南开大学

Nankai University

南　开　大　学

网络空间安全学院

网络技术与应用课程报告

编程实验（1）IP 数据报捕获与分析

学号：2011897

姓名：任薏霖

年级：2020 级

专业：物联网工程

2022 年 10 月 27 日

第 1 节 实验内容说明

1. 实验题目

IP 数据包捕获与分析

2. 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。

第 2 节 实验准备

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)。目前，WinPcap 停止维护，故采用 NPcap 进行实验。运行相关程序，需要安装驱动程序。

在 Visual Studio2019 中配置环境过程如下：

1. 下载并安装 WinPcap、Npcap 运行库
2. 下载 WinPcap、Npcap 开发包
3. 创建并设置项目：

(1) 打开项目属性，添加 WPCAP 和 HAVE_REMOTE 两个宏定义。

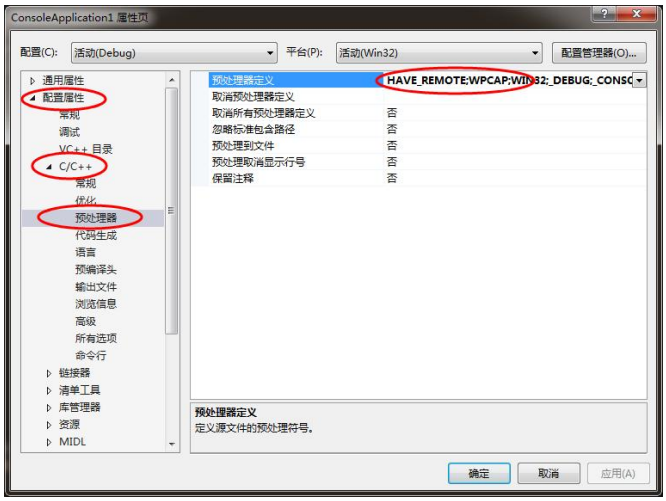

图 1 添加宏定义

(2) 添加 wpcap.lib 和 ws2_32.lib 两个库。

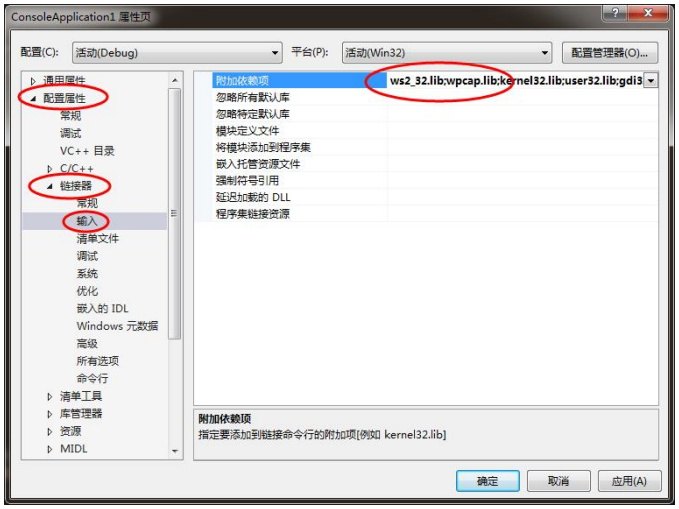

图 2 添加库

(3) 添加包含路径和库路径：

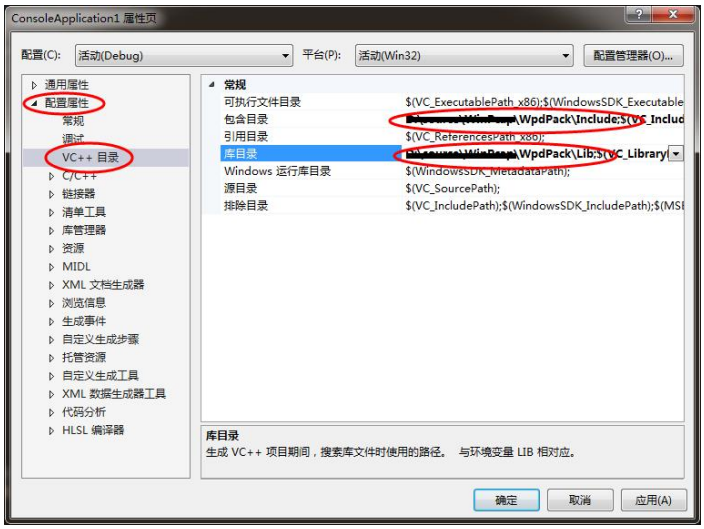

图 3 添加包含路径和库路径

第 3 节 实验过程

1. 程序设计

为了利用 Npcap 捕获 IP 报文，需要能够获得本机网卡设备，能够捕获设备上的数据包并解析报文。调用 Npcap 接口 pcap_findalldevs_ex 可以获得网卡列表，调用 Npcap 接口 pcap_open()从而打开网卡，调用 pcap_next_ex()从而主动捕获数据。

其流程图如下：

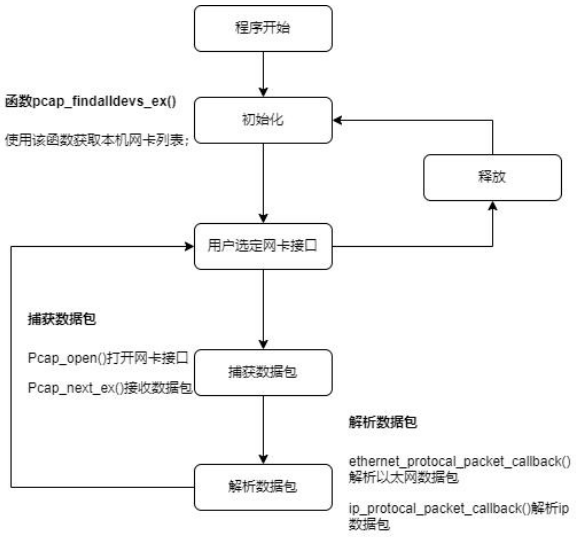

图 4 流程图

(1) 报文格式定义：

```
/*报文格式定义 */
struct ethernet_header
{
    uint8_t mac_dhost[6];      /*目的MAC地址*/
    uint8_t mac_shost[6];      /*源MAC地址*/
    uint16_t frame_type;         /*帧类型*/
};
```

(2) IP 地址格式定义：

```
typedef uint32_t in_addr_t;

struct ip_header{
#ifdef WORKS_BIGENDIAN
    uint8_t ip_version : 4,                    /*version:4*/
```

```
        ip_header_length : 4;                       /*IP协议首部长度Header
Length*/
#else
    uint8_t ip_header_length : 4,
        ip_version : 4;
#endif
    uint8_t ip_tos;                                 /*服务类型Differentiated
Services  Field*/
    uint16_t total_len;                             /*总长度Total Length*/
    uint16_t ip_id;                                 /*标识identification*/
    uint16_t ip_off;                                /*片偏移*/
    uint8_t ip_ttl;                                 /*生存时间Time To Live*/
    uint8_t ip_protocol;                            /*协议类型（TCP或者UDP协议）*/
    uint16_t ip_checksum;                           /*首部检验和*/
    struct in_addr  ip_source_address;              /*源IP*/
    struct in_addr  ip_destination_address;         /*目的IP*/
};
```

2. 关键代码分析

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。

(1) 获得网卡列表

```
//获得网卡的列表
    if (pcap_findalldevs(&alldevs, errbuf) == -1) {
        cout << stderr << "Error in pcap_findalldevs: %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;
    }
```

alldevs 是获得的指向网卡列表的指针，可以从中获得网卡名称和描述。

(2) 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*/
    u_char* mac_string;
    static int packet_number = 1;
    /*输出内容*/
    ethernet_protocol = (struct ethernet_header*)packet_content;
    /*获得一太网协议数据内容*/
    ethernet_type = ntohs(ethernet_protocol->frame_type);
    /*获得以太网类型*/
    switch (ethernet_type) {
    /*判断以太网类型的值*/
    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 0x8035:
        cout << "网络层是:       RARP 协议\n" << endl;
        break;
    default:
        break;
    }
    /*输出Mac源地址、目的地址*/
    switch (ethernet_type) {
    case 0x0800:
        ip_protocol_packet_callback(argument, packet_header, packet_content);
        break;
    default:break;
    }
    packet_number++;
}
```

(3) 解析报文

```
/*IP数据包分析的函数定义ethernet_protocol_packet_callback*/
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部*/
    checksum = ntohs(ip_protocol->ip_checksum);
    /*获得校验和*/
    header_length = ip_protocol->ip_header_length * 4;        /*获得长度*/
    tos = ip_protocol->ip_tos;                               /*获得tos*/
    offset = ntohs(ip_protocol->ip_off);                     /*获得偏移量*/
    /*输出解析的内容，略……*/
    switch (ip_protocol->ip_protocol) {
    case 6:
        cout << "TCP\n";
        break;
    case 17:
        cout << "UDP\n";
        break;
    case 1:
        cout << "ICMP\n";
        break;
    case 2:
        cout << "IGMP\n";
        break;
    default:break;
    }
}
```