南 开 大 学

计算机学院

网络技术与应用课程报告

第 4 次实验报告

学号：1911406
姓名：丁彦添
年级：2019 级
专业：计科

2021 年 11 月 9 日

第1节　实验内容说明

本次实验为 IP 数据报捕获与分析编程实验，具体要求如下：

（1）了解 WinPcap 的架构。

（2）学习 WinPcap 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）学习多线程程序编写方法。

（4）通过 WinPcap 编程，实现本机的 IP 数据报捕获。

（5）捕获的数据报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，源 IP、目的 IP 地址、校验和字段的数值。

（6）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

在使用 WinPcap 之前，首先需要安装 WinPcap 驱动程序和 DLL 程序。这些程序可以从官网上下载获得。本次实验使用的是 WinPcap 的升级版 NPcap。

NPcap 编程环境配置。在官网上下载四个文件，安装 NPcap 并将 NPcap 的头文件解压到对应的目录。

在 Visual Studio Code 2019 中需要添加相应的包含目录、库目录；链接器输入添加附加依赖项 ws2_32.lib、wpcap.lib。

第3节　实验过程

项目设计思路：

参考教材上的 WinPcap 编程各种函数如 pcap_next_ex，pcap_open，

pcap_findalldevs_ex 等函数的功能和实例化方法。

代码思路具体为：

先利用 pcap_findalldevs_ex 打开本机所有的端口，让用户输入选择监听哪一个端口，并捕获多少个 IP 数据包。

利用 pcap_open 函数打开网卡，利用 pcap_loop 函数来实现本机的数据包的捕获，通过调用回调函数 packet_handler 以简单明了的方式在命令行中显示源 MAC 地址、目的 MAC 地址，源 IP、目的 IP 地址、校验和字段的值。

关键代码分析：（完整代码在实验报告同一目录中）

```
//存储MAC地址的结构体
struct ethernet_header
{
    uint8_t ether_final[6];//目的MAC地址
    uint8_t ether_from[6];//源MAC地址
    uint16_t ether_type;
};
//存储IP地址和校验和的结构体
struct ip_header
{
    uint8_t ip_header_length : 4,
        ip_version : 4;

    uint8_t ip_tos;
    uint16_t ip_length;
    uint16_t ip_checksum;//校验和字段
    struct in_addr  ip_source_address;//源地址
    struct in_addr  ip_destination_address;//目的地址
};
/* packet handler 函数 */
/* 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;
    struct ethernet_header* ethernet_protocol; /*以太网协议变量*/
```

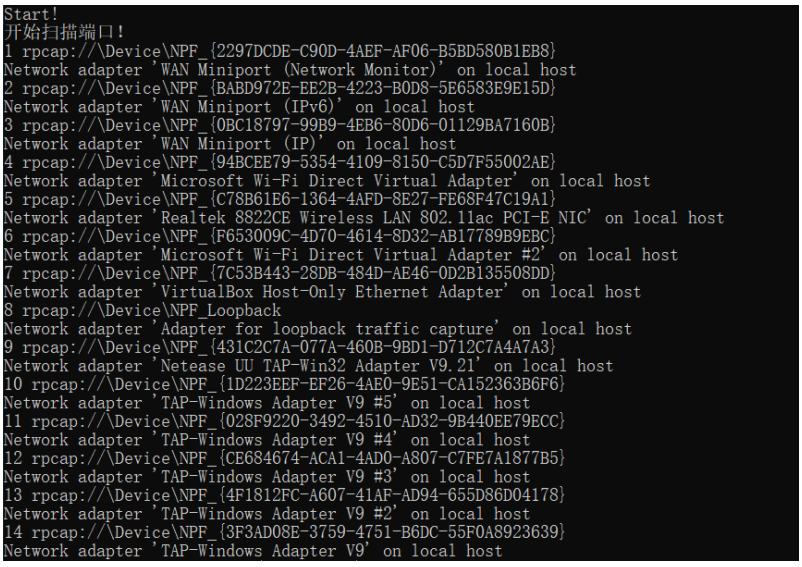
```
    struct ip_header* ip_protocol;/*ip协议变量*/
    ip_protocol = (struct ip_header*)(pkt_data + 14); /*处理ip数据包的内容*/
    char timestr[16];
    time_t local_tv_sec;
    u_char* macsave;
    cout << "捕获到数据包!" << endl;
    /* 将时间戳转换成可识别的格式 */
    local_tv_sec = header->ts.tv_sec;
    ltime = localtime(&local_tv_sec);
    cout << "捕获时间:    ";
    strftime(timestr, sizeof timestr, "%H:%M:%S", ltime);
    cout << timestr << endl;
    cout << "数据包长度:  " << header->len << "字节" << endl;
    ethernet_protocol = (struct ethernet_header*)pkt_data;
    /*源MAC地址*/
    macsave = ethernet_protocol->ether_from;
    cout << "源MAC地址:   ";
    printf("%02x:%02x:%02x:%02x:%02x:%02x", *macsave, *(macsave + 1), *(macsave
+ 2), *(macsave + 3), *(macsave + 4), *(macsave + 5));//经过测试，cout会产生奇怪
的bug，故用也能支持的printf来表示
    /*目的MAC地址*/
    cout << endl;
    macsave = ethernet_protocol->ether_final;
    cout << "目的MAC地址: ";
    printf("%02x:%02x:%02x:%02x:%02x:%02x", *macsave, *(macsave + 1), *(macsave
+ 2), *(macsave + 3), *(macsave + 4), *(macsave + 5));
    cout << endl;
    /*源ip地址*/
    cout << "源IP地址:    " << inet_ntoa(ip_protocol->ip_source_address) <<
endl;
    /*目的ip地址*/
    cout << "目的IP地址:  " << inet_ntoa(ip_protocol->ip_destination_address) <<
endl;
    /*校验和字段*/
    cout << "校验和字段:  " << ip_protocol -> ip_checksum << endl;
    cout << endl << endl;
}
    //main函数中扫描所有端口并展示
    if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING,
        NULL,
        &alldevs,
        errbuf
    ) == -1)
    {
```

```
        cout << "error!" << endl << "获取端口错误！";
        exit(1);
    }
    for (d = alldevs; d != NULL; d = d->next)
    {
        cout << ++i << " " << d->name << endl;
        if (d->description)
        {
            cout << d->description << endl;
        }
        else
        {
            cout << "No description available" << endl << "没有可用的描述！" <<
endl;
        }
    }
    if (i == 0)
    {
        cout << "Check Winpcap" << endl << "没有找到端口！请检查 NPcap！" <<
endl;
        return -1;
    }
    //main中通过调用pcap_loop来捕获pnum个数据包
    pcap_loop(adhandle, pnum, packet_handler, NULL);
```

程序测试过程：

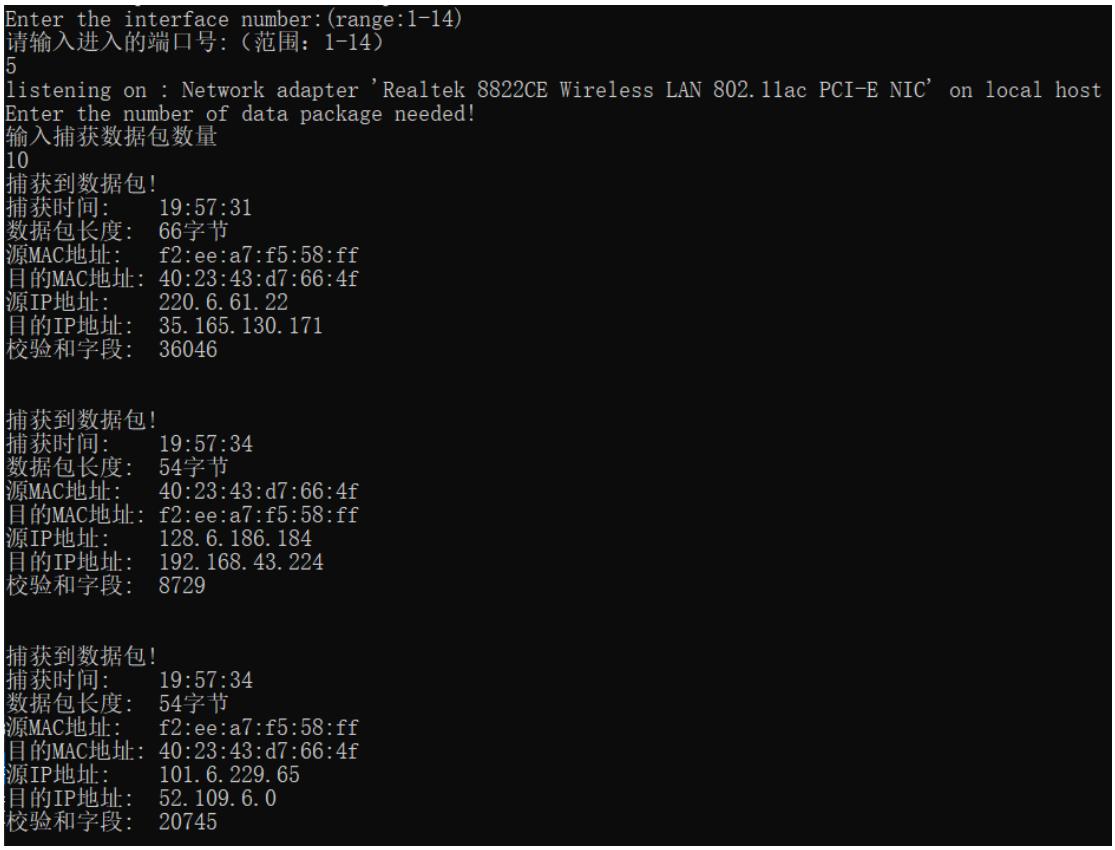
对最终代码生成解决方案并运行。

扫描到本机的所有网络端口如下：



可以看出一共有 14 个网络端口，选择第 5 个端口，即“Realtek 8822CE Wireless LAN 802.11ac PCI-E NIC”。

捕获 10 个 IP 数据包。



如图，能正确得到 IP 数据包的长度、源 MAC 地址、目的 MAC 地址、源 IP 地址、目的 IP 地址、校验和字段。

第4节　特殊现象分析

在使用 C++的 cout 语句输出 MAC 地址时：

```
cout << hex << setw(2) << *macsave << *(macsave + 1) <<*(macsave + 2) << *(macsave + 3) << *(macsave + 4) <<*(macsave + 5) << endl;
```

无法正常输出 mac 地址，乱码输出，而使用 printf 则不会，可以正常得到正确输出。