第二次实验报告

郭裕彬 2114052 物联网工程

前期准备

1. 了解NPcap的架构

   NPcap 是一个用于 Windows 操作系统的网络数据包捕获库，它提供了强大的网络数据包捕获功能，用于网络分析和监控应用程序的开发。NPcap 的架构可以大致分为三个部分：

   - NPF(Netgroup Packet Filter)，是一个虚拟设备驱动程序文件。它的功能是过滤数据包,并把这些数据包原封不动地传给用户态模块，这个过程中包括了一些操作系统特有的代码。
   - packet.dll为WIN32平台提供了一个公共的接口。不同版本的Windows系统都有自己的内核模块和用户层模块。Packet.dll用于解决这些不同。调用Packet.dll的程序可以运行在不同版本的Windows平台上，而无需重新编译。
   - Wpcap.dll，不依赖于操作系统的。它提供了更加高层、抽象的函数。

   NPcap具有如下的特点：

   - **Npcap Library**（Npcap 库）：NPcap 提供了一个编程接口，允许开发人员使用 C/C++ 和其他编程语言来控制数据包捕获操作。Npcap 库包括了用于数据包捕获的 API，如 `pcap_open`, `pcap_sendpacket` 等函数。
   - **Packet Filtering and Analysis**（数据包过滤与分析）：NPcap 允许用户定义数据包过滤规则，以便只捕获特定类型的数据包。这对于网络分析应用程序非常有用。
   - **Remote Packet Capture**（远程数据包捕获）：NPcap 支持远程数据包捕获。
   - **Loopback Packet Capture**（环回数据包捕获）：NPcap 具有支持捕获本地环回接口上的数据包的功能。
   - **User-Mode and Kernel-Mode**（用户模式和内核模式）：NPcap 可以在用户模式和内核模式之间切换，以提供不同级别的性能和特权。

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

   - 在网页Npcap Development Tutorial | Npcap Reference Guide上的Obtaining the device list部分学习设备列表获取方法：使用返回pacp_if结构体的pcap_findalldevs_ex()函数。
   - 在Opening an adapter and capturing the packets部分学习打开设备并捕获数据包的方法：使用pacp_open()打开设备，参数snaplen、flags和to_ms；使用自主编写的packet_handler(u_char *param,const struct pcap_pkthdr *header,const u_char *pkt_data)来对数据包进行捕获和解析，参数为数据包存储的文件指针、堆文件包的结构体首部指针以及指向数据包内容的指针；pkt_data包括了协议头，可以经过计算获得IP数据包头部的位置， UDP首部的位置。

程序编写

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。

- 结构体设计

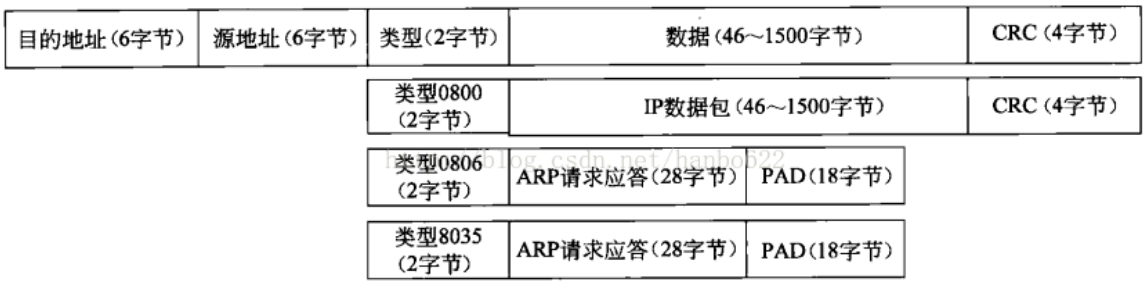
```
//ip地址结构体
typedef struct ip_address {
        u_char byte1;
        u_char byte2;
        u_char byte3;
        u_char byte4;
    }ip_address;

//以太帧头部结构体
typedef struct ehter_header {
    u_char    ether_dhost[6];   //目的MAC地址
    u_char    ether_shost[6];   //源MAC地址
    u_short   ether_type;       //帧类型
}ether_header;

//ip头结构体
typedef struct ip_header {
    u_char  ver_ihl; // 版本号(4 bits) + IP头长度(4 bits)
    u_char  tos;     // 服务类型
    u_short tlen;    // 总长度
```

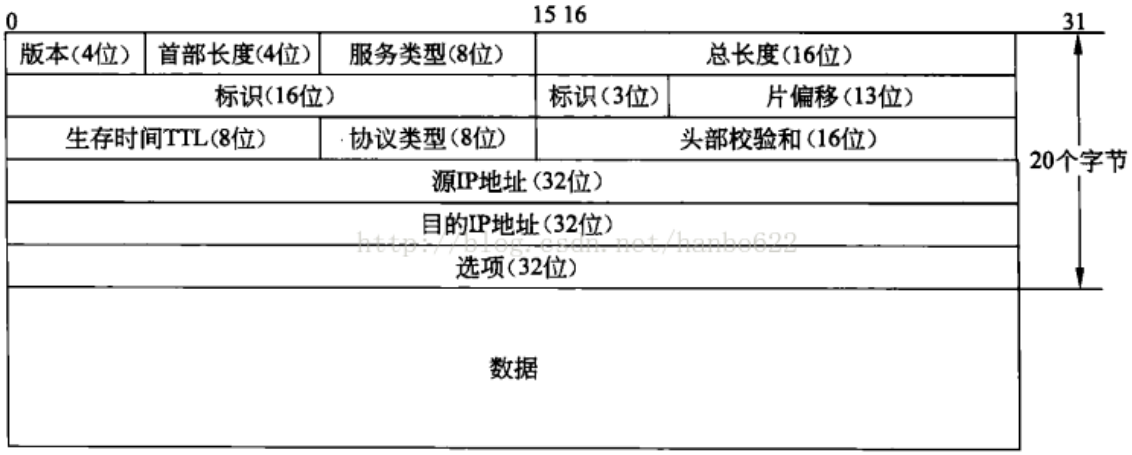
```
    u_short identification; // 标识位
    u_short flags_fo; // 标志位(3 bits) + 偏移量(13 bits)
    u_char  ttl;      // 存活时间
    u_char  proto;    // 协议
    u_short crc;      // 校验位
    ip_address  saddr; // 源地址
    ip_address  daddr; // 目标地址
    u_int  op_pad;     // 其他参数和补齐
}ip_header;
```

- 解包函数



以太帧长度固定为14字节，之后若类型为IPv4，接下来的数据都为IP数据包和CRC。于是可以使用一个以太帧头部和IP头结构体划分数据包`ether_header* eh = (ether_header*)pkt_data;`

`ip_header* ih = (struct ip_header*)(pkt_data + 14);` 之后的UDP数据包和TCP数据包所在的传输层内容则需要通过获取IP数据包内记录的包总长度，计算得到其位置才能获取到，这里不作实现。



之后打印双端MAC地址；对于通过以太帧最后两个字节确定为`0x0800`IP协议的数据包，开始进行IP包的解析，将一些大于一个字节的内容从网络字节序转换为机器字节序，按照上图给出的IP数据包结构逐个进行解析至目标IP地址为止。

```
void  packet_handler(u_char* param,
```

```
    const struct pcap_pkthdr* header,
    const u_char* pkt_data)
{
    u_short check_sum;
    u_short offset;
    u_short id;
    ether_header* eh = (ether_header*)pkt_data;
    ip_header* ih = (struct ip_header*)(pkt_data + 14);
    u_short ethernet_type;

    ethernet_type = ntohs(eh->ether_type);
    printf("++++++++++以太帧解析+++++++++\n");
    printf("数据包类型为:%x-", ethernet_type);
    switch (ethernet_type)
    {
    case 0x0800:

        printf("IPv4协议\n");
        break;
    case 0x0806:
        printf("ARP请求应答\n");
        break;
    case 0x8035:
        printf("RARP请求应答\n");
        break;
    default:
        break;
    }
    printf("源MAC地址为：%02x:%02x:%02x:%02x:%02x:%02x\n", eh-
>ether_shost[0], eh->ether_shost[1], eh->ether_shost[2], eh-
>ether_shost[3], eh->ether_shost[4], eh->ether_shost[5]);
    printf("目标MAC地址为：%02x:%02x:%02x:%02x:%02x:%02x\n",
eh->ether_dhost[0], eh->ether_dhost[1], eh->ether_dhost[2],
eh->ether_dhost[3], eh->ether_dhost[4], eh->ether_dhost[5]);

    if (ethernet_type != 0x0800)
    {
        cout << "非IPv4报文，退出解析\n" << endl;
        return;
```

```
    }
    //网络字节序转为主机字节序
    id = ntohs(ih->identification);
    check_sum = ntohs(ih->crc);
    offset = ntohs(ih->flags_fo);
    printf("++++++++++IP数据报解析+++++++++\n");
    printf("IP Version :%d\n", ih->ver_ihl >> 4);
    printf("首部长度：%d\n", (ih->ver_ihl & 0xF)*4);
    printf("服务类型：%d\n", ih->tos);
    printf("总长度：%d\n", ih->tlen);
    printf("标识：0x%x\n",id);
    printf("标志：0x%x\n", offset >> 13);
    printf("片偏移：%d\n",offset&0x1fff);
    printf("生存时间：%d\n",ih->ttl);
    printf("头部校验和：0x%x\n",check_sum);
    printf("源IP地址：%d.%d.%d.%d\n",ih->saddr.byte1,ih-
>saddr.byte2,ih->saddr.byte3,ih->saddr.byte4);
    printf("目标IP地址：%d.%d.%d.%d\n",ih->daddr.byte1,ih-
>daddr.byte2,ih->daddr.byte3,ih->daddr.byte4);
    printf("协议类型：%d-", ih->proto);
    switch (ih->proto)
    {
    case 1:
        printf("ICMP\n");
        break;
    case 2:
        printf("IGMP\n");
        break;
    case 6:
        printf("TCP\n");
        break;
    case 17:
        printf("UDP\n");
        break;
    case 41:
        printf("IPv6\n");
        break;
    default:
        break;
```

```
    }
    printf("++++++++++本次解析完成+++++++++\n\n");
    return;
}
```

- 通过使用在前期准备中学习到的获取设备列表、打开网卡和开启监测的方法，检测设备列表并打印，获得控制流输入的设备编号，开启对应的网卡设备，同时使用学习到的pcap_compile和pcap_setfilter方法设置并应用过滤器来进行包的条件过滤，获取需要捕获的包个数，最后使用pcap_loop()来调用指定的回调函数packet_handler进行解包。

```
int main()
{
    pcap_if_t* alldevs;
    pcap_if_t* d;
    int inum;
    int i = 0;
    pcap_t* adhandle;
    char errbuf[PCAP_ERRBUF_SIZE];
    //获取设备列表
    if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING,
        NULL, &alldevs, errbuf) == -1)
    {
        fprintf(stderr, "在检测设备时出现错误: %s\n", errbuf);
        exit(1);
    }
    //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(" 无可用设备! \n");
    }
    if (i == 0)
        return -1;
    printf("请输入设备编号 (1-%d):", i);
    scanf_s("%d", &inum);
```

```
    if (inum < 1 || inum > i)
    {
        printf("\n超出可用范围\n");
        pcap_freealldevs(alldevs);
        return -1;
    }

    //跳转至想要监听的设备
    for (d = alldevs, i = 0; i < inum - 1; d = d->next,
i++);
    //打开设备
    if ((adhandle = pcap_open(d->name, // name of the device
        65536, // portion of the packet to capture
               // 65536 guarantees that the whole packet
will
               // be captured on all the link layers
        PCAP_OPENFLAG_PROMISCUOUS, // promiscuous mode
        1000, // read timeout
        NULL, // authentication on the remote machine
        errbuf // error buffer
    )) == NULL)
    {
        fprintf(stderr,
            "\n%s设备不支持！%\n",
            d->name);
        pcap_freealldevs(alldevs);
        return -1;
    }
    printf("\n正在 %s 上监听...\n", d->description);
    cout << "正在设置过滤条件..."<<endl;
    char filter[40] = "";
    u_int netmask;
    struct bpf_program fcode;
    getchar();
    scanf("%[^\n]", filter);
    printf("输入要捕获的个数：");
    int count = 1;
    scanf_s("%d", &count);
    if (d->addresses != NULL)
```

```
        //获取掩码
        netmask = ((struct sockaddr_in*)(d->addresses-
>netmask))->sin_addr.S_un.S_addr;
    else
        //C类设备
        netmask = 0xffffff;
    if (pcap_compile(adhandle, &fcode, filter, 1, netmask) <
0)
    {
        fprintf(stderr,"\n无法解析过滤器，请检查输入\n");
        pcap_freealldevs(alldevs);
        return -1;
    }
    else
    {
        if (pcap_setfilter(adhandle, &fcode) < 0)
        {
            cout << "过滤器发生错误！\n" << endl;
            return -1;
        }
        cout << "正在监听" << d->description << endl;
        pcap_freealldevs(alldevs);
        pcap_loop(adhandle, count, packet_handler, NULL);
    }
    return 0;
}
```

实验结果

- 使用过滤器ip and udp，捕获2个数据包，结果如下：

```
ip and udp
输入要捕获的个数：2
正在监听Network adapter 'Intel(R) Wi-Fi 6 AX200 160MHz' on local host
++++++++++以太帧解析+++++++++
数据包类型为:800-IPv4协议
源MAC地址为：50:c2:e8:68:bf:69
目标MAC地址为：01:00:5e:00:00:fb
++++++++++IP数据报解析+++++++++
IP Version :4
首部长度：20
服务类型：0
总长度：14336
标识：0x221b
标志：0x0
片偏移：0
生存时间：1
头部校验和：0xfb69
源IP地址：10.130.176.179
目标IP地址：224.0.0.251
协议类型：17-UDP
++++++++++本次解析完成+++++++++

++++++++++以太帧解析+++++++++
数据包类型为:800-IPv4协议
源MAC地址为：50:c2:e8:68:bf:69
目标MAC地址为：01:00:5e:00:00:fb
++++++++++IP数据报解析+++++++++
IP Version :4
首部长度：20
服务类型：0
总长度：14336
标识：0x221c
标志：0x0
片偏移：0
生存时间：1
头部校验和：0xfb68
源IP地址：10.130.176.179
目标IP地址：224.0.0.251
协议类型：17-UDP
++++++++++本次解析完成+++++++++
```

可以看到，程序成功捕获并打印了MAC地址、IP地址以及其余的IP数据包的信息。

- 使用过滤器host 202.113.18.106来筛选访问校园网登录窗的数据包，结果如下：

```
host 202.113.18.106
输入要捕获的个数：2
正在监听Network adapter 'Intel(R) Wi-Fi 6 AX200 160MHz' on local host
++++++++++以太帧解析+++++++++
数据包类型为:800-IPv4协议
源MAC地址为：10:51:07:ff:17:39
目标MAC地址为：00:00:5e:00:01:0d
++++++++++IP数据报解析+++++++++
IP Version :4
首部长度：20
服务类型：0
总长度：15360
标识：0xdcbe
标志：0x2
片偏移：0
生存时间：128
头部校验和：0x0
源IP地址：10.130.36.69
目标IP地址：202.113.18.106
协议类型：6-TCP
++++++++++本次解析完成+++++++++

++++++++++以太帧解析+++++++++
数据包类型为:800-IPv4协议
源MAC地址为：00:00:5e:00:01:0d
目标MAC地址为：10:51:07:ff:17:39
++++++++++IP数据报解析+++++++++
IP Version :4
首部长度：20
服务类型：0
总长度：15360
标识：0x0
标志：0x2
片偏移：0
生存时间：62
头部校验和：0x311a
源IP地址：202.113.18.106
目标IP地址：10.130.36.69
协议类型：6-TCP
++++++++++本次解析完成+++++++++
```

可以看到，程序捕获并解析了访问校园网登录窗的数据包。

相关问题

- 理解回调函数pacp_loop()：

  pacp_loop()作为回调函数，使得程序编写者可以不用在每个自定义的函数中去关心数据包如何捕获，只需要专注于自己设计的捕获后的处理部分，编写完后即可使用回调函数来自动捕获数据包，调用处理函数。

- 数据包捕获机制：

  注意到，校园网登录过滤时，将捕获包的值设置为0，也就是永远捕获时，网页也能够正常访问，说明NPcap获取数据包并不是直接截断，而是以字符数组形式存储到程序的缓冲区中，再由一个指针传递给解析函数。查阅资料，WinPcap数据包捕获的实现算法为“当网卡设置为混杂模式时，在网内的数据包都会被发送一份到网卡上。以电子邮件为例,在发送邮件连接信息和邮件头的端口号是25端口。据此，只需要监听网络是否25端口的数据包，如果有，则进一步判断其数据报的IP是否是我们需要监听的计算机的IP，如果是，则检查是否有以指定IP命名的文件夹，动态命文件名存储数据包；如果不是，一方面我们可以存储数据包，也可以丢弃。由

此可以推测，NPcap捕获数据包时将经过过滤器的包都复制存储到文件夹中以供后续的操作。