计算机网络编程
实验报告

班级：07111707

组长：1120171189 崔程远

成员：1120172149 吴沁璇

1120172153 张澈

1120172163 王晓媛

1120172736 张鉴昊

1120172765 曾煜瑾

1120173326 曾紫飞

北京理工大学

计算机学院

2020 年 4 月

第五章 实验 9　ARP 程序

1. 实验目的

以 Winpcap 为基础实现 ARP 地址解析和 ARP 高速缓冲的记录。借助 Winpcap 发送帧函数广播 ARP 请求，捕获 ARP 响应，并记录对应关系到缓冲。

2. 实验内容

程序运行屏幕输出要点：
屏幕显示当前配置的网络适配器，并要求选择捕获适配器编号
显示广播 ARP 请求包的内容，send 函数发送广播
捕获 MAC 帧中识别 ARP 响应包，显示包内容
显示登记到缓冲中，显示缓冲区内容

3. 实验原理

WinPcap 是一个基于 Win32 平台的，用于捕获网络数据包并进行分析的开源库。它提供了以下功能：捕获原始数据包；在数据包发送给某应用程序前，根据用户指定的规则过滤数据包；将原始数据包通过网络发送出去；收集并统计网络流量信息。

ARP 是用于建立 IP 地址与 MAC 地址之间对应关系的协议。一台主机先向全网广播发送包含某个 IP 地址的 ARP 请求包，该 IP 地址对应的主机收到之后发送包含自己 MAC 地址的 ARP 响应包，本机将响应包对应的 MAC 地址登记到缓冲中。

4. 实验环境

操作系统：Windows 10
编译器：Visual Studio 2017
环境：WinPcap4.1.3　WpdPack

5. 实验步骤

以下是 C++代码和具体思路：

1) VS 中环境的配置

首先下载 WinPcap 和 WpdPack，然后在 vs 的项目中添加包含目录和库目录，修改预处理器，添加依赖项。然后就可以进行代码的编写。

2) 定义数据报头结构

如下图所示，分别定义了以太网帧头、IP 地址、ARP 数据包首部、ARP 缓冲区、ARP 报文结构，相应的数据类型进行定义。

```
/* 14字节的以太网帧头 */
typedef struct mac_header {
    u_char dmac[6]; //目的mac地址 6字节
    u_char smac[6]; //源mac地址 6字节
    u_short type;   //类型 2字节
    //string
}mac_header;

typedef struct ip_address {
    u_char byte1; //地址第一个字节 8位
    u_char byte2;
    u_char byte3;
    u_char byte4;
}ip_address;
```

```
/*28字节的 ARP帧结构 */
struct arp_header
{
    unsigned short hdType;   //硬件类型
    unsigned short proType;   //协议类型
    unsigned char hdSize;   //硬件地址长度
    unsigned char proSize;   //协议地址长度
    unsigned short op;   //操作类型，ARP请求（1），ARP应答（2），RARP请求（3），RARP应答（4）。
    u_char smac[6];   //源MAC地址
    ip_address sip;   //源IP地址
    u_char dmac[6];   //目的MAC地址
    ip_address dip;   //目的IP地址
};
```

```
/* ARP的缓冲区 */
struct arp_buffer
{
    u_char dmac[6];   //目的MAC地址
    u_char dip1;   //目的IP地址
    u_char dip2;
    u_char dip3;
    u_char dip4;
};

//定义整个arp报文包，总长度42字节
struct ArpPacket {
    mac_header ed;
    arp_header ah;
};

arp_buffer *ab[10] ;
```

3） 获得设备并打印设备列表

如下图，使用了在 WinPacp 中文文档给出的函数，来获得设备列表和打印列表设备具体信息。还定义了在当前适配器下的 send 函数。

```
/* 获得设备列表 */
if (pcap_findalldevs(&alldevs, errbuf) == -1)
{
    fprintf(stderr, "Error in pcap_findalldevs: %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(" (No description available)\n");
}

if (i == 0)
{
    printf("\nNo interfaces found! Make sure WinPcap is installed.\n");
    return -1;
}

printf("Enter the interface number (1-%d):", i);
scanf("%d", &inum);

if (inum < 1 || inum > i)
{
    printf("\nInterface number out of range.\n");
    /* 释放设备列表 */
    pcap_freealldevs(alldevs);
    return -1;
}
```

```
/* 打开适配器 */
if ((adhandle = pcap_open(d->name,  // 设备名
    65536,      // 要捕捉的数据包的部分
                // 65535保证能捕获到不同数据链路层上的每个数据包的全部内容
    PCAP_OPENFLAG_PROMISCUOUS,          // 混杂模式
    1000,       // 读取超时时间
    NULL,       // 远程机器验证
    errbuf      // 错误缓冲池
)) == NULL)
{
    fprintf(stderr, "\nUnable to open the adapter. %s is not supported by WinPcap\n");
    /* 释放设备列表 */
    pcap_freealldevs(alldevs);
    return -1;
}
```

```
//构造一个ARP请求
memset(sendbuf, 0, sizeof(sendbuf));   //ARP清零
memcpy(sendbuf, &eh, sizeof(eh));
memcpy(sendbuf + sizeof(eh), &ah, sizeof(ah));
pcap_sendpacket(adhandle, sendbuf, 42);
/*
```

4) 设置过滤器并调用回调函数

由于捕获的报文信息太多，所以需要设置一个过滤器，用于只处理 tcp 和 udp 的报文信息，其他信息都过滤。具体过滤器的设置如下。

```
u_int netmask;
char packet_filter[] = "ip and udp or ip and tcp";
```

```
//编译过滤器
if (pcap_compile(adhandle, &fcode, packet_filter, 1, netmask) < 0)
{
    fprintf(stderr, "\nUnable to compile the packet filter. Check the syntax.\n");
    /* 释放设备列表 */
    pcap_freealldevs(alldevs);
    return -1;
}

//设置过滤器
if (pcap_setfilter(adhandle, &fcode) < 0)
{
    fprintf(stderr, "\nError setting the filter.\n");
    /* 释放设备列表 */
    pcap_freealldevs(alldevs);
    return -1;
}

printf("\nlistening on %s...\n", d->description);

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

/* 开始捕捉 */
pcap_loop(adhandle, 0, packet_handler, NULL);
```

5) 获得各报文具体字段的值和打印各结构的各字段值

首先将时间戳转换为可识别的格式，然后打印数据包的时间戳和长度。ARP 包的结构是有 14 个字节的以太网帧头和 28 字节的 ARP 首部加上 18 字节的数据和 4 字节的 CRC。其中，28 字节的 ARP 帧和 14 字节以太网帧头单独定义。

将捕获报文的内容指针所指地址为以太网帧头的地址，再加上 14 个字节就可得到 ARP 首部的地址。也就是回调函数的*pkt_data 量为以太网帧头地址，加上 14 得到 ARP 帧的地址。然后进行对应的字段根据字段长度取相应的值。

最后是打印 ARP 请求包、ARP 相应包和缓冲区的具体值。由于在代码编译时，程序必须包含的头文件“pcap.h”和 string、ostream 之间存在问题导致编译失败，大概是里面有一些重复定义，然而这个问题没能得到解决，所以就只能用 printf 对结构体的每一项一个个打印出来。然后根据各个字段值的数据结构依次输出相应值即可。

```
struct tm *ltime;
char timestr[16];
ip_header *ih;
udp_header *uh;
u_int ip_len,ip_tlen,udp_len,tcp_len;
u_short sport, dport, sport2, dport2;
time_t local_tv_sec;

/* 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);
strftime(timestr, sizeof timestr, "%H:%M:%S", ltime);

/* 打印数据包的时间戳和长度 */
printf("%s.%.6d len:%d\n ", timestr, header->ts.tv_usec, header->len);
```

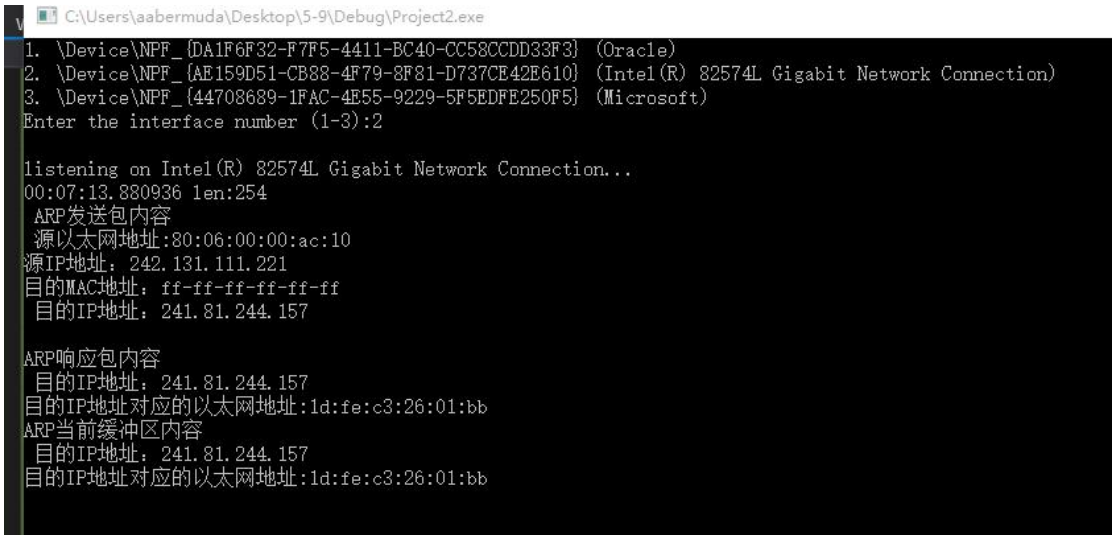
```
for (j = 0; j < 10; j++)
{
    if (flag <= 10)
    {
        flag++;
        ab[j]->dip1 = ah->dip.byte1;
        ab[j]->dip2 = ah->dip.byte2;
        ab[j]->dip3 = ah->dip.byte3;
        ab[j]->dip4 = ah->dip.byte4;

        for( i = 0; i < 6; i++)
        {
            ab[j]->dmac[i] = ah->dmac[i];
        }
    }
    else
    {
        j = 0;
        flag++;
        ab[j]->dip1 = ah->dip.byte1;
        ab[j]->dip2 = ah->dip.byte2;
        ab[j]->dip3 = ah->dip.byte3;
        ab[j]->dip4 = ah->dip.byte4;
        for (i = 0; i < 6; i++)
        {
            ab[j]->dmac[i] = ah->dmac[i];
        }
    }
}
```

```
printf("ARP发送包内容\n ");
printf("源以太网地址:%02x:%02x:%02x:%02x:%02x:%02x\n", *ah->smac, *(ah->smac + 1), *(ah->smac + 2), *(ah->smac + 3), *(ah
u_char *mac_string;
//printf("源IP地址:%s\n", (source_ip_address));
printf("源IP地址：%d.%d.%d.%d\n",(ah->sip.byte1),(ah->sip.byte2),(ah->sip.byte3),(ah->sip.byte4));
printf("目的MAC地址：ff-ff-ff-ff-ff-ff\n ");
printf("目的IP地址：%d.%d.%d.%d\n\n", (ah->dip.byte1), (ah->dip.byte2), (ah->dip.byte3), (ah->dip.byte4));

printf("ARP响应包内容\n ");
printf("目的IP地址：%d.%d.%d.%d\n", (ah->dip.byte1), (ah->dip.byte2), (ah->dip.byte3), (ah->dip.byte4));
mac_string = ah->dmac;
printf("目的IP地址对应的以太网地址:%02x:%02x:%02x:%02x:%02x:%02x\n", *mac_string, *(mac_string + 1), *(mac_string + 2), *
//printf("目的IP地址:%s\n", (destination_ip_address));

printf("ARP当前缓冲区内容\n ");
printf("目的IP地址：%d.%d.%d.%d\n", (ah->dip.byte1), (ah->dip.byte2), (ah->dip.byte3), (ah->dip.byte4));
mac_string = ah->dmac;
printf("目的IP地址对应的以太网地址:%02x:%02x:%02x:%02x:%02x:%02x\n", *mac_string, *(mac_string + 1), *(mac_string + 2), *

for (i = 0; i < 10; i++)
{
    printf("目的IP地址：%d.%d.%d.%d\n", (ab[i]->dip1), (ab[i]->dip2), (ab[i]->dip3), (ab[i]->dip4));
    mac_string = ab[i]->dmac;
    printf("对应的以太网地址:%02x:%02x:%02x:%02x:%02x:%02x\n", *mac_string, *(mac_string + 1), *(mac_string + 2), *(mac_s
    //printf("目的IP地址:%s\n", (destination_ip_address));
}
```

运行结果截图：

```
C:\Users\aabermuda\source\repos\Project1\Debug\Project1.exe
1. \Device\NPF_{DA1F6F32-F7F5-4411-BC40-CC58CCDD33F3} (Oracle)
2. \Device\NPF_{AE159D51-CB88-4F79-8F81-D737CE42E610} (Intel(R) 82574L Gigabit Network Connection)
3. \Device\NPF_{44708689-1FAC-4E55-9229-5F5EDFE250F5} (Microsoft)
Enter the interface number (1-3):
```

```
(全局范围)
C:\Users\aabermuda\source\repos\Project1\Debug\Project1.exe
1. \Device\NPF_{DA1F6F32-F7F5-4411-BC40-CC58CCDD33F3} (Oracle)
2. \Device\NPF_{AE159D51-CB88-4F79-8F81-D737CE42E610} (Intel(R) 82574L Gigabit Network Connection)
3. \Device\NPF_{44708689-1FAC-4E55-9229-5F5EDFE250F5} (Microsoft)
Enter the interface number (1-3):2

listening on Intel(R) 82574L Gigabit Network Connection...
```



实验结果如上图所示，选定了 2 号适配器后，依次打印了 ARP 请求包、ARP 响应包、缓冲区内容的 IP 地址和对应的 MAC 地址，所得的结果如上图所示。

6. 实验总结

本次实验和第四章的实验差不多，都是利用 WinPcap 进行包的捕获和分析。本次实验更侧重于对 ARP 协议的理解和分析。首先得知道输出的请求包和响应包分别是什么，其中存在着什么样的过程。然后用合适的数据结构将 ARP 包描述出来，捕获的时候将源 ip、mac 地址和目的 ip 地址都找到，然后找到与目的 ip 地址相对于的目的 mac 地址。

本次实验我也遇到了很多的问题，一开始不知道怎么表示 ARP 包的各个字段和如何表示缓冲区的存储信息。然后经过不断的试错和调试，一点点的完成了实验。通过本实验，我更加熟悉了 ARP 包的结构，对其中每个字段具体值都有了更深的理解，还了解到如何利用 ARP 进行欺骗和监听等有趣的知识。