南開大學
Nankai University

南　　开　　大　　学

网 络 空 间 安 全 学 院

网络技术与应用

实验 2：IP 数据报捕获与分析（利用 NPcap 编程捕获数据包）

姓名：郑盛东

学号：2010917

年级：2020 级

专业：信息安全、法学双学位班

指导教师：张建忠、徐敬东

2023 年 10 月 17 日

目录

一、 实验内容说明

(一) IP 数据报捕获与分析编程实验

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。

二、 实验准备

(一) 下载、配置 NPcap

需要下载 NPcap，注意，必须下载 NPcap SDK，它提供必要的函数库。使用 VS2019 时，需要进行必要的配置。

1. 添加 pcap.h 包含文件，即 include ”pcap.h”
2. 添加包含文件目录，包含 npcap sdk 提供的函数
3. 添加库文件目录，在附加库目录下添加 npcap sdk 提供的函数

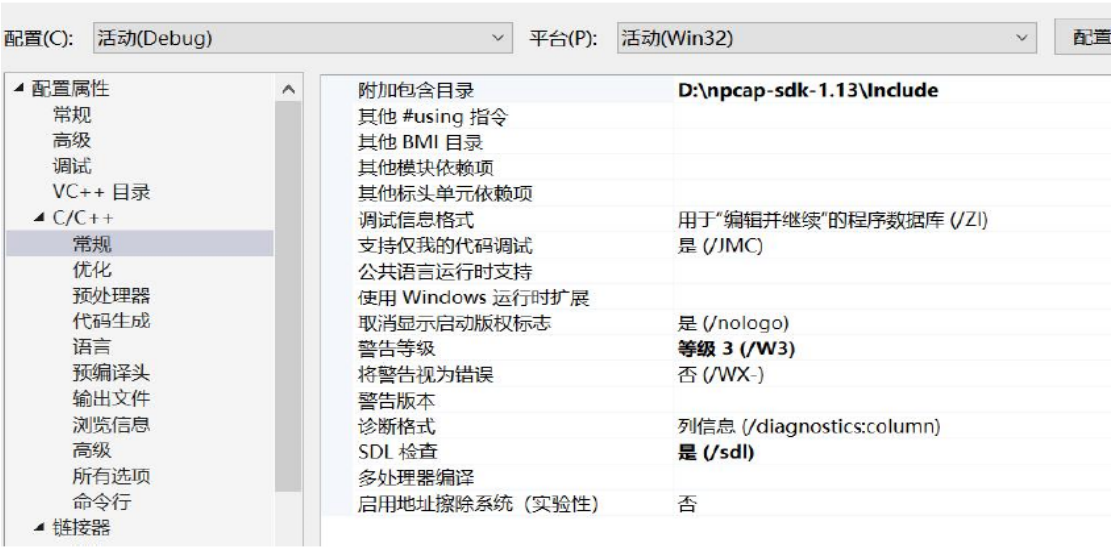

图 1: 添加包含文件

4. 添加链接时使用的库文件

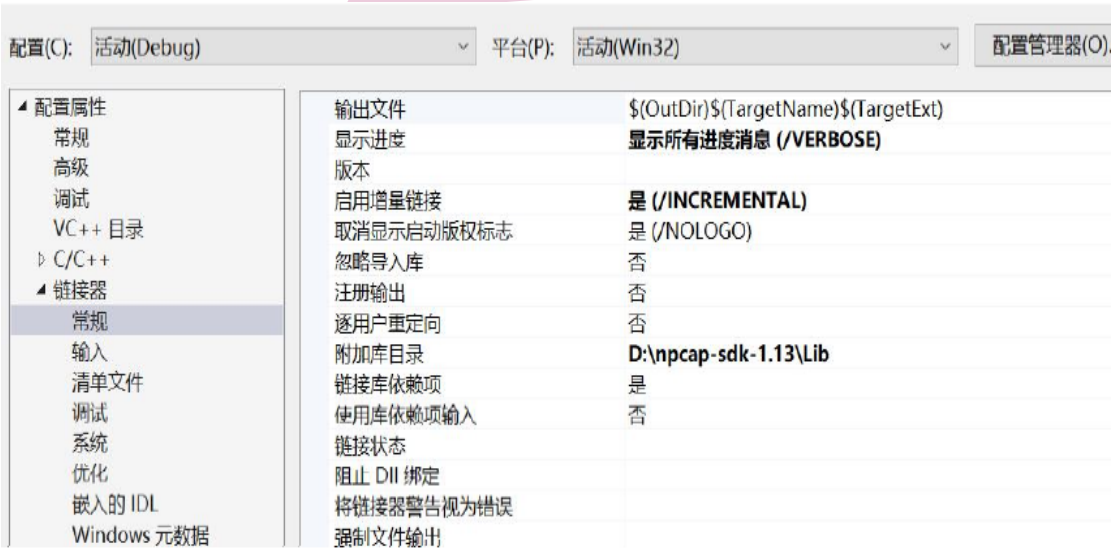

图 2: 添加库文件

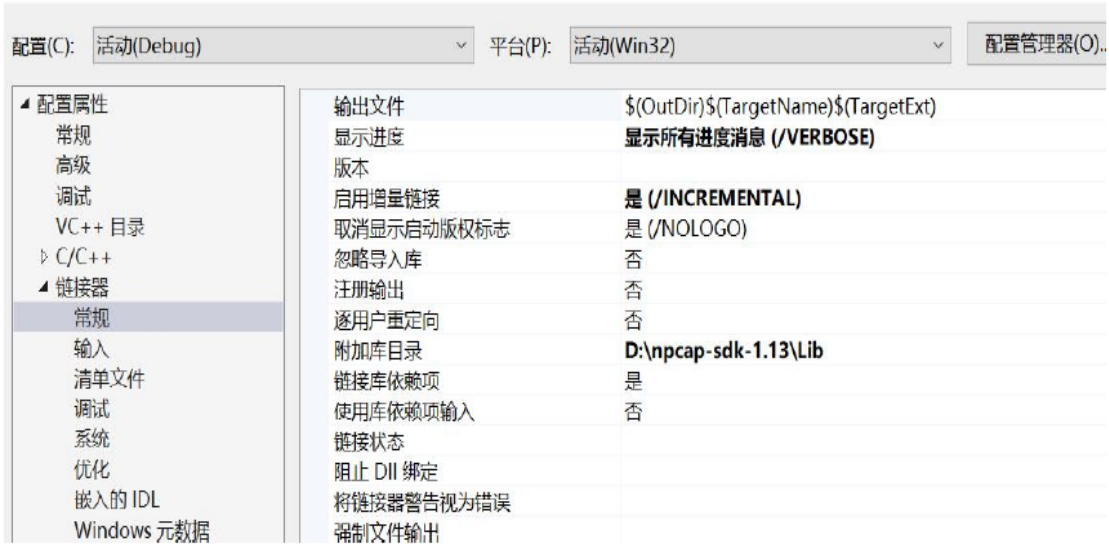

图 3: 添加链接文件

(二) 学习相关过程调用

1. 老师课上给出类定义模版，主要包括帧首部、IP 首部、数据包

2. 学习基本过程定义，参考 https://dandelioncloud.cn/article/details/1560542885714305025

3. 学习基本过程对应结果展示,参考 https://blog.csdn.net/lyshark_csdn/article/details/126688509

4. 最重要参考 RTFM。

三、 实验过程

(一) 实验核心思路

首先，通过过程调用 pcap_findalldevs_ex() 获取本机的网络接口设备；接着，根据对应数据结构，遍历并通过 pcap_open() 打开网卡适配器；接着，通过 pcap_next_ex() 直接获得数据包；最后根据老师提供的类定义模版，将数据包对应数据解析并输出。

函数调用逻辑如下:

pcap_findalldevs_ex->CreateThread->pcap_open->pcap_next_ex

(二) 数据结构和过程分析

1. 帧首部数据结构，主要包括源 MAC 地址和目的 MAC 地址，以及类型/长度字段

```
typedef struct FrameHeader_t //帧首部
{
        BYTE DesMAC[6];//目的地址
        BYTE SrcMAC[6];//源地址
        WORD FrameType;//帧类型
}FrameHeader_t;
```

2.pcap_findalldevs_ex() 过程，获取本机的网络接口设备，取得的设备，存放在 alldevs 中。adddevs 中每个元素都是 pcap_if_t 结构。

```
pcap_t *pcap_open(const char *source, int snaplen, int flags, int
    read_timeout, struct pcap_rmtauth *auth, char *errbuf);
//获取适配器列表，返回0表示正常，-1表示出错
//最终取得的设备，存放在alldevs中。adddevs中每个元素都是 pcap_if_t 结构。
```

3.pcap_if 数据结构，表示适配器列表中的一项

```
struct pcap_if {
    struct pcap_if *next;
    char *name;         /* name to hand to "pcap_open_live()" */
    char *description;    /* textual description of interface, or NULL */
    struct pcap_addr *addresses;
    bpf_u_int32 flags;    /* PCAP_IF_ interface flags */
};
typedef struct pcap_if pcap_if_t;
//其中，description为对应网卡名称
//
```

4.pcap_open() 过程，打开网卡适配器，第一个参数即为要打开的网卡的名字。pcap_open() 返回一个指向 pcap_t 的指针，它将在后续的函数 pcap_next_ex() 中使用；超时设置为 1000，1000 毫秒如果读不到数据直接返回超时，在实际中，由于同步互斥占用大量性能，可以进行一定的放宽以避免超时；第四个参数 PCAP_OPENFLAG_PROMISCUOUS = 网卡设置为混杂模式。

```
pcap_t *pcap_open(const char *source, int snaplen, int flags, int
    read_timeout, struct pcap_rmtauth *auth, char *errbuf);
//第一个参数即为要打开的网卡的名字。
pcap_open() 返回一个指向 pcap_t 的指针，它将在后续的函数pcap_next_ex() 中使用
//超时设置为1000，1000毫秒如果读不到数据直接返回超时
//第四个参数PCAP_OPENFLAG_PROMISCUOUS = 网卡设置为混杂模式
```

5.pcap_next_ex() 过程，直接获得数据包，捕获数据，最终捕获到的数据存入 pkt_data 中。捕获成功，该 pcap_next_ex 会返回 1

```
int pcap_next_ex  ( pcap_t *  p,
  struct pcap_pkthdr **  pkt_header,
  const u_char **  pkt_data
 )
 \\捕获数据，最终捕获到的数据存入pkt_data中。捕获成功，该pcap_next_ex会返回1
```

(三) 解析报文

成功捕获后，需解析报文，在此之前，需要正确定义定义以太网帧和 IP 数据包的结构。按照老师的提示，如下定义结构体。先定义帧首部，再定义 IP 首部，二者共同组成了 IP 数据报结构。而帧内部及 IP 首部也按实际存储方式依次定义了相应的数据。

NPcap 捕获到的数据包在缓冲区中是连续存放的，但通常 VC++ 默认 IDE 的设置是以 4 字节对齐的。因此, 定义这些数据结构时，一定要注意使用 pragma pack(1) 通知生成程序按照字节对齐方式生成下面的数据结构。在这些数据结构定义完成后，可以使用 pragma pack() 恢复默认对齐方式。

捕获到数据后，进行解析，由于我们事先按照数据包的实际存储方式定义好了数据包的结构，因而可以直接将 char 数组类型的 pkt_data 转化为定义好的结构 Data_t，而 Data_t 包括事先定义好的帧首部和 IP 首部，因而可直接取得相应的 MAC 地址与帧长度。

值得注意的是，Intel 采用小端存储，而网络传输字节顺序是大端，因此应用程序在输出长度时需要进行类型变换。

（四） 实验结果截图

1. 网卡设备捕获

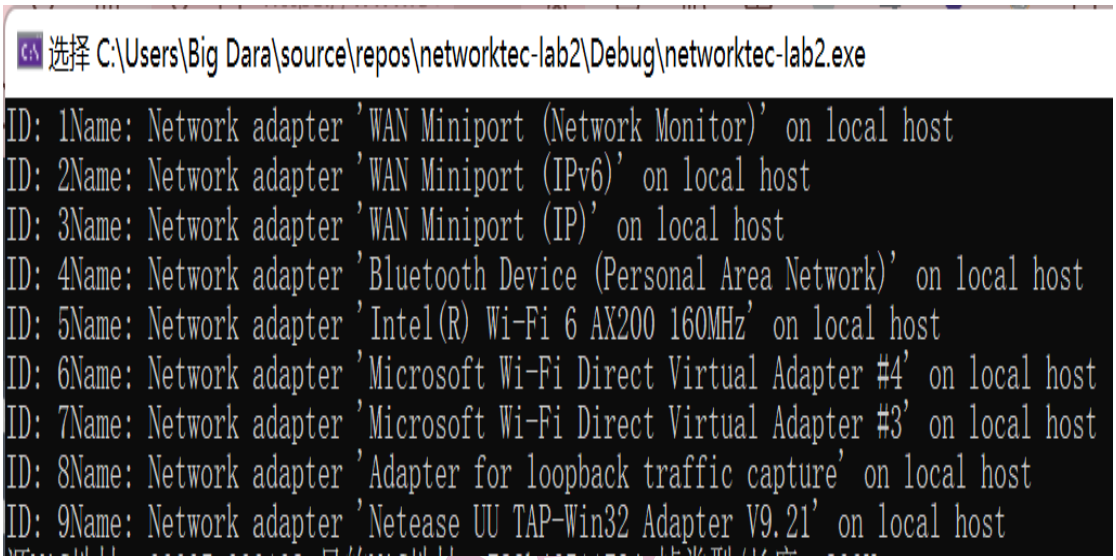

图 4: 网卡设备捕获

2. 数据包解析

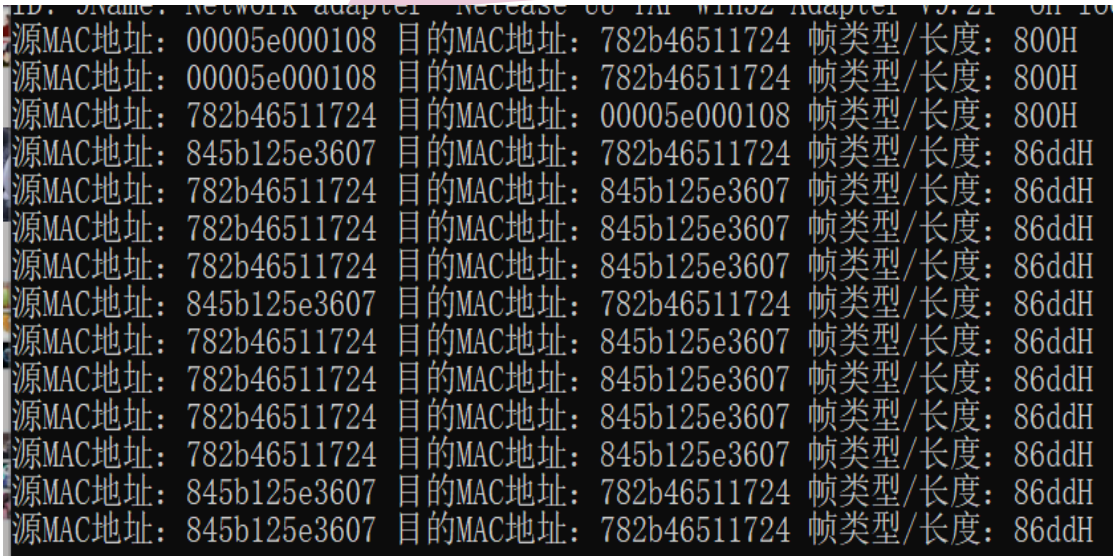

图 5: 数据包解析

（五） 调试分析

本次多线程编程实验，相比于线程通信，打开网络适配器和捕获数据包存在超时问题，互斥访问临界区和超时之间的矛盾需要进行一定的平衡。本实验中，需要同步互斥代码部分为解析数据包，必须保证只有一个线程进行此操作；而扩大临界区将占用大量性能导致大量超时。

而由于 C++ 对输入输出的缓存,也会导致一定程度上的超时和数据丢失,可以通过 ios::sync_with_stdio(false); 语句关闭默认缓存，提高整体性能；但是，如果仅仅获取前一部分的解析结构，缓存的支持反而能够保证部分的正确性，这是不言而喻的。

四、 总结

通过本次实验，学会了 NPcap 捕获的代码实现，并了解了相关数据结构和过程调用。通过阅读 RTFM 增加了对网络接口的了解。通过多线程编程，增加了对同步互斥的了解。