

网络技术与应用课程报告

第二次实验报告

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

IP数据报捕获与分析编程实验

要求如下：

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. 通过NPcap编程，实现本机的IP数据报捕获，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. 捕获的数据报不要求硬盘存储，但应以简单明了的方式在屏幕上显示。必显字段包括源MAC地址、目的MAC地址和类型/长度字段的值
5. 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从特定的格式。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。

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，即输出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。

所以按照课本上的要求，依次完成：

1. 获取设备列表
2. 打开想要嗅探的设备
3. 捕获数据包

需要注意的是我们需要按照书上的介绍，将捕获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：

```
#pragma pack(1)

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 Protocal;
    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()
```

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，比如IPPacket->FrameHeader.SrcMAC、IPPacket->FrameHeader.DesMAC、IPPacket->FrameHeader.FrameType等，但一定要注意格式。

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，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：

```
pcap_if_t *alldevs;
pcap_if_t *d;
int interface_num; // 先声明之后用户选择要用到的端口号
int i=0;
pcap_t *adhandle;
char errbuf[PCAP_ERRBUF_SIZE];

/* 获取本机设备列表 */
if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &alldevs, errbuf) ==
-1)
{
    fprintf(stderr, "Error in pcap_findalldevs: %s\n", errbuf);
    exit(1);
}
```

接下来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，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返回手动设置输出的信息，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：

```
/* 打印列表 */
for(d = alldevs; d; 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 WinPcap is
installed.\n");
    return -1;
}

printf("Enter the interface number which you like between (1 —
%d):", i); // 输入你想要监听的接口
scanf("%d", &interface_num);

if(interface_num < 1 || interface_num > i)
{
    printf("\nInterface number out of range.\n");
}
```

```

        /* 释放设备列表 */
        pcap_freealldevs(alldevs);
        return -1;
    }

    /* 跳转到选中的适配器 */
    for(d = alldevs, i=0; i< interface_num - 1 ; d = d->next, i++);

    /* 打开设备 */
    if ( (adhandle = pcap_open(d->name,          // 设备名
        65535,                                // 65535保证能捕获到不同数据链路层上的每个数据包的全部内
容
        PCAP_OPENFLAG_PROMISCUOUS,          // 混杂模式
        1000,                                // 读取超时时间
        NULL,                                // 远程机器验证
        errbuf                                // 错误缓冲池
        ) ) == NULL)
    {
        fprintf(stderr, "\nUnable to open the adapter. %s is not supported
by WinPcap\n", d->name);
        /* 释放设备列表 */
        pcap_freealldevs(alldevs);
        return -1;
    }

    printf("\nlistening on %s...\n", d->description);

```

最后是捕获数据包，本次实验用的是pcap_loop函数来捕获并产生回调信息，在这个函数中会调用另一个函数并一直循环，这也是它叫做loop的原因，相当于开启了又一个线程，用其开启自定义的packet_handler函数，这个函数的参数本人是仿照标准获取信息的参数来设置的，并在函数体中首先对时间输出标准化，输出时间、长度的相关信息，接下来对源MAC地址和目的MAC地址，以及类型/长度字段进行输出，在输出源MAC地址和目的MAC地址的时候需要使用%02x获得统一格式的输出生，在输出类型的时候要对捕获到的具体数值做出判断，通过switch来确定其究竟是哪一种类型，并输出相应字段的值，实现的具体代码如下：

```

/* 每次捕获到数据包时，libpcap都会自动调用这个回调函数 */
void packet_handler(u_char* param, const struct pcap_pkthdr* header, const
u_char* pkt_data)
{
    struct tm* ltime;
    char timestr[16];
    time_t local_tv_sec;

    /* 将时间戳转换成可识别的格式 */
    local_tv_sec = header->ts.tv_sec;
    ltime = localtime(&local_tv_sec);
    strftime(timestr, sizeof timestr, "%H:%M:%S", ltime);

    printf("%s,%.6ld len:%d\n", timestr, header->ts.tv_usec, header->len);

    // 检验校验和

```

```

Data_t* IPPacket;
WORD RecvChecksum;

IPPacket = (Data_t*)pkt_data;

RecvChecksum = IPPacket->IPHeader.Checksum;
// 需要编码的地方, 仿照校验和, 写输出语句, 输出源MAC地址、目的MAC地址和类型/长度字
段的值的语句
/*char src[6];
char des[6];
for (int j = 0; j < 6; j++)
{
    src[j] = (char)IPPacket->FrameHeader.SrcMAC[j];
    des[j] = (char)IPPacket->FrameHeader.DesMAC[j];
}
char type = (char)IPPacket->FrameHeader.FrameType;*/

//char* src = (char*)IPPacket->FrameHeader.SrcMAC;
//char* des = (char*)IPPacket->FrameHeader.DesMAC;
//char* type = (char*)IPPacket->FrameHeader.FrameType;

printf("-----\n");
//printf("数据报的源MAC地址为%s, 数据报的目的MAC地址为%s, 其类型为%s\n", src,
des, type);
//std::cout << src[0] << src[1] << src[2] << src[3] << src[4] <<
src[5] << std::endl;
//std::cout << des[0] << des[1] << des[2] << des[3] << des[4] <<
des[5] << std::endl;
printf("数据报的源MAC地址为: %02x:", IPPacket->FrameHeader.SrcMAC[0]);
printf("%02x:", IPPacket->FrameHeader.SrcMAC[1]);
printf("%02x:", IPPacket->FrameHeader.SrcMAC[2]);
printf("%02x:", IPPacket->FrameHeader.SrcMAC[3]);
printf("%02x:", IPPacket->FrameHeader.SrcMAC[4]);
printf("%02x\n", IPPacket->FrameHeader.SrcMAC[5]);

printf("数据报的目的MAC地址为: %02x:", IPPacket->FrameHeader.DesMAC[0]);
printf("%02x:", IPPacket->FrameHeader.DesMAC[1]);
printf("%02x:", IPPacket->FrameHeader.DesMAC[2]);
printf("%02x:", IPPacket->FrameHeader.DesMAC[3]);
printf("%02x:", IPPacket->FrameHeader.DesMAC[4]);
printf("%02x\n", IPPacket->FrameHeader.DesMAC[5]);

u_short ethernet_type;
ethernet_type = ntohs(IPPacket->FrameHeader.FrameType);

printf("其类型/长度字段的值为: %04x\n", ethernet_type);

cout<<"其类型为: ";
switch (ethernet_type)
{
case 0x0800:
    cout << "IP";

```

```

        break;
    case 0x0806:
        cout << "ARP";
        break;
    case 0x0835:
        cout << "RARP";
        break;
    default:
        cout << "Unknown Protocol";
        break;
}

printf("\n");

printf("-----\n");
}

```

最后需要释放设备列表，即可退出进程。

(3)结果展示

首先是捕获到设备接口，打印设备列表的界面：

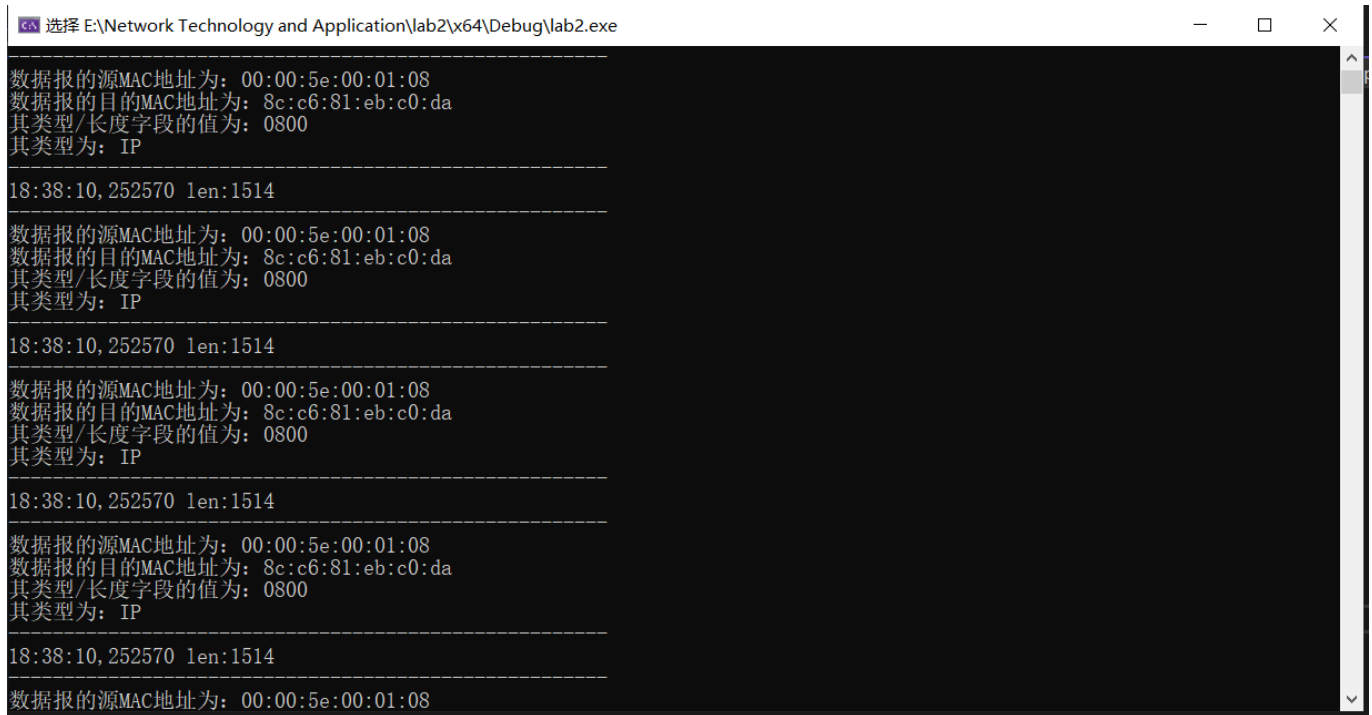
```

选择 E:\Network Technology and Application\lab2\x64\Debug\lab2.exe
1. rpcap://Device\NPF_{F6317548-93C6-4DD2-BA8F-91B099488C64} (Network adapter 'WAN Miniport (IPv6)' on local host)
2. rpcap://Device\NPF_{820A4078-2712-4803-97BE-BF1B3DB447DF} (Network adapter 'WAN Miniport (IP)' on local host)
3. rpcap://Device\NPF_{3BA1B06A-853F-4818-89BF-0B7AC1332403} (Network adapter 'WAN Miniport (Network Monitor)' on local host)
4. rpcap://Device\NPF_{0793C35D-A088-4F4A-922F-BB424BEC2AFE} (Network adapter 'Bluetooth Device (Personal Area Network)' on local host)
5. rpcap://Device\NPF_{CC718990-47D6-431B-A999-BCFC2A0FFFEA} (Network adapter 'Intel(R) Wi-Fi 6 AX200 160MHz' on local host)
6. rpcap://Device\NPF_{50A9F634-1BC5-445A-8E80-1F115AFBB71E} (Network adapter 'VMware Virtual Ethernet Adapter for VMnet8' on local host)
7. rpcap://Device\NPF_{BBF355C3-40FF-4BA8-9BA3-F64EFD8BF5A7} (Network adapter 'VMware Virtual Ethernet Adapter for VMnet1' on local host)
8. rpcap://Device\NPF_{2903E1B8-2ED8-497F-9874-10BF9E1DF552} (Network adapter 'Microsoft Wi-Fi Direct Virtual Adapter #3' on local host)
9. rpcap://Device\NPF_{39F5F915-DFD9-4A36-BFBD-DFB18281451C} (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_{9C749BB7-AA76-4104-9A99-AF64FFD322F9} (Network adapter 'TAP-Windows Adapter V9' on local host)
12. rpcap://Device\NPF_{686A0F79-665F-475E-BCA2-DC82EA050075} (Network adapter 'Realtek PCIe GbE Family Controller' on local host)
Enter the interface number which you like between (1 —— 12):5

listening on Network adapter 'Intel(R) Wi-Fi 6 AX200 160MHz' on local host...
18:38:10,252570 len:1514
-----
数据报的源MAC地址为: 00:00:5e:00:01:08
数据报的目的MAC地址为: 8c:c6:81:eb:c0:da
其类型/长度字段的值为: 0800
其类型为: IP
-----

```

然后是输出数据报信息的画面：



```
选择 E:\Network Technology and Application\lab2\x64\Debug\lab2.exe

-----
数据报的源MAC地址为: 00:00:5e:00:01:08
数据报的目的MAC地址为: 8c:c6:81:eb:c0:da
其类型/长度字段的值为: 0800
其类型为: IP
-----
18:38:10, 252570 len:1514
-----
数据报的源MAC地址为: 00:00:5e:00:01:08
数据报的目的MAC地址为: 8c:c6:81:eb:c0:da
其类型/长度字段的值为: 0800
其类型为: IP
-----
18:38:10, 252570 len:1514
-----
数据报的源MAC地址为: 00:00:5e:00:01:08
数据报的目的MAC地址为: 8c:c6:81:eb:c0:da
其类型/长度字段的值为: 0800
其类型为: IP
-----
18:38:10, 252570 len:1514
-----
数据报的源MAC地址为: 00:00:5e:00:01:08
数据报的目的MAC地址为: 8c:c6:81:eb:c0:da
其类型/长度字段的值为: 0800
其类型为: IP
-----
18:38:10, 252570 len:1514
-----
数据报的源MAC地址为: 00:00:5e:00:01:08
```

四、特殊现象分析

本次实验中确实遇到了一个难点，但并不是在网络的获取设备列表或者是嗅探数据报的层面，而是对各种东西进行输出的时候不知道怎么输出。

因为之前从来没有用过byte、word等数据，以前写程序基本上都是int、char、string等类型，所以基本上没有注意过格式输出，一开始在输出的时候直接卡住，退出进程。在找到问题的根源是输出的时候，想的是因为byte型的数据本来在c++里就是拿unsigned char来定义的，所以直接强制类型转换不就行了。果然，强制转换后程序并没有卡住，而是一堆的乱码。

接下来就开始在CSDN上查阅资料，但基本上都是类似强制类型转换的方法，试过后还是乱码，然后又去向学长求助，得知可以用%x来格式化后，随即改变程序，得到了看起来有模样的数据，但是这样子打印的话会使得输出的位数不全相同，因为有些小于0x10的十六进制数输出就会只输出一位，所以将%x换成%02x就可以获得统一格式的输出。最后再对数据格式工整化，本次的实验基本上也就大功告成了。

五、总结与展望

(1)总结

本次实验是网络技术与应用的第二次实验，一开始对于捕获数据包方面的知识并不太了解，甚至都从来没有用过wireshark软件，但是通过动手编程后，不仅对数据报的架构、npcap/winpcap的架构、捕获数据包的细节等方面有了更深层的认识，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，期望自己在这学期未来实验的更好的发展。