南开大学
Nankai University

南　　开　　大　　学

网 络 空 间 安 全 学 院

网络技术与应用

IP 数据报捕获与分析

学号：2013018

姓名：许健

年级：2020 级

专业：信息安全

2022 年 10 月 19 日

目录

一、 实验内容说明

(一) 实验题目

编程实验：IP 数据包捕获与分析 (利用 npcap 编程捕获数据包)

(二) 实验说明

1. 了解 NPcap 的架构
2. 学习 NPcap 的设备列表获取方法、网卡设备打开方法，以及数据包捕获方法
3. 显示捕获数据帧的源 MAC 地址和目的 MAC 地址，以及类型/长度字段的值
4. 编写的程序应结构清晰，具有较好的可读性

(三) 实验准备

Npcap 是一个开源的、运行于 Windows 的数据包捕获与发送函数库，主要功能包括：数据包捕获、发送和网络分析。包含内核级、低层次的包过滤动态连接库 Packet.dll 和高级别系统无关函数库 wpcap.dll。

使用 VS2019 新建项目，使用 Npcap 编程需要以下三步：

1. 添加 pcap.h 包含文件并添加包含文件目录
2. 添加库文件目录
3. 添加链接时使用的库文件 Packet.lib 和 wpcap.lib

二、 实验过程

(一) 获取设备列表，打开网卡设备

PrintAlldevs 函数

```
void PrintAlldevs() {
        cout << "输出网卡的信息" << endl;
        // 获取本地机器设备列表
        if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &alldevs, errbuf)
            == -1)
            printf("error\n");
        if(alldevs == nullptr)
            printf("No interfaces found.");

        /* 打印网卡信息列表 */
        for (d = alldevs; d != NULL; d = d->next)
                printf("ID: %d %s --> Name: %s \n", index, d->name, d->
                    description);
    pcap_freealldevs(alldevs);
}
```

使用 pcap_findalldevs_ex 函数可以获取本机设备列表,如果返回值为-1 则发生错误。alldevs 是指向设备列表的指针，类型为 pcap_if，如果是 nullptr 说明没有设备发现。可以使用 for 循环打印每个网卡的 name 和 description 信息，最后要使用 pcap_freealldevs 函数释放设备列表。

除此之外，我还尝试获取每个网络接口设备的 IP 地址信息，对其进行了探究。获取到的 IP 地址和 Windows 下使用 ipconfig 输出的一样，而没有 IP 地址的网卡则打印媒体已断开连接。感到意外的是，使用相同的方式获取子网掩码和广播地址的时候，即便我打印出来 sockaddr 结构体的字节序列，看到的也全是 00，结果如图1所示，对此我感到困惑，这是我待解决的问题。

```
输出网卡的信息
ID: 1 rpcap://{3BEC3EE6-0B6D-4E84-951B-0F5B2E5811F6} --> Name: Network adapter 'VMware Virtual Ethernet Adapter' on local host
        addr: 192.168.31.1
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ID: 2 rpcap://{3DEEC827-B66E-4FEC-8CC1-8C11ADBD40EE} --> Name: Network adapter 'Microsoft' on local host
        addr: 10.130.226.195
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ID: 3 rpcap://{9D6DDF74-3400-4EC5-A770-15F92734BC64} --> Name: Network adapter 'Microsoft' on local host
        媒体已断开连接
ID: 4 rpcap://{A41D6819-9241-4AAE-811C-370A23B4AFEE} --> Name: Network adapter 'Microsoft' on local host
        媒体已断开连接
ID: 5 rpcap://{60C5D0AF-3F7D-45CA-8BC9-DBFECC4E7202} --> Name: Network adapter 'Oracle' on local host
        addr: 192.168.56.1
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ID: 6 rpcap://{AECA73A3-B338-4F8E-B6DC-4BC8FC8B0109} --> Name: Network adapter 'Microsoft Corporation' on local host
        addr: 172.18.192.1
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ID: 7 rpcap://{594D3E41-AA04-4DE0-9335-A6BDBAF252DB} --> Name: Network adapter 'VMware Virtual Ethernet Adapter' on local host
        addr: 192.168.231.1
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ID: 8 rpcap://{04112D3C-030D-4DD2-BC86-BFF368CEB3A5} --> Name: Network adapter 'Microsoft' on local host
        媒体已断开连接
ID: 9 rpcap://{7367811A-28CC-4C1C-BBB0-56CADCC8CB0C} --> Name: Network adapter 'Realtek PCIe GbE Family Controller' on local host
        媒体已断开连接
```

图 1: 获取设备列表，打印网卡信息

打印网络接口设备的 IP 地址信息

```
for (a = d->addresses; a != nullptr; a = a->next) {
        sockaddr_in* tmp;
        u_char b1, b2, b3, b4;
        if (a->addr->sa_family == AF_INET) {  //判断该地址是否IP地址
                tmp = (sockaddr_in*)(a->addr);
                b1 = tmp->sin_addr.S_un.S_un_b.s_b1;
                b2 = tmp->sin_addr.S_un.S_un_b.s_b2;
                b3 = tmp->sin_addr.S_un.S_un_b.s_b3;
                b4 = tmp->sin_addr.S_un.S_un_b.s_b4;
                if (b1 != 0) {
                        printf("        addr: %d.%d.%d.%d\n", b1, b2, b3, b4)
                            ;
                        if (a->netmask) {
                                u_char* p = (u_char*)((sockaddr_in*)(a->
                                    netmask));
                                for (int k = 0; k < 16; k++) {
                                        printf("%02x ", *(p + k));
                                }
                                cout << endl;
                        }
                        if (a->broadaddr) {
                                u_char* p = (u_char*)((sockaddr_in*)(a->
                                    broadaddr));
```

```
                                for (int k = 0; k < 16; k++) {
                                        printf("%02x ", *(p + k));
                                }
                                cout << endl;
                        }
                }
                else{
                        cout << "        媒体已断开连接" << endl;
                }
        }
}
```

(二) 监听网络设备，捕获 IP 数据包

首先使用 pcap_findalldevs_ex 获取设备列表,然后找到指定的网卡。使用 pcap_open 函数打开网卡, 返回一个 pcap_t* 类型的 handle 句柄。捕获数据包采用直接捕获函数 pcap_next_ex(): read_timeout 到时返回。

对于大小不足 64 字节的 MAC 帧则直接丢弃，我尝试打印出 MAC 帧的内容，作进一步的的分析，打印的字段包括:MAC 帧的源地址和目的地址、类型字段，IP 数据报的源 IP 地址和目的 IP 地址。最后我尝试使用 wireshark，将抓到的包与之对比，确认信息无误。

打印过程中用到了网络序转换函数 ntohs。

MonitorAdapter

```
if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &alldevs, errbuf) == -1)
        return;
for (int x = 0; x < nChoose - 1; ++x)// 找到指定的网卡
        alldevs = alldevs->next;
pcap_t* handle = pcap_open(alldevs->name, 65534, PCAP_OPENFLAG_PROMISCUOUS,
    1, 0, 0);
if (alldevs == NULL)
        return;
while ((retvalue = pcap_next_ex(handle, &pkt_header, &pkt_data)) >= 0)
{
        if (retvalue == 0)
                continue;
        IPPacket = (Data_t*)pkt_data;
        cout << "包大小:  " << pkt_header->len << "字节" << endl;
        if (pkt_header->len < 64) {
                cout << "MAC帧不足64字节, 丢弃" << endl << endl;
                continue;
        }
        for (int k = 0; k < 60; k++) {//输出每个包的前60个字节数据
                if (k % 15 == 0 && k != 0)//输出美观
                        printf("\n");
                printf("%02x ", *(pkt_data + k));
        }
        cout << endl;
```

```
24          u_char* p = (u_char*)(pkt_data)+(pkt_header->len - 1) - 4;
25          SourceMAC = IPPacket->FrameHeader.SrcMAC;
26          DestinationMAC = IPPacket->FrameHeader.DesMAC;
27          u_short ether_type = ntohs(IPPacket->FrameHeader.FrameType);   // 以太
                网类型
28          printf("类型: 0x%04x \t", ether_type);
29          printf("源MAC地址: %02X:%02X:%02X:%02X:%02X:%02X \t",
30                  SourceMAC[0], SourceMAC[1], SourceMAC[2], SourceMAC[3],
                        SourceMAC[4], SourceMAC[5]);
31          printf("目标MAC地址: %02X:%02X:%02X:%02X:%02X:%02X \t",
32                  DestinationMAC[0], DestinationMAC[1], DestinationMAC[2],
                        DestinationMAC[3], DestinationMAC[4], DestinationMAC[5]);
33          ULONG           SourceIP, DestinationIP;
34          u_char* source = (u_char*)(&(IPPacket->IPHeader.SrcIP));
35          u_char* destination = (u_char*)(&(IPPacket->IPHeader.DstIP));
36          printf("源IP地址: %d.%d.%d.%d\t", source[0], source[1], source[2],
                source[3]);
37          printf("目的IP地址: %d.%d.%d.%d\n\n", destination[0], destination[1],
                destination[2], destination[3]);
38  }
39  pcap_freealldevs(alldevs);
```

实验结果如图2所示，一个有意思的话题是，无论是使用 wireshark 还是 Npcap 编程捕获到的数据包貌似都不带有 CRC 校验码，可能是 Wireshark 抓包前，在物理层网卡已经去掉了一些之前几层加的东西，比如前导同步码，FCS 等等，之后利用校验码 CRC 校验，正确时才会进行下一步操作，这时才开始进行抓包。因此，抓包软件抓到的是去掉前导同步码、FCS 之外的数据，没有校验字段。

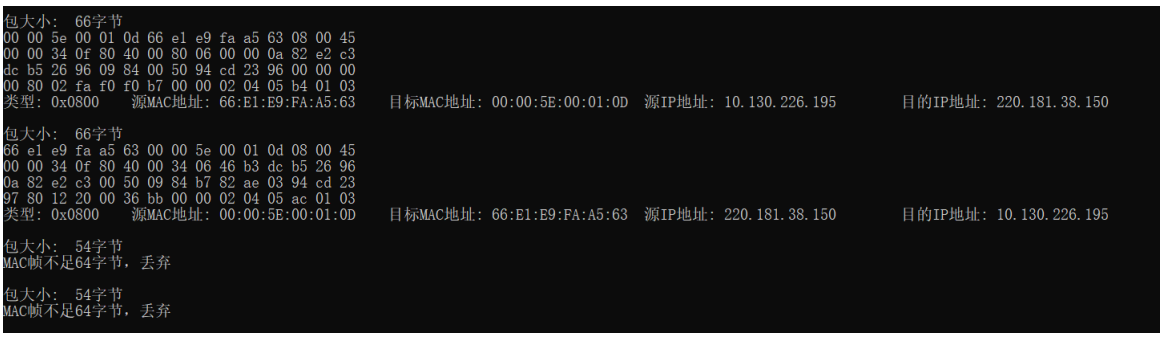

图 2: 捕获 IP 数据包

三、 附录

(一) 后续工作

未来改进包括：解决实验中遇到的问题、添加校验字段、编写可视化界面、对网络技术做进一步的探究等，同时在此感谢达益鑫助教的帮助和指导。

(二) 实验中用到的数据结构

MAC 帧和 IP 数据包头部

```
#pragma pack(1)          //进入字节对齐方式
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;
typedef struct IPHeader_t {              //IP首部
        BYTE    Ver_HLen;
        BYTE    TOS;
        WORD    TotalLen;
        WORD    ID;
        WORD    Flag_Segment;
        BYTE    TTL;
        BYTE    Protocol;
        WORD    Checksum;
        ULONG   SrcIP;
        ULONG   DstIP;
} IPHeader_t;
typedef struct Data_t { //包含帧首部和IP首部的数据包
        FrameHeader_t   FrameHeader;
        IPHeader_t              IPHeader;
} Data_t;
#pragma pack()   //恢复缺省对齐方式
```

网络设备数据结构

```
typedef struct pcap_if pcap_if_t;
struct pcap_if {
        struct pcap_if *next;
        char *name;
        char *description;
        struct pcap_addr *addresses;
        u_int flags;
};

struct pcap_addr {
        struct pcap_addr *next;
        struct sockaddr *addr;/* IP地址 */
        struct sockaddr *netmask;/* 网络掩码 */
        struct sockaddr *broadaddr;/* 广播地址 */
        struct sockaddr *dstaddr;/* 目的地址 */
};

struct sockaddr {
        u_short sa_family;              /* address family */
        char    sa_data[14];            /* up to 14 bytes of direct address
            */
```

```
};

struct sockaddr_in {
        short   sin_family; //2字节，地址族，socket编程中只能是AF_INET
        u_short sin_port; //2字节，绑定的端口（网络字节序）
        struct  in_addr sin_addr; //4字节，IP地址结构(s_addr按照网络字节顺序存储IP地址)
        char    sin_zero[8];//sin_zero是为了让sockaddr与sockaddr_in两个数据结构保持大小相同而保留的空字节。
};

struct in_addr {
        union {
                struct { u_char s_b1,s_b2,s_b3,s_b4; } S_un_b;
                struct { u_short s_w1,s_w2; } S_un_w;
                u_long S_addr;
        } S_un;//注意，这里是一个联合，通常在用的时候用S_addr
```

字节顺序

```
网络序→主机序
u_short ntohs(u_short netshort)
u_long ntohl(u_long netlong)
主机序→网络序
u_short htons(u_short hostshort)
u_long htonl(u_long hostlong)
```