# 实验2:数据包捕获与分析



# 一、前期准备

## 1、了解NPcap的架构和工作原理

Npcap是一个为Microsoft Windows操作系统设计的网络数据包捕获和分析架构，它由软件库和网络驱动程序组成。Npcap的目的是为Windows应用程序提供对网络数据包的直接访问，即所谓的“原始”数据，这些数据在操作系统的协议处理介入之前就已经被捕获。

### （1）Npcap的架构

Npcap的架构包括以下几个关键部分：

1. **设备驱动程序（NPF）**：Npcap实现了一个名为NPF（Netgroup Packet Filter）的Windows内核驱动程序，该驱动程序负责从Win10 miniport驱动获取网卡数据，实现监控网络数据包的功能。NPF驱动工作在Windows内核的网络部分，允许捕获和发送原始网络数据包。
2. **DLLs（动态链接库）**：Npcap安装了几个DLL，包括`wpcap.dll`和`npcap.dll`，这些库文件提供了与NPF驱动程序交互的接口。
3. **API（应用程序编程接口）**：Npcap通过libpcap API提供了一系列函数和结构，这些API使得应用程序可以捕获原始数据包、过滤数据包、传输数据包以及收集网络流量的统计信息。Npcap的API是libpcap的一个扩展，它包括了一些Windows特有的扩展函数，如`pcap_setbuff`、`pcap_setmode`、`pcap_setmintocopy`等。
4. **用户空间工具**：Npcap提供了一些用户空间的工具，如`Packet.dll`，这些工具可以帮助开发者在应用程序中更容易地使用Npcap的功能。

### （2）Npcap的工作原理

Npcap的工作原理可以概括为以下几个步骤：

1. **安装和配置**：用户需要在Windows系统上安装Npcap，这通常涉及到运行一个安装程序来安装NPF驱动和相关的DLLs。
2. **获取设备列表**：应用程序可以通过调用Npcap的API来获取系统上所有可用的网络设备的列表。这是通过`pcap_findalldevs`或`pcap_findalldevs_ex`函数实现的。
3. **打开设备**：一旦选择了一个网络设备，应用程序可以通过调用`pcap_open_live`函数来打开该设备，准备开始捕获数据包。
4. **设置过滤器**：在开始捕获之前，应用程序可以设置一个过滤器，以便只捕获符合特定条件的数据包。这是通过`pcap_compile`和`pcap_setfilter`函数实现的。
5. **捕获数据包**：应用程序可以通过调用`pcap_loop`或`pcap_next_ex`函数来捕获数据包。这些函数会将数据包从网络适配器读取到应用程序中。
6. **处理数据包**：捕获到的数据包可以在用户空间进行处理，例如分析、统计或者显示数据包的内容。
7. **关闭设备**：完成数据包捕获后，应用程序应该通过调用`pcap_close`函数来关闭网络设备，释放相关资源。

## 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。

### (1) 设备列表获取方法

```
int pcap_findalldevs_ex(
        char *source,
        struct  pcap_rmtauth auth,
        pcap_if_t **alldevs,
        char *errbuf
);

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;
        struct sockaddr *netmask;
        struct sockaddr *broadaddr;
        struct sockaddr *dstaddr;
};

pcap_if_t   *alldevs;                   //指向设备链表首部的指针
pcap_if_t   *d;
pcap_addr_t *a;
```

```
char        errbuf[PCAP_ERRBUF_SIZE];   //错误信息缓冲区
//获得本机的设备列表
if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING,     //获取本机的接口设备
        NULL,                   //无需认证
        &alldevs,               //指向设备列表首部
        errbuf                 //出错信息保存缓存区
        ) == -1)
{   ……  //错误处理      }

for(d= alldevs; d != NULL; d= d->next)      //显示接口列表
{
    ……  //利用d->name获取该网络接口设备的名字
    ……  //利用d->description获取该网络接口设备的描述信息
    //获取该网络接口设备的IP地址信息
    for(a=d->addresses; a!=NULL; a=addr->next)
        if (a->addr->sa_family==AF_INET)  //判断该地址是否IP地址
        {
            ……  //利用a->addr获取IP地址
            ……  //利用a->netmask获取网络掩码
            ……  //利用a->broadaddr获取广播地址
            ……  //利用a->dstaddr)获取目的地址
        }
}

//释放设备列表
pcap_freealldevs(alldevs);
```

## (2) 网卡设备打开方法

```
pcap_t* pcap_open(
    const char *source,
    int snaplen,
    int flags,
    int read_timeout,
    struct pcap_rmtauth *auth,
    char *errbuf
};
```

## (3) 数据包捕获方法

```
//利用回调函数捕获
pcap_dispatch()：read_timeout到时返回
pcap_loop()：捕获到cnt个数据包后返回
//直接捕获
pcap_next_ex()：read_timeout到时返回
```

# 3、实验环境配置

## (1) 实验环境

C++、Visual Studio 2022

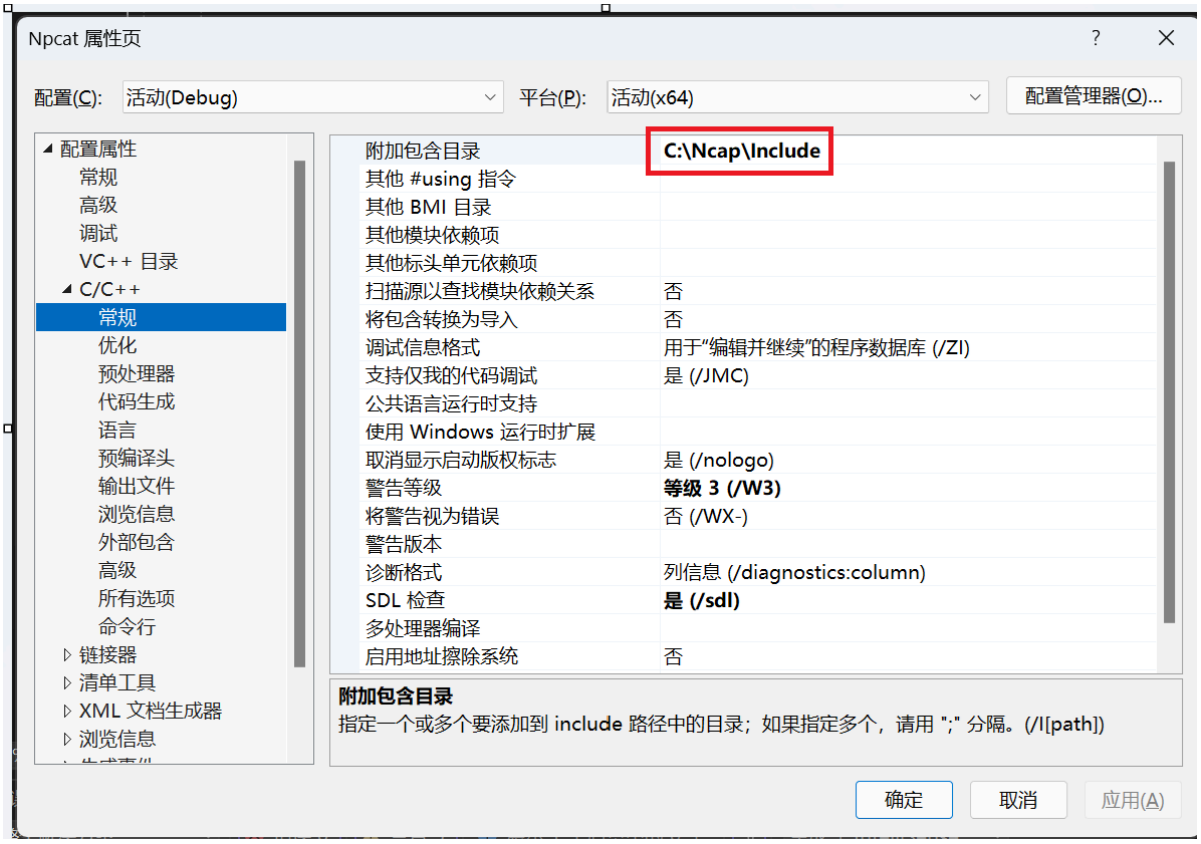
## (2) 添加pcap.h包含文件

所有使用Npcap函数的源文件中都需添加pcap.h包含文件：#include "pcap.h

> 注意：要先去Winpcap下载开发包

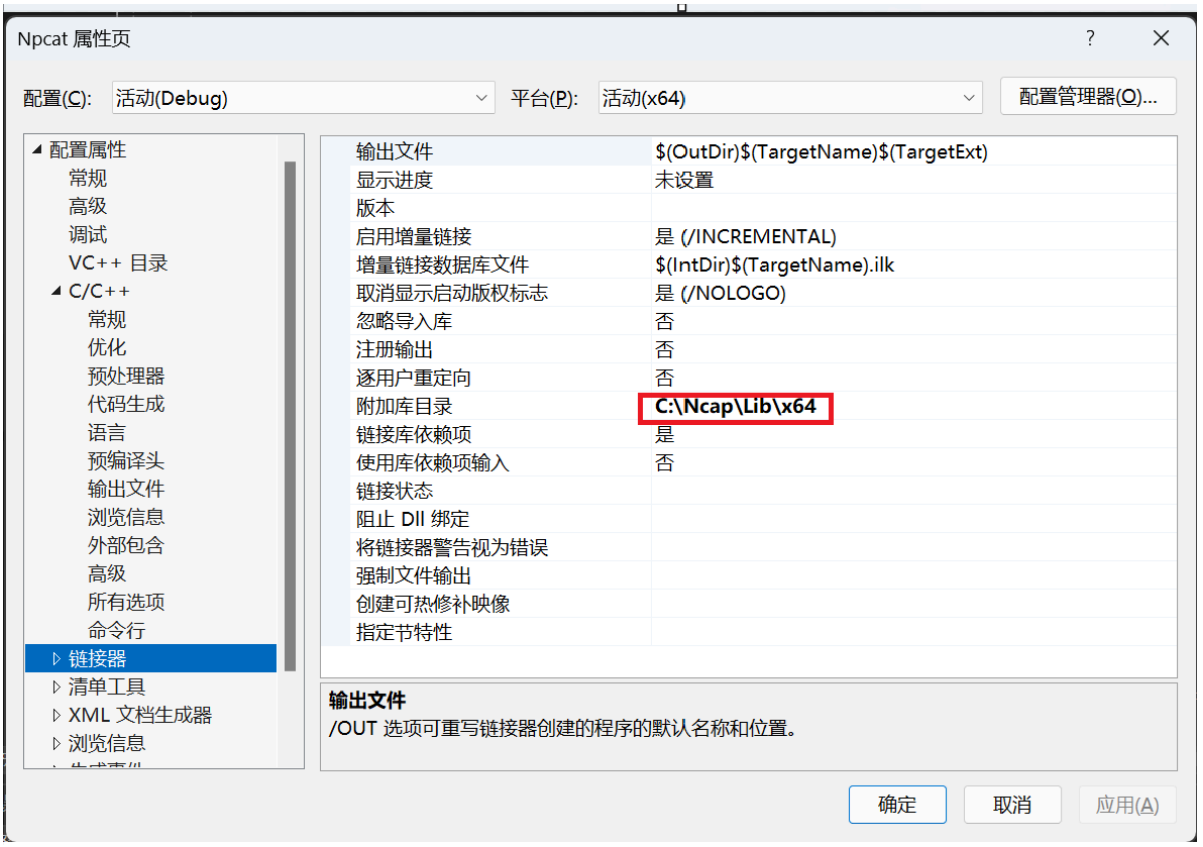
## (3) 添加包含文件目录

项目 - 属性 - 配置属性 - C/C++ - 常规 - 附加包含目录

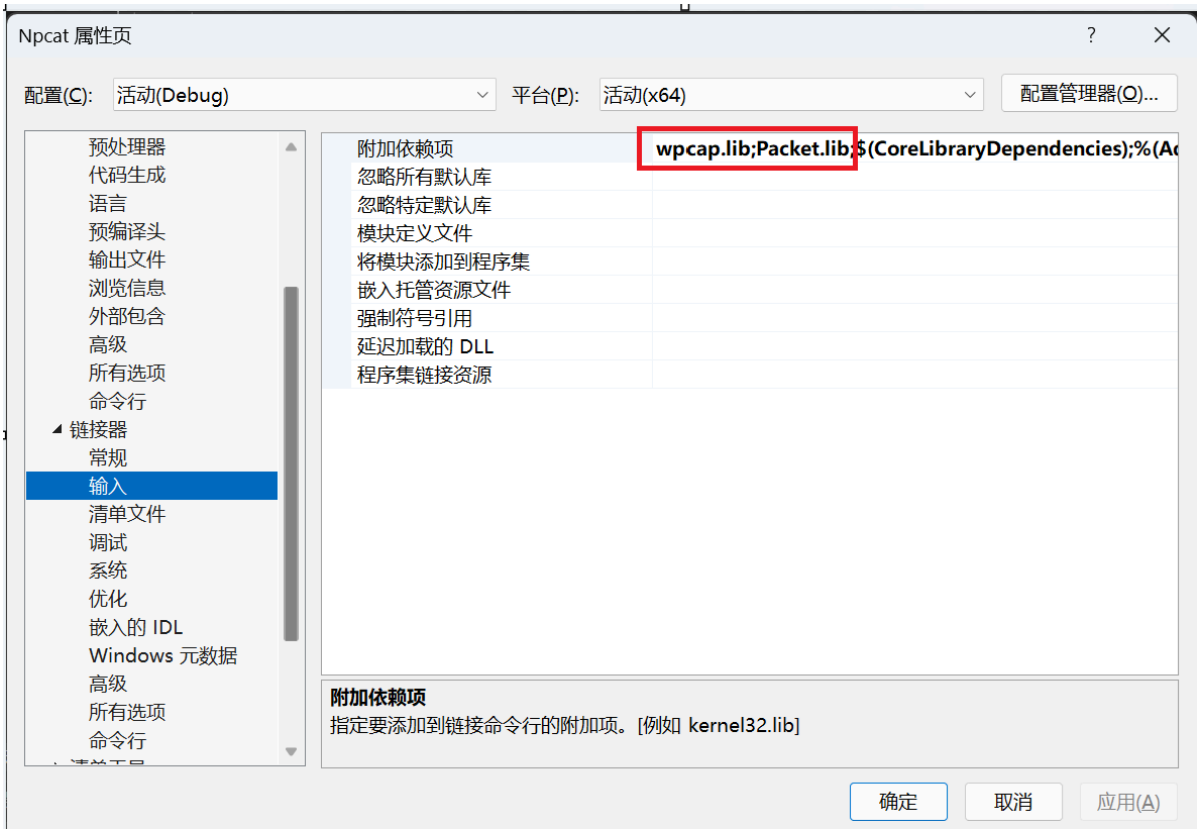



## (4) 添加库文件目录

项目 - 属性 - 配置属性 - 连接器 - 常规 - 附加库目录

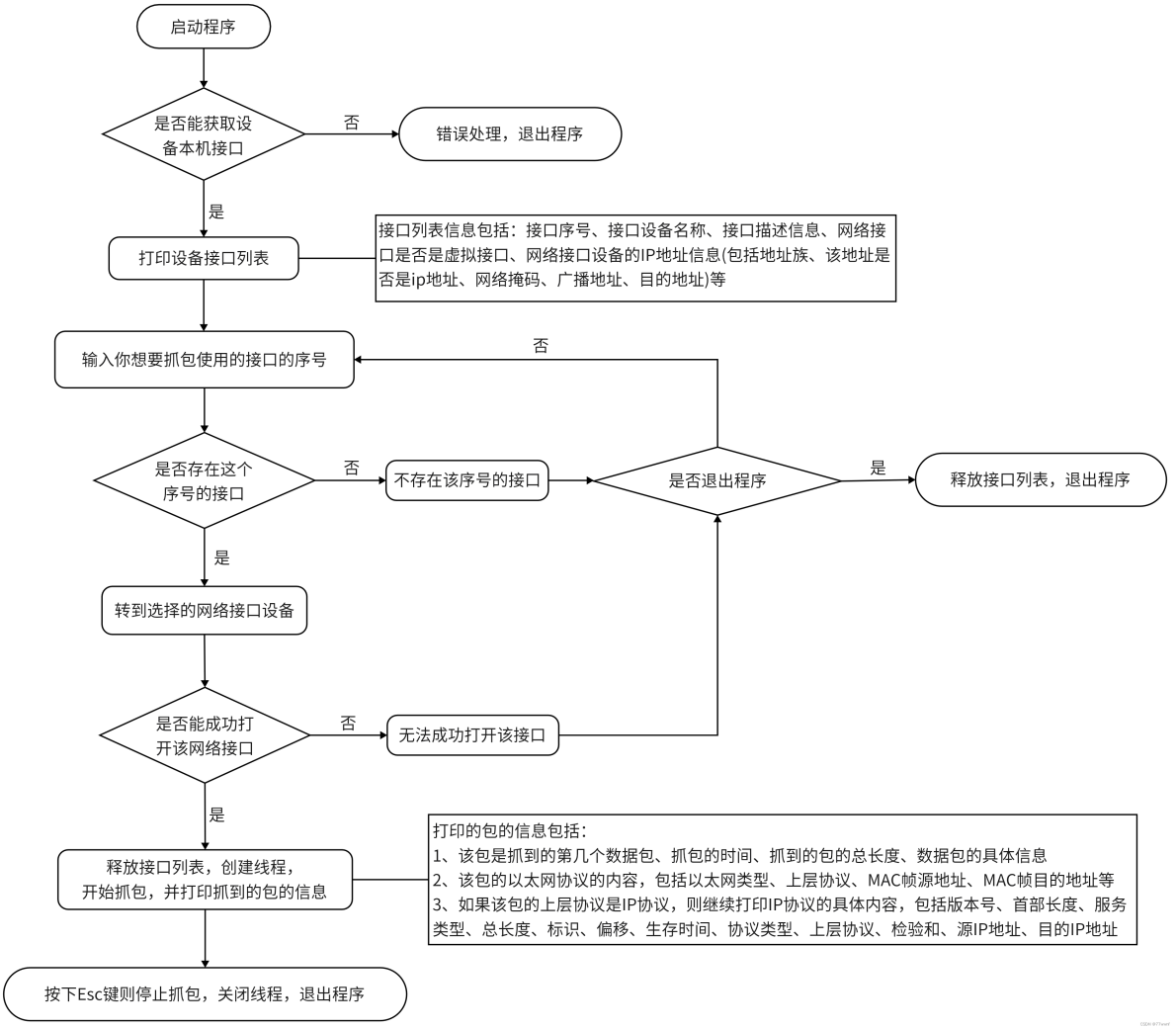



## (5) 添加链接时使用的库文件



# 二、实验过程

## （1）实验流程图




## （2）程序代码

```
#include<iostream>
#include<winsock2.h>
#include <conio.h>
#include "pcap.h"
#include "remote-ext.h"
#pragma comment(lib,"wpcap.lib")
#pragma comment(lib,"packet.lib")
#pragma comment(lib,"ws2_32.lib")
#pragma warning(disable:4996)
#define LINE_LEN 16
#define IPTOSBUFFERS 12
using namespace std;
int i = 0;

/*帧首部*/
struct FrameHeader_t {
    byte DesMAC[6]; //目的Mac地址
    byte SrcMAC[6]; //源Mac地址
    WORD FrameType; //协议（帧）类型
};
```

```
21
22  /*IP首部*/
23  struct IPHeader_t {
24  #if defined(WORDS_BIENDIAN)
25      byte Version : 4, HeaderLength : 4;
26  #else
27      byte HeaderLength : 4, Version : 4;
28  #endif
29      byte TOS;//服务类型
30      WORD TotalLen; //总长度
31      WORD ID;//标识
32      WORD Flag_Segment;//标志字段
33      byte TTL;//生存时间值
34      byte Protocol;//协议类型，是TCP/IP体系中的网络层协议
35      WORD Checksum;//检验和
36      struct in_addr SrcIP;//源IP
37      struct in_addr DstIP;//目的IP
38  };
39
40  /*IP协议首部处理函数*/
41  void ip_protocol_packet_callback(const struct pcap_pkthdr* packet_header,
    const u_char* packet_data)
42  {
43      struct IPHeader_t* IPHeader;
44      IPHeader = (struct IPHeader_t*)(packet_data + 14);//MAC首部是14位的，加上
    14位得到IP协议首部
45      printf("---------IP协议-----------------------------------------------
    ---------\n");
46      printf("版本号:%d\n", IPHeader->Version);
47      printf("首部长度:%d\n", IPHeader->HeaderLength);
48      printf("服务类型:%d\n", IPHeader->TOS);
49      printf("总长度:%d\n", ntohs(IPHeader->TotalLen));
50      printf("标识:%d\n", ntohs(IPHeader->ID));
51      printf("偏移:%d\n", (ntohs(IPHeader->Flag_Segment) & 0x1fff) * 8);
52      printf("生存时间:%d\n", IPHeader->TTL);
53      printf("协议类型:%d\n", IPHeader->Protocol);
54      switch (IPHeader->Protocol)
55      {
56      case 1: printf("上层协议:ICMP协议\n"); break;
57      case 2: printf("上层协议:IGMP协议\n"); break;
58      case 6: printf("上层协议:TCP协议\n"); break;
59      case 17: printf("上层协议:UDP协议\n"); break;
60      default:printf("上层协议:Unknown\n"); break;
61      }
62      printf("检验和:%d\n", ntohs(IPHeader->Checksum));
63      printf("源IP地址:%s\n", inet_ntoa(IPHeader->SrcIP));
64      printf("目的地址:%s\n", inet_ntoa(IPHeader->DstIP));
65  }
66
67  /*帧首部处理函数*/
68  void ethernet_protocol_packet_callback(const struct pcap_pkthdr*
    packet_header, const u_char* packet_data)
69  {
70      struct FrameHeader_t* frameHeader;
71      frameHeader = (struct FrameHeader_t*)packet_data;//获得数据包内容
```

```
    printf("---------以太网协议-----------------------------------------------
--------\n");
    printf("以太网类型:0x%04x\n", ntohs(frameHeader->FrameType));
    switch (ntohs(frameHeader->FrameType))
    {
    case 0x0800: printf("上层协议:IP协议\n"); break;
    case 0x0806: printf("上层协议:ARP协议\n"); break;
    case 0x8035: printf("上层协议:RARP协议\n"); break;
    default:printf("上层协议:Unknown\n"); break;
    }

    u_char* macS = frameHeader->SrcMAC;
    u_char* macD = frameHeader->DesMAC;
    printf("MAC帧源地址:%02x:%02x:%02x:%02x:%02x:%02x\n", *macS, *(macS +
1), *(macS + 2), *(macS + 3), *(macS + 4), *(macS + 5));
    printf("MAC帧目的地址:%02x:%02x:%02x:%02x:%02x:%02x\n", *macD, *(macD +
1), *(macD + 2), *(macD + 3), *(macD + 4), *(macD + 5));

    if (ntohs(frameHeader->FrameType) == 0x0800)//继续分析IP协议
    {
        ip_protocol_packet_callback(packet_header, packet_data);
    }
}

/*线程参数结构体*/
struct parame
{
    pcap_t* adhandle;
    struct pcap_pkthdr* packet_header;
    const u_char* packet_data;
    int num;
};

/*抓包线程*/
DWORD WINAPI Capturer(PVOID hWnd)
{
    int res;
    int packet_number = 0;
    struct tm* ltime;
    time_t local_tv_sec;
    char timestr[16];

    //将传入线程中的参数携带的数据取出
    parame* Packet = (parame*)hWnd;
    pcap_t* adhandle = Packet->adhandle;
    struct pcap_pkthdr* packet_header = Packet->packet_header;
    const u_char* packet_data = Packet->packet_data;
    int num = Packet->num;

    while ((res = pcap_next_ex(adhandle, &packet_header, &packet_data)) >=
0)
    {
        //接收数据包超时
        if (res == 0)
        {
            continue;
```

```
        }


printf("================================================================================\n");
        packet_number++;

        //将时间戳转化为可识别格式
        local_tv_sec = packet_header->ts.tv_sec;
        ltime = localtime(&local_tv_sec);
        strftime(timestr, sizeof(timestr), "%H:%M:%S", ltime);

        printf("捕获第%d个网络数据包\n", packet_number);;
        printf("捕获时间:%s\n", timestr);
        printf("数据包长度:%d\n", packet_header->len);
        printf("---------数据包内容----------------------------------------------\n");

        //输出数据包的具体内容
        char temp[LINE_LEN + 1];
        for (int i = 0; i < packet_header->caplen; ++i)
        {
            printf("%.2x ", packet_data[i]);
            if (isgraph(packet_data[i]) || packet_data[i] == ' ')
                temp[i % LINE_LEN] = packet_data[i];
            else
                temp[i % LINE_LEN] = '.';

            if (i % LINE_LEN == 15)
            {
                temp[16] = '\0';
                printf("        ");
                printf("%s", temp);
                printf("\n");
                memset(temp, 0, LINE_LEN);
            }
        }
        printf("\n");

        //分析数据包
        ethernet_protocol_packet_callback(packet_header, packet_data);

printf("================================================================================\n");

        //停止数据包捕捉
        if (_kbhit())
        {
            int ch = _getch();//使用_getch()函数获取按下的键值
            if (ch == 27)//当按下ESC时退出
            {
                cout << "\nThe Capturer has been closed..." << endl;
                return 0;
            }
        }

```

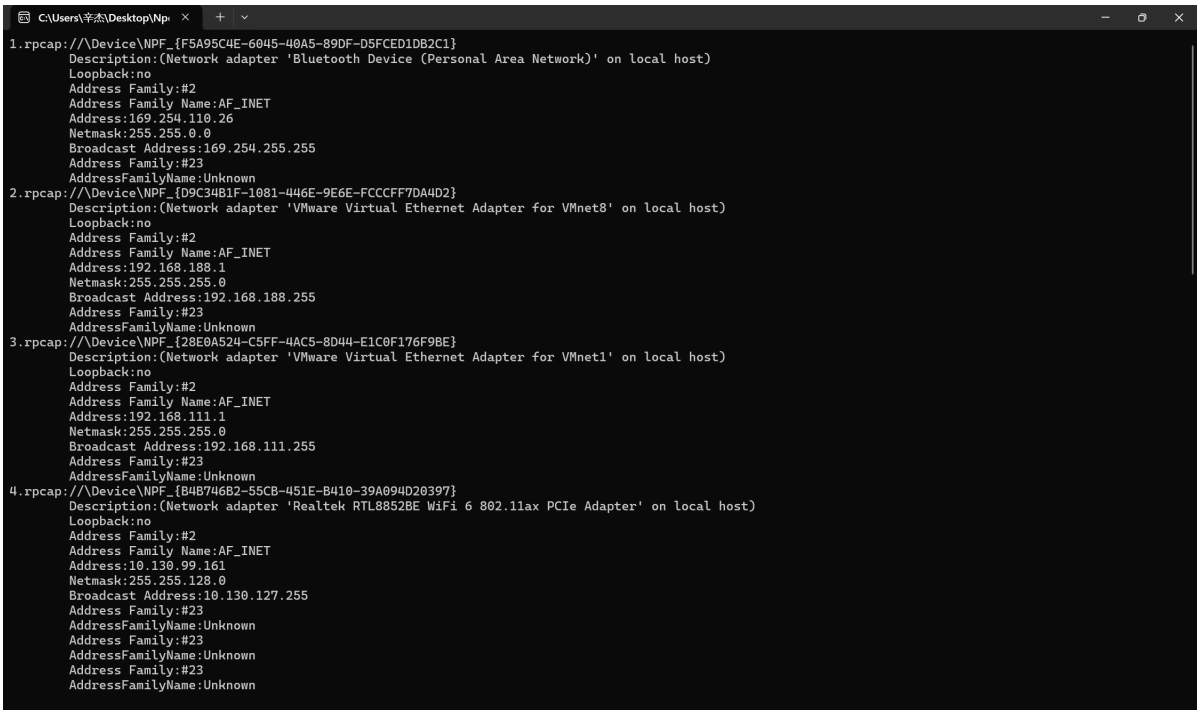
```
        //每1秒抓一次
        Sleep(1000);
    }
}

/*IP格式转化函数*/
char* iptos(u_long in)
{
    static char output[IPTOSBUFFERS][3 * 4 + 3 + 1];
    static short which = (which + 1 == IPTOSBUFFERS ? 0 : which + 1);
    u_char* p = (u_char*)&in;
    sprintf(output[which], "%d.%d.%d.%d", p[0], p[1], p[2], p[3]);
    return output[which];
}

/*打印本机接口列表*/
void ifprint(pcap_if_t* d)
{
    pcap_addr_t* a;//表示接口地址指针
    printf("%d.%s\n", ++i, d->name);//打印网络接口设备的名字
    if (d->description)
    {
        printf("\tDescription:(%s)\n", d->description);//打印描述信息
    }
    else {
        printf("\t(No description available)\n");
    }
    printf("\tLoopback:%s\n", (d->flags & PCAP_IF_LOOPBACK) ? "yes" :
"no");//是否是虚拟接口

    //获取该网络接口设备的IP地址信息
    for (a = d->addresses; a != NULL; a = a->next)
    {
        printf("\tAddress Family:#%d\n", a->addr->sa_family);
        switch (a->addr->sa_family)
        {
        case AF_INET://判断该地址是否IP地址
            printf("\tAddress Family Name:AF_INET\n");
            if (a->addr)//ip地址
            {
                printf("\tAddress:%s\n", iptos(((struct sockaddr_in*)a-
>addr)->sin_addr.s_addr));
            }
            if (a->netmask)//网络掩码
            {
                printf("\tNetmask:%s\n", iptos(((struct sockaddr_in*)a-
>netmask)->sin_addr.s_addr));
            }
            if (a->broadaddr)//广播地址
            {
                printf("\tBroadcast Address:%s\n", iptos(((struct
sockaddr_in*)a->broadaddr)->sin_addr.s_addr));
            }
            if (a->dstaddr)//目的地址
            {
```

```
                printf("\tDestination Address:%s\n", iptos(((struct
sockaddr_in*)a->dstaddr)->sin_addr.s_addr));
            }
            break;
        default:
            printf("\tAddressFamilyName:Unknown\n");
            break;
        }
    }
}

int main()
{
    pcap_if_t* alldevs;//指向设备链表首部的指针
    pcap_if_t* d;
    int num;
    pcap_t* adhandle;//定义打开设备的返回值
    char errbuf[PCAP_ERRBUF_SIZE];//错误信息缓冲区

    //取得列表
    if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING,//获取本机的接口设备
        NULL,//无需认证
        &alldevs,//指向设备列表首部
        errbuf//出错信息保存缓冲区
    ) == -1)
    {
        printf("Error in pcap_findalldevs_ex!%s\n");//错误处理
        return -1;
    }

    //输出列表
    for (d = alldevs; d != NULL; d = d->next)
    {
        ifprint(d);
    }
    if (i == 0)
    {
        printf("\nNo interfaces found!\n");
        return -1;
    }

LOOP:
    printf("\n输入接口号 (1-%d):", i);
    scanf("%d", &num);
    if (num <1 || num >i)
    {
        printf("\nInterface number out of range.\n");
        printf("Continue or Quit [c/q]:");
        char a;
        cin >> a;
        if (a == 'c' || a == 'C')
        {
            goto LOOP;
        }
        else if (a == 'q' || a == 'Q')
        {
```

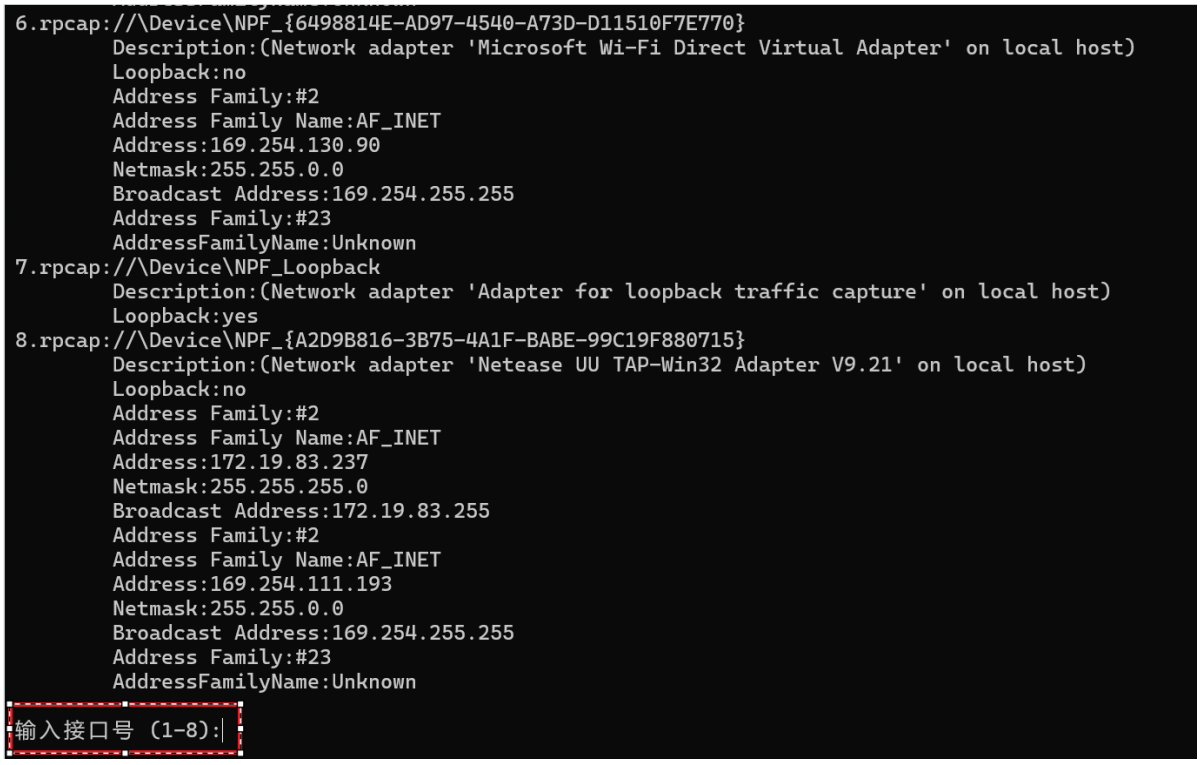
```
            pcap_freealldevs(alldevs);
            return 0;
        }
    }

    //转到选择的设备
    for (d = alldevs, i = 0; i < num - 1; d = d->next, i++);
    //打开失败
    if ((adhandle = pcap_open_live(d->name, 65536, 1, 1000, errbuf)) ==
NULL)
    {
        fprintf(stderr, "\nUnable to open the adapter.\n");
        printf("Continue or Quit [c/q]:");
        char a;
        cin >> a;
        if (a == 'c' || a == 'C')
        {
            goto LOOP;
        }
        else if (a == 'q' || a == 'Q')
        {
            pcap_freealldevs(alldevs);
            return 0;
        }
    }
    //打开成功
    printf("\nlistening on %s...\n\n", d->description);
    //释放列表
    pcap_freealldevs(alldevs);

    //开始抓包，创建线程
    HANDLE m_capturer;
    struct pcap_pkthdr* packet_header = new pcap_pkthdr;
    const u_char* packet_data = new u_char;
    parame* m_pam = new parame;
    m_pam->adhandle = adhandle;
    m_pam->packet_header = packet_header;
    m_pam->packet_data = packet_data;
    m_pam->num = num;
    m_capturer = CreateThread(NULL, NULL, &Capturer, (PVOID*)m_pam, 0,
NULL);
    CloseHandle(m_capturer);
    while (1);
    return 0;
}
```

## （3）程序展示

### 1）打印设备接口列表




## 2）输入想要抓包的端口号




## 3）开始抓包

> 以接口4（wife接口）为例

```
输入接口号 (1-8):4

listening on Network adapter 'Realtek RTL8852BE WiFi 6 802.11ax PCIe Adapter' on local host...

================================================================================
捕获第1个网络数据包
捕获时间:20:07:27
数据包长度:168
---------数据包内容------------------------------------------------------------
01 80 c2 00 00 0e b4 8c 9d df 0f 58 88 cc 02 07         ...........X....
07 78 69 6e 6a 69 65 04 09 07 70 6f 72 74 2d 30         .xinjie...port-0
30 31 06 02 00 14 0a 06 58 49 4e 4a 49 45 0c 44         01......XINJIE.D
48 50 20 48 50 20 50 61 76 69 6c 69 6f 6e 20 50         HP HP Pavilion P
6c 75 73 20 4c 61 70 74 6f 70 20 31 34 2d 65 68         lus Laptop 14-eh
30 78 78 78 2c 54 79 70 65 31 50 72 6f 64 75 63         0xxx,Type1Produc
74 43 6f 6e 66 69 67 49 64 2c 38 43 47 32 32 30         tConfigId,8CG220
30 59 52 4d 0e 04 00 80 00 80 10 14 05 01 00 00         0YRM............
00 00 02 00 00 00 01 08 2b 06 01 04 01 81 c0 6e         ........+......n
fe 08 00 0e cf 02 00 00 00 00 fe 0a 00 0e cf 05         ................
b4 8c 9d df 0f 58 00 00
---------以太网协议------------------------------------------------------------
以太网类型:0x88cc
上层协议:Unknown
MAC帧源地址:b4:8c:9d:df:0f:58
MAC帧目的地址:01:80:c2:00:00:0e
================================================================================
================================================================================
捕获第2个网络数据包
捕获时间:20:07:28
数据包长度:86
---------数据包内容------------------------------------------------------------
33 33 ff cc 30 c1 84 5b 12 5e 36 02 86 dd 6c 00         33..0..[.^6...l.
00 00 00 20 3a ff fe 80 00 00 00 00 00 00 86 5b         ... :..........[
12 ff fe 5e 36 02 ff 02 00 00 00 00 00 00 00 00         ...^6...........
00 01 ff cc 30 c1 87 00 f5 7f 00 00 00 00 20 01             0.
```

```
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
---------以太网协议-------------------------------------------------------
以太网类型:0x8918
上层协议:Unknown
MAC帧源地址:74:1f:4a:7e:77:1a
MAC帧目的地址:01:0f:e2:00:00:07
================================================================================
================================================================================
捕获第4个网络数据包
捕获时间:20:07:29
数据包长度:86
---------数据包内容-------------------------------------------------------
33 33 ff 4a 3a 07 84 5b 12 5e 36 02 86 dd 6c 00          33.J:..[.^6...l.
00 00 00 20 3a ff fe 80 00 00 00 00 00 00 86 5b          ... :..........[
12 ff fe 5e 36 02 ff 02 00 00 00 00 00 00 00 00          ...^6...........
00 01 ff 4a 3a 07 87 00 46 d9 00 00 00 00 20 01          ...J:...F..... .
02 50 04 01 65 61 d9 84 49 f2 76 4a 3a 07 01 01          .P..ea..I.vJ:...
84 5b 12 5e 36 02
---------以太网协议-------------------------------------------------------
以太网类型:0x86dd
上层协议:Unknown
MAC帧源地址:84:5b:12:5e:36:02
MAC帧目的地址:33:33:ff:4a:3a:07
================================================================================
================================================================================
捕获第5个网络数据包
捕获时间:20:07:29
数据包长度:86
---------数据包内容-------------------------------------------------------
33 33 ff d8 fb 60 84 5b 12 5e 36 02 86 dd 6c 00          33...`.[.^6...l.
00 00 00 20 3a ff fe 80 00 00 00 00 00 00 86 5b          ... :..........[
12 ff fe 5e 36 02 ff 02 00 00 00 00 00 00 00 00          ...^6...........
00 01 ff d8 fb 60 87 00 56 c8 00 00 00 00 20 01          .....`..V..... .
02 50 04 01 65 61 20 db 95 dd 4f d8 fb 60 01 01          .P..ea ...O..`..
84 5b 12 5e 36 02
---------以太网协议-------------------------------------------------------
以太网类型:0x86dd
上层协议:Unknown
MAC帧源地址:84:5b:12:5e:36:02
MAC帧目的地址:33:33:ff:d8:fb:60
================================================================================
================================================================================
捕获第6个网络数据包
```

成功抓包

# 三、实验感想

通过这次数据包捕获与分析的实验，我深入了解了NPcap的架构和使用方法。编程实现数据包捕获并分析源MAC、目的MAC和类型/长度字段的过程，锻炼了我的编程能力和对网络协议的理解。