网络技术与应用课程报告

实验二：数据包捕获与分析

姓名：孙悦

学号：2110052

专业：物联网工程

一、实验内容

数据包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，实现本机的数据包捕获，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、实验准备

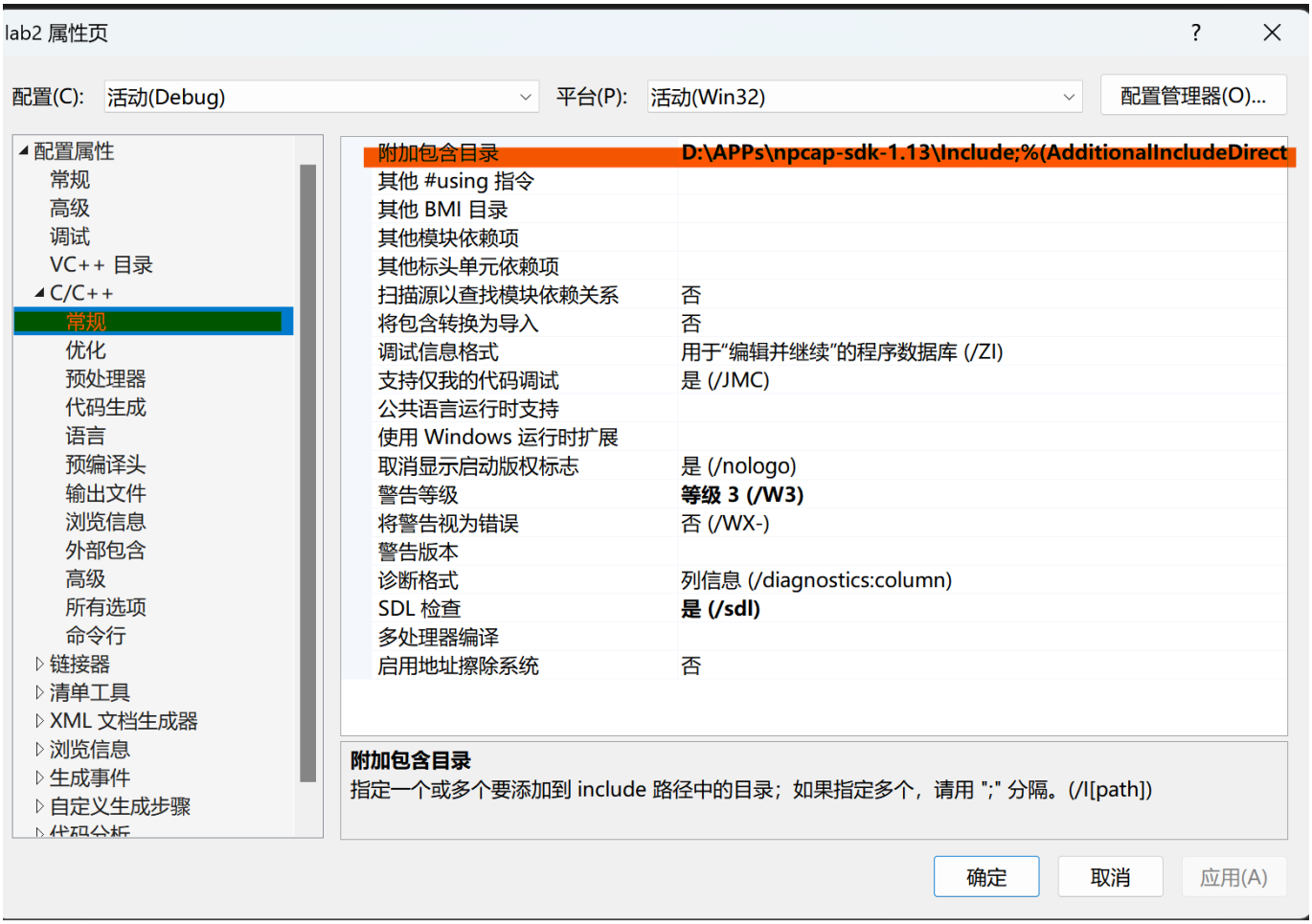
1.npcap与npcap-sdk下载

WinPcap 是一个数据包捕获体系框架，主要功能是进行数据包捕获和网络分析。包括了内核基本的包过滤、低层次的库(packet.lib)、高级别系统无关的函数库(wpcap.dll)。WinPcap已 停止维护，采用 NPcap 进行实验。

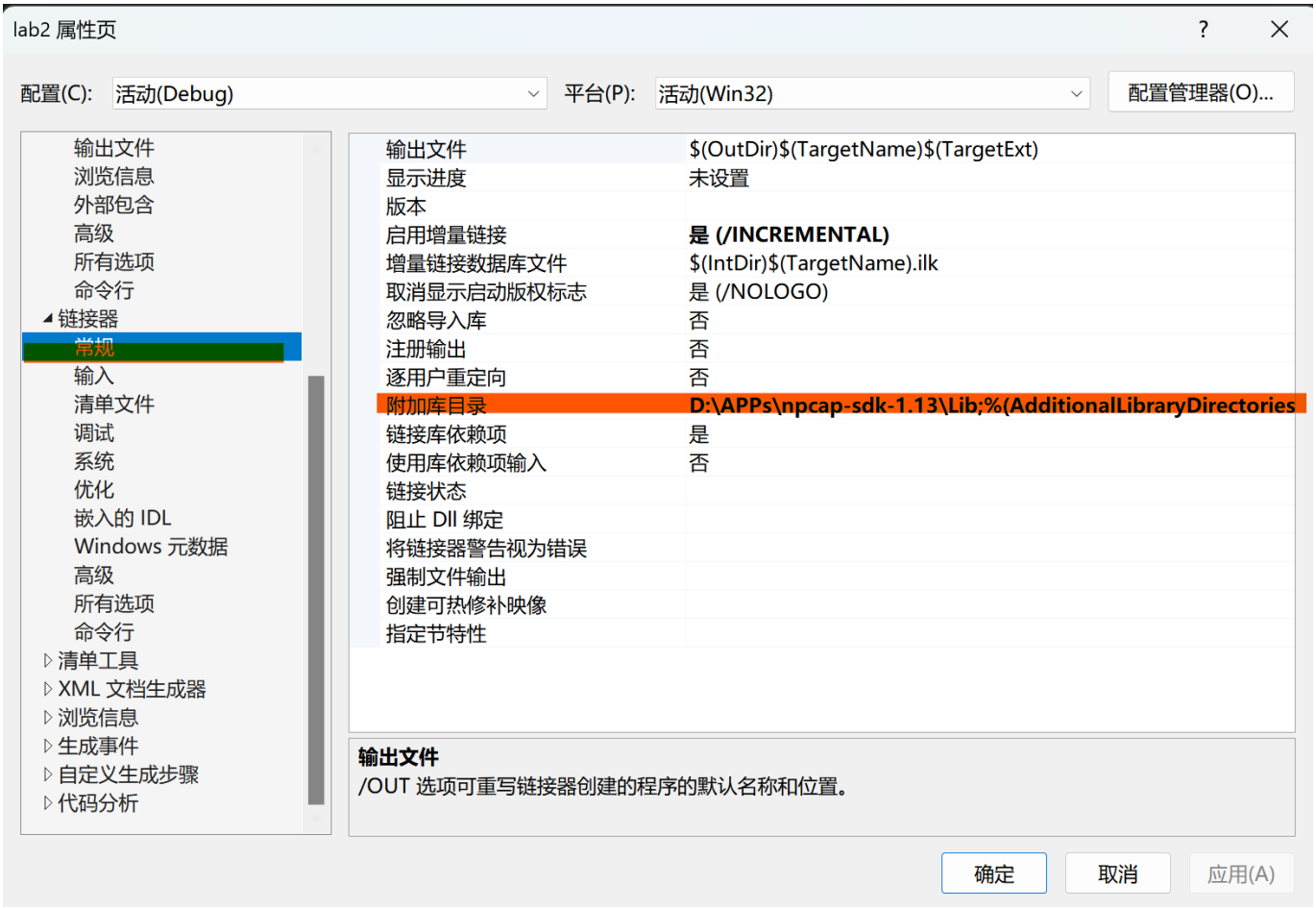
2.VS2019配置环境

在项目属性页中进行如下操作：

- C/C++-常规-附加包含目录添加npcap-sdk目录下的**Include**文件夹

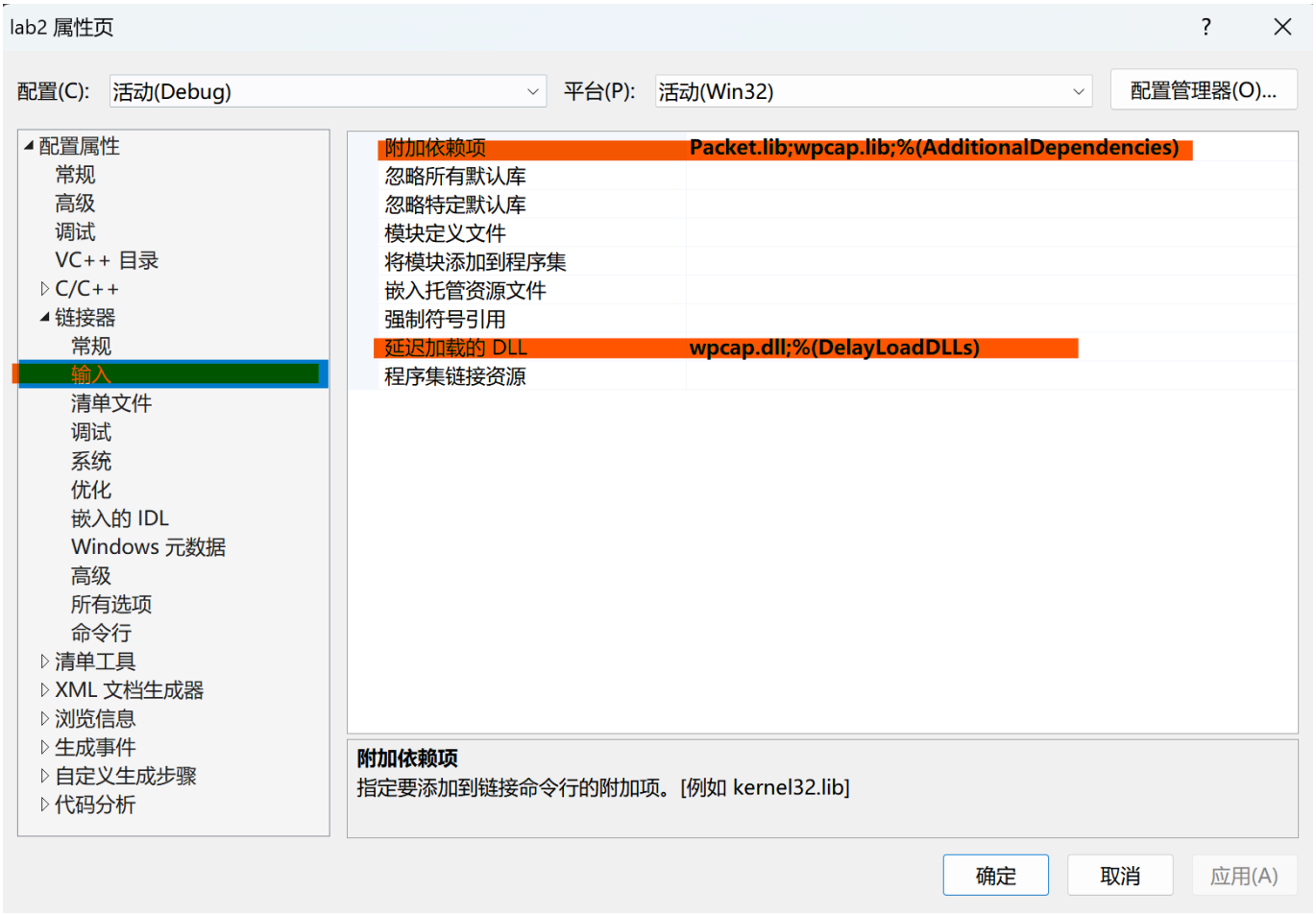


- 链接器-常规-附加库目录 添加npcap-sdk目录下的Lib文件夹



- 链接器-输入-附加依赖项 输入添加Packet.lib和wpcap.lib

  链接器-输入-延迟加载的DLL 输入添加wpcap.dll



三、实验过程

1.程序设计流程为：

开始→初始化（利用`pcap_findalldevs_ex`获得本机网卡列表）→调用Npcap接口`pcap_open()`打开网卡接口，调用`pcap_next_ex()`主动捕获数据接收数据包→解析数据包，`ethernet_protocal_packet_callback()`解析以太网数据包，`ip_protocal_packet_callbact()`解析ip数据包→接着捕获和分析数据包——释放。

- **定义报文和ip地址格式**

```
//报文
struct ethernet_header {//帧首部
    uint8_t mac_dst[6];//目的MAC地址
    uint8_t mac_src[6];//源MAC地址
    uint16_t frame_type;//帧类型
};

//IP地址
typedef uint32_t in_addr_t;
struct ip_header {//ip首部
    uint8_t ip_header_length : 4,//首部长度
        ip_version : 4;//版本

    uint8_t ip_tos;//服务类型
    uint16_t total_len;//总长度
    uint16_t ip_id;//标识
    uint16_t ip_off;//片偏移
    uint8_t ip_ttl;//生存时间
    uint8_t ip_protocol;//协议类型
    uint16_t ip_checksum;//首部检验和
    struct in_addr ip_source_address;//源IP地址
    struct in_addr ip_destination_address;//目的IP地址
};
```

2.程序运行主要流程为：

`main→user_select_device→pthread_create→watch_ippkt→print_hdr`

`user_select_device` 函数用于打印所有设备信息，并获得所选设备的名称。

`watch_ippkt` 用于打开并监听设备。 `print_hdr` 用于解析和输出报文。

- 获得网卡列表

pcap_if_t* alldevs;指向网卡列表的指针，从中获得网卡名称和描述

```
//获取网卡
    if (pcap_findalldevs(&alldevs, errbuf) == -1) {
        cout << stderr << "获取网卡失败:%s\n" << errbuf;
        exit(1);
    }
    //打印网卡信息
    for (d = alldevs; d; d = d->next) {
        cout << ++i << d->name;
        if (d->description)
            cout << d->description;
        else
            cout << "No description available\n";
        cout << endl;
    }
```

- 捕获数据包

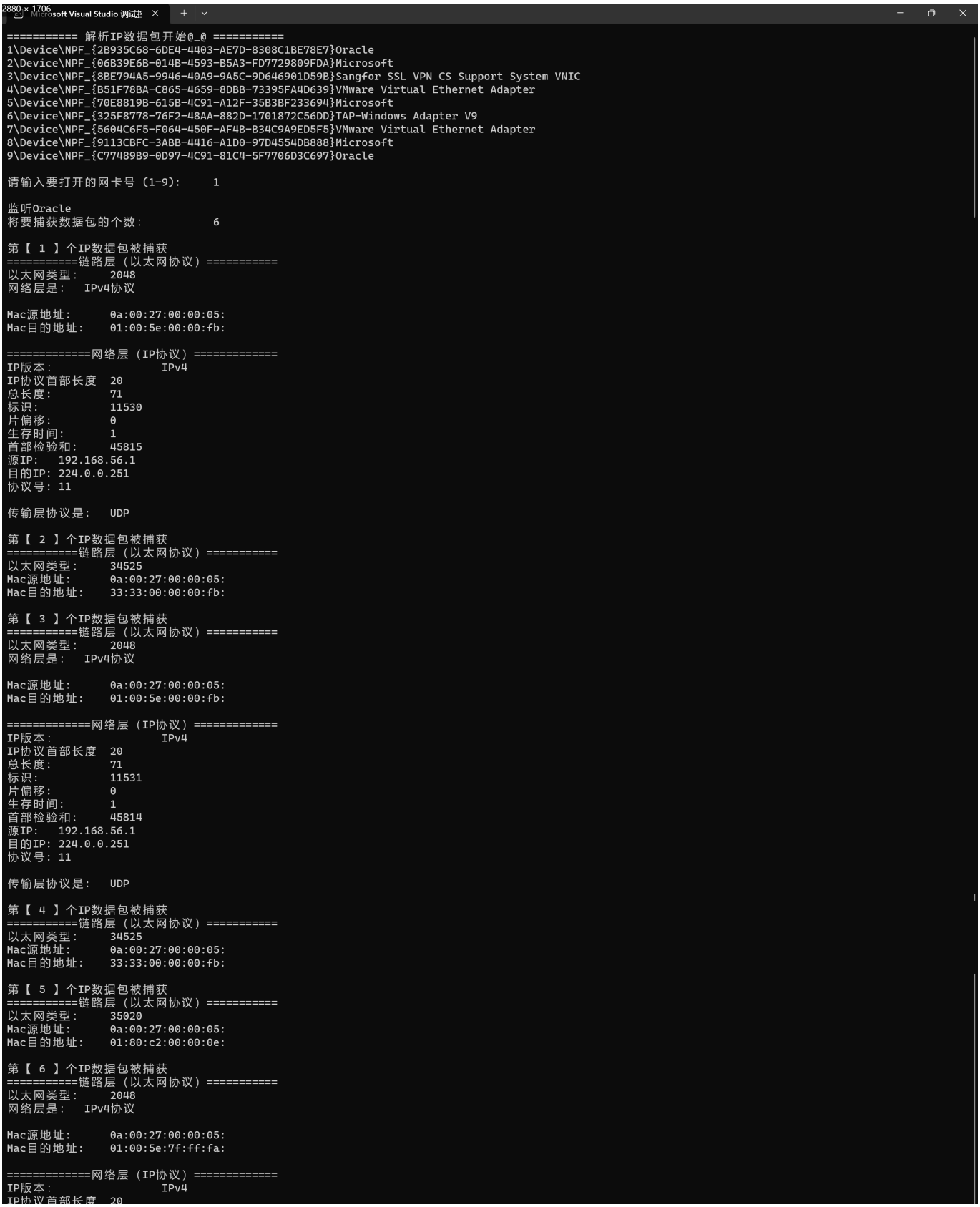
```
void ethernet_protocol_packet_callback(u_char* argument, const struct pcap_pkthdr*
packet_header, const u_char* packet_content) {
    u_short ethernet_type;//以太网协议类型
    struct ethernet_header* ethernet_protocol;//以太网协议变量
    uint8_t* mac_src;//Mac源地址
    uint8_t* mac_dst;//Mac目的地址
    static int packet_number = 1;//抓包数量
    cout << endl;
    printf("第【 %d 】个IP数据包被捕获\n", packet_number);
    cout << "============链路层（以太网协议）============" << endl;
    ethernet_protocol = (struct ethernet_header*)packet_content;//获得以太网协议数据内
容//指针指向捕获的数据包内容的起始位置，以便访问以太网首部
    cout << "以太网类型：\t";
    ethernet_type = ntohs(ethernet_protocol->frame_type);//获得以太网类型//ntohs 函数将
网络字节序转换为主机字节序。
    cout << ethernet_type << endl;
    switch (ethernet_type) {//判断以太网类型
    case 0x0800:
        cout << "网络层是:   IPv4协议\n" << endl;
        break;
    case 0x0806:
        cout << "网络层是:   ARP协议\n" << endl;
        break;
    case 0x0835:
        cout << "网络层是:   RARP协议\n" << endl;
        break;
    default:break;
    }

    //Mac源地址
    mac_src = ethernet_protocol->mac_src;
    printf("Mac源地址:\t%02x:%02x:%02x:%02x:%02x:%02x:\n", *mac_src, *(mac_src + 1), *
(mac_src + 2), *(mac_src + 3), *(mac_src + 4), *(mac_src + 5));//X 表示以十六进制形式输
出 02 表示不足两位，前面补0输出
    //Mac目的地址
    mac_dst = ethernet_protocol->mac_dst;
    printf("Mac目的地址:\t%02x:%02x:%02x:%02x:%02x:%02x:\n", *mac_dst, *(mac_dst + 1),
*(mac_dst + 2), *(mac_dst + 3), *(mac_dst + 4), *(mac_dst + 5));

    switch (ethernet_type) {
    case 0x0800://如果上层是IPv4协议,就调用分析ip协议的函数对IP数据包进行进一步的解析
        ip_protocol_packet_callback(argument, packet_header, packet_content);
        break;
    default:break;
    }
    packet_number++;
}
```

- 报文解析

```
void ip_protocol_packet_callback(u_char* argument, const struct pcap_pkthdr*
packet_header, const u_char* packet_content) {
    struct ip_header* ip_protocol;//IP协议变量
    u_int header_length;//长度
    u_int offset;//片偏移
    u_char tos;//服务类型
    uint16_t checksum;//首部检验和
    ip_protocol = (struct ip_header*)(packet_content + 14);//获得IP数据包的内容 去掉以太
头//因为前14字节通常是以太网首部，所以需要跳过以太网首部部分，以获取到IPv4首部。
    checksum = ntohs(ip_protocol->ip_checksum);//获得检验和
    header_length = ip_protocol->ip_header_length * 4;//获得长度//IPv4首部长度以32位字为
单位，所以需要将其乘以4以获取实际字节数。
    tos = ip_protocol->ip_tos;//获得tos服务类型
    offset = ntohs(ip_protocol->ip_off);//获得偏移量
    cout << "\n==============网络层（IP协议）==============\n";
    printf("IP版本：\t\tIPv%01X\n", ip_protocol->ip_version);//
    cout << "IP协议首部长度\t" << header_length << endl;
    cout << "总长度:\t\t" << ntohs(ip_protocol->total_len) << endl;//获得总长度
    cout << "标识:\t\t" << ntohs(ip_protocol->ip_id) << endl;//获得标识
    cout << "片偏移:\t\t" << (offset & 0x1fff) * 8 << endl;
    printf("生存时间:\t%01X\n", ip_protocol->ip_ttl);//获得ttl
    cout << "首部检验和:\t" << checksum << endl;
    cout << "源IP:\t" << inet_ntoa(ip_protocol->ip_source_address) << endl;//获得源ip地
址
    cout << "目的IP:\t" << inet_ntoa(ip_protocol->ip_destination_address) << endl;//获
得目的ip地址
    printf("协议号:\t%01X\n", ip_protocol->ip_protocol);//获得协议类型
    cout << "\n传输层协议是:\t";
    switch (ip_protocol->ip_protocol) {
    case 1:
        cout << "ICMP" << endl;
        break;
    case 2:
        cout << "IGMP" << endl;
        break;
    case 3:
        cout << "GGP" << endl;
        break;
    case 6:
        cout << "TCP" << endl;
        break;
    case 8:
        cout << "EGP" << endl;
        break;
    case 17:
        cout << "UDP" << endl;
        break;
    case 89:
        cout << "OSPF" << endl;
        break;
```

```
        default:break;
        }
    }
```

3.程序运行结果



```
总长度:         203
标识:           36133
片偏移:         0
生存时间:       1
首部检验和:     17241
源IP:   192.168.56.1
目的IP: 239.255.255.250
协议号: 11

传输层协议是:   UDP

        解析IP数据包结束!

C:\Users\MNH\Desktop\大三\mycode\Computer_Architectures\labs\lab2\Debug\lab2.exe (进程 23852)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```

实验完成!