
-----
```

从虚拟机中查询 mac 地址如下，发现与抓捕到的 mac 地址也相同，实验成功

```
qiuqiu@qiuqiu-virtual-machine:~/桌面$ ip add show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:9b:0d:cc brd ff:ff:ff:ff:ff:ff
    altname eno2s1
    inet 192.168.188.131/24 brd 192.168.188.255 scope global dynamic noprefixroute ens33
        valid_lft 1620sec preferred_lft 1620sec
    inet6 fe80::dab5:b1e3:7af2:88d9/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

四、实验总结

- 1.对 NPcap 的架构有了更深入的了解与认识，学习到了 NPcap 的设备列表获取方法、网卡设备打开方法，以及数据包构造方法、数据包捕获方法，增强了动手实践的能力。
- 2.通过此次实验也发现一些问题，比如如果不设置#pragma pack(1)指定结构体

按照一个字节对齐，那么抓捕数据包会出现速度上的问题，较为浪费时间。

3. 对于 ARP 数据包的结构是需要重点记忆的，否则在拆分数数据包的时候就会恍惚到底拆分的是哪些数据，因此还需要在未来继续学习，继续实践。

五、实验源码

```
#define WIN32
#define HAVE_REMOTE
#include "pcap.h"
#include <iostream>
#include <WinSock2.h>
using namespace std;
#pragma comment(lib, "ws2_32.lib")
#pragma warning(disable:4996)
#pragma pack(1)
typedef struct Frame_Header//帧首部
{
    BYTE DesMAC[6]; //目的地址
    BYTE SrcMAC[6]; //源地址
    WORD FrameType; //帧类型
};
typedef struct ARP_Frame//ARP 数据
{
    Frame_Header FrameHeader;
    WORD HardwareType; //硬件类型
    WORD ProtocolType; //协议类型
    BYTE HLen; //硬件长度
    BYTE PLen; //协议长度
    WORD op; //操作类型
    BYTE SrcMAC[6]; //源 MAC 地址
    DWORD SrcIP; //源 IP 地址
    BYTE DesMAC[6]; //目的 MAC 地址
    DWORD DesIP; //目的 IP 地址
};

void* getaddress(struct sockaddr* sa)//得到对应的 IP 地址
{
    if (sa->sa_family == AF_INET)
    {
        return &(((struct sockaddr_in*)sa)->sin_addr); //IPv4 地址
    }
    return &(((struct sockaddr_in6*)sa)->sin6_addr); //IPv6 地址
}

int main() {
    /*获取设备列表，打印信息*/
```

```

pcap_if_t* d; //遍历用的指针
pcap_addr_t* a; //地址指针
pcap_if_t* devices; //指向设备列表第一个
int i = 0; //统计设备数量
char errbuf[PCAP_ERRBUF_SIZE]; //错误信息缓冲区
//输出错误信息
if (pcap_findalldevs(&devices, errbuf) == -1)
{
    cout << stderr << "查找设备失败: " << errbuf << endl;
    return 0;
}
//打印设备信息
//打印设备列表中设备信息
pcap_if_t* count; //遍历用的指针
char srcip[INET_ADDRSTRLEN]; //本机 ip
//输出设备名和描述信息
for (count = devices; count; count = count->next) //借助 count 指针从第一个设备开始访问到最后一个设备
{
    cout << ++i << ". " << count->name; //输出设备信息和描述
    if (count->description) {
        cout << "描述: (" << count->description << ")" << endl;
    }
    for (a = count->addresses; a != NULL; a = a->next) {
        if (a->addr->sa_family == AF_INET) {
            char str[INET_ADDRSTRLEN];
            inet_ntop(AF_INET, getaddress((struct sockaddr*)a->addr), str, sizeof(str)); //
            将 a->addr 强制转换为 struct sockaddr_in 类型的指针，并访问 sin_addr 成员，其中包含了 IPv4 地址。
            cout << "IP 地址: " << str << endl;
            inet_ntop(AF_INET, getaddress((struct sockaddr*)a->netmask), str, sizeof(str));
            //将 a->netmask 强制转换为 struct sockaddr_in 类型的指针，从 a->netmask 这个结构中提取子网掩码。
            cout << "子网掩码: " << str << endl;
            inet_ntop(AF_INET, getaddress((struct sockaddr*)a->broadaddr), str,
            sizeof(str)); //将 a->netmask 强制转换为 struct sockaddr_in 类型的指针，从 a->broadaddr 这个结构中提
            取广播地址。
            cout << "广播地址: " << str << endl;
        }
    }
}
//设备数量为 0
if (i == 0) {
    cout << endl << "存在错误! 无查找设备!" << endl;
    return 0;
}

```

```

cout << "-----" << endl;
/*选择设备及打开网卡*/
pcap_if_t* count2; //遍历用的指针 2
int num = 0;
cout << "输入当前要连接的网卡序号: ";
cin >> num;
while (num < 1 || num>11) {
    cout << "请检查网卡序号输入是否正确!" << endl;
    cout << "重新输入当前要连接的网卡序号: ";
    cin >> num;
}
count2 = devices;
for (int i = 1; i < num; i++) { //循环遍历指针选择第几个网卡
    count2 = count2->next;
}
inet_ntop(AF_INET, getaddress((struct sockaddr*)count2->addresses->addr), srcip,
sizeof(srcip));
//将 a->addr 强制转换为 struct sockaddr_in 类型的指针, 并访问 sin_addr 成员, 其中包含了 IPv4 地址。
cout << "当前网络设备接口卡 IP 为: " << srcip << endl << "当前网络设备接口卡名字为: " << count2->name << endl;
//打开网络接口
//指定获取数据包最大长度为 65536, 可以确保程序可以抓到整个数据包
//指定时间范围为 200ms
pcap_t* point = pcap_open(count2->name, 65536, PCAP_OPENFLAG_PROMISCUOUS, 200, NULL, errbuf);
if (point == NULL) {
    cout << "打开当前网络接口失败" << endl; //打开当前网络接口失败
    pcap_freealldevs(devices);
    return 0;
}
else {
    cout << "打开当前网络接口成功!! " << endl;
}
ARP_Frame send_ARPFrame;
//获取本机的 MAC 地址
//组装报文
unsigned char mac[48];
for (int i = 0; i < 6; i++) {
    send_ARPFrame.FrameHeader.DesMAC[i] = 0xFF; //DesMAC 设置为广播地址
    send_ARPFrame.DesMAC[i] = 0x00; //DesMAC 设置为 0
    //SrcMAC 用不到可以不设置
}
send_ARPFrame.FrameHeader.FrameType = htons(0x0806); //帧类型为 ARP
send_ARPFrame.HardwareType = htons(0x0001); //硬件类型为以太网

```

```

send_ARPFrame.ProtocolType = htons(0x0800); //协议类型为 IP
send_ARPFrame.HLen = 6; //硬件地址长度为 6
send_ARPFrame.PLen = 4; //协议地址长度为 4
send_ARPFrame.op = htons(0x0001); //操作为 ARP 请求
send_ARPFrame.DesIP = inet_addr(srcip); //设置为本机 IP 地址
struct pcap_pkthdr* pkt_header;
const u_char* packetData;
int ret;
while ((ret = pcap_next_ex(point, &pkt_header, &packetData)) >= 0)
{
    cout << "加载中...";
    //发送构造好的数据包
    pcap_sendpacket(point, (u_char*)&send_ARPFrame, sizeof(ARP_Frame));
    if (ret == 0) { //未捕获到数据包
        continue;
    }
    //通过报文内容比对判断是否是要发打印的 ARP 数据包内容
    //result=1, 捕获成功
    else if (*(unsigned short*)(packetData + 12) == htons(0x0806) //帧类型为 ARP (htons(0x0806))
        && *(unsigned short*)(packetData + 20) == htons(0x0002)) //操作类型为 ARP 响应
        (htons(0x0002))
    {
        cout << endl;
        cout << "-----" << endl;
        cout << "ARP 数据包内容: " << endl;
        //打印数据包
        cout << "源 IP 地址:\t ";
        for (int i = 28; i < 32; ++i) {
            printf("%d", packetData[i]);
            if (i < 31) cout << ".";
        }
        cout << endl;
        // 提取 MAC 地址 (0-6 字节)
        cout << "源 MAC 地址:\t ";
        for (int i = 6; i < 12; ++i) {
            printf("%02X", packetData[i]);
            if (i < 11) cout << "-";
        }
        cout << endl;
        //用 mac 数组记录本机的 MAC 地址
        for (int i = 0; i < 6; i++)
        {
            mac[i] = *(unsigned char*)(packetData + 22 + i);
        }
    }
}

```

```

        cout << "获取 MAC 地址成功, MAC 地址为: ";
        for (int i = 6; i < 12; ++i) {
            printf("%02X", packetData[i]);
            if (i < 11) cout << "-";
        }
        cout << endl;
        cout << "-----" << endl;
        break;
    }
}

//输出错误信息
if (ret == -1) { //调用过程发生错误
    cout << "捕获数据包出错" << endl;
    pcap_freealldevs(devices);
    return 0;
}

ARP_Frame rev_ARPFrame;
/*获取目的主机的 MAC 地址*/
for (int i = 0; i < 6; i++) {
    rev_ARPFrame.FrameHeader.DesMAC[i] = 0xff; //广播地址
    rev_ARPFrame.FrameHeader.SrcMAC[i] = mac[i]; //本机 MAC 地址
    rev_ARPFrame.DesMAC[i] = 0x00; //设置为 0
    rev_ARPFrame.SrcMAC[i] = mac[i]; //本机 MAC 地址
}

rev_ARPFrame.FrameHeader.FrameType = htons(0x0806);
rev_ARPFrame.HardwareType = htons(0x0001);
rev_ARPFrame.ProtocolType = htons(0x0800);
rev_ARPFrame.HLen = 6;
rev_ARPFrame.PLen = 4;
rev_ARPFrame.op = htons(0x0001);
rev_ARPFrame.SrcIP = inet_addr(srcip);
cout << "请输入目的 IP 地址: ";
char ip[INET_ADDRSTRLEN];
cin >> ip;
rev_ARPFrame.DesIP = inet_addr(ip);
while ((ret = pcap_next_ex(point, &pkt_header, &packetData)) >= 0) //判断获取报文
{
    //发送构造好的数据包
    cout << "加载中...";
    pcap_sendpacket(point, (u_char*)&rev_ARPFrame, sizeof(ARP_Frame));
    if (ret == 0) { //未捕获到数据包
        continue;
    }
}

//result=1, 捕获成功

```



```

else if (*(unsigned short*)(packetData + 12) == htons(0x0806) //帧类型为 ARP (htons(0x0806))
    && *(unsigned short*)(packetData + 20) == htons(0x0002) //操作类型为 ARP 响应
(htons(0x0002))
    && *(unsigned long*)(packetData + 28) == rev_ARPFrame.DesIP)//ip 地址为填入的目标 IP
地址
{
    cout << endl;
    cout << "-----" << endl;
    cout << "ARP 数据包内容: " << endl;
    //打印数据包
    cout << "源 IP 地址:\t ";
    for (int i = 38; i < 42; ++i) {
        printf("%d", packetData[i]);
        if (i < 41) cout << ".";
    }
    cout << endl;
    // 提取 MAC 地址 (0-6 字节)
    cout << "源 MAC 地址:\t ";
    for (int i = 0; i < 6; ++i) {
        printf("%02X", packetData[i]);
        if (i < 5) cout << "-";
    }
    cout << endl;
    cout << "目的 IP 地址:\t ";
    for (int i = 28; i < 32; ++i) {
        printf("%d", packetData[i]);
        if (i < 31) cout << ".";
    }
    cout << endl;
    // 提取目的 MAC 地址 (后 6 字节)
    cout << "目的 MAC 地址:\t ";
    for (int i = 6; i < 12; ++i) {
        printf("%02X", packetData[i]);
        if (i < 11) cout << "-";
    }
    cout << endl;
    cout << "获取 MAC 地址成功, MAC 地址为: ";
    for (int i = 6; i < 12; ++i) {
        printf("%02X", packetData[i]);
        if (i < 11) cout << "-";
    }
    cout << endl;
    cout << "-----" << endl;
}

```

```
        break;
    }
}

//输出错误信息
if (ret == -1) { //调用过程发生错误
    cout << "捕获数据包出错" << endl;
    pcap_freealldevs(devices);
    return 0;
}

// 关闭设备
pcap_close(point);
pcap_freealldevs(devices);
return 0;
}
```