实验 2：数据包捕获与分析

姓名：张刘明　学号：2110049

一、实验要求

数据包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。
（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、实验过程

（1）了解 NPcap 的架构。

Npcap 是一个用于 Windows 操作系统的数据包捕获和网络分析的架构，它包括一个软件库和一个网络驱动程序。

大多数网络应用程序通过广泛使用的操作系统原语（如套接字）访问网络。使用这种方法访问网络上的数据很容易，因为操作系统处理低级细节（协议处理、数据包重组等）并提供了一个类似于读写文件的熟悉接口。

然而，有时候“简单的方式”不足以完成任务，因为某些应用程序需要直接访问网络上的数据包。换句话说，它们需要访问网络上的“原始”数据，而不需要操作系统进行协议处理。

Npcap 的目的是为 Windows 应用程序提供这种类型的访问。它提供以下功能：

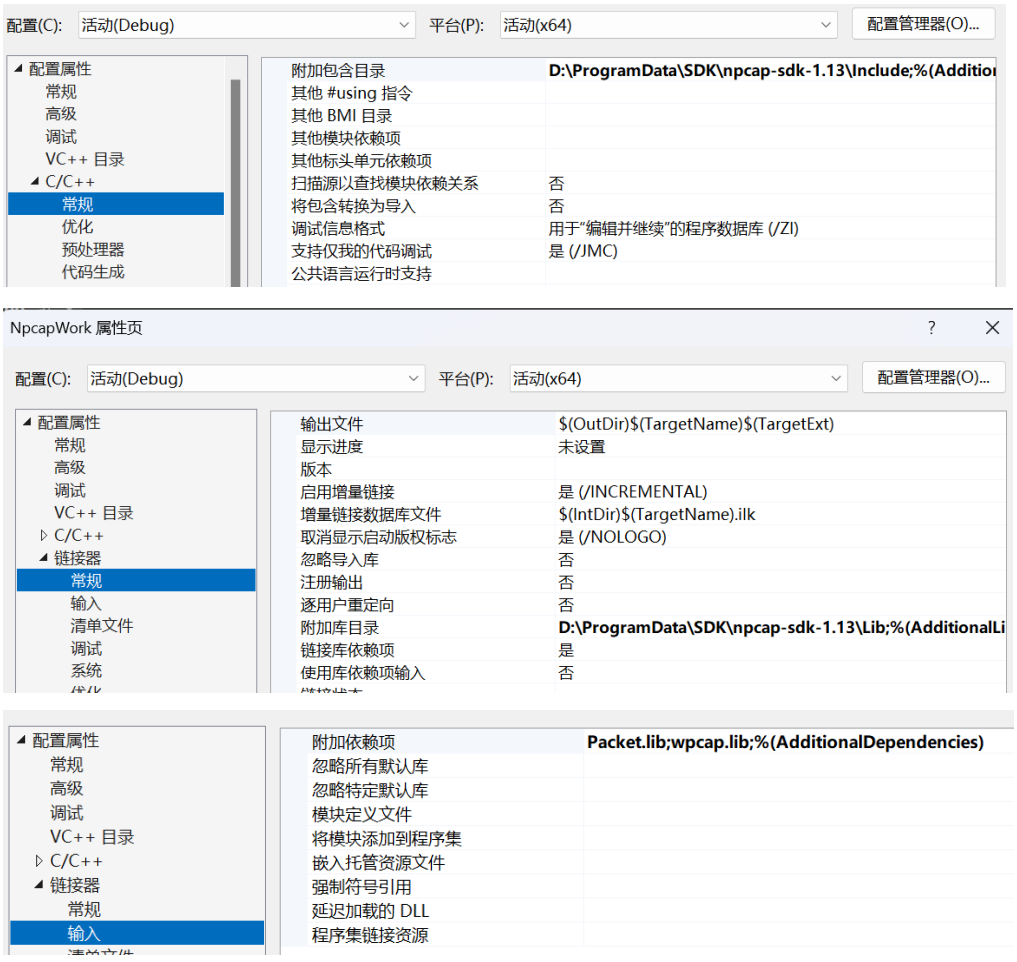
1. 捕获原始数据包，包括发送到运行它的计算机以及其他主机交换的数据包（在共享媒体上）。
2. 根据用户指定的规则在将数据包传递给应用程序之前对数据包进行过滤。
3. 发送原始数据包到网络。
4. 收集有关网络流量的统计信息。

这些功能是通过一个设备驱动程序实现的，该驱动程序安装在 Windows 内核的网络部分中，还包括几个 DLL 文件。

NPcap 的架构：

1. **驱动程序层：**NPcap 的核心是一个驱动程序，它安装在 Windows 操作系统内核层。这个驱动程序允许 NPcap 与网络接口进行交互，捕获数据包，以及注入数据包到网络。
2. **用户模式库：**NPcap 包含一些用户模式库，用于管理和控制抓包功能。npcap.dll：这个库提供了对 NPcap 驱动程序的用户模式接口，允许应用程序与驱动程序通信。wpcap.dll：这是兼容 WinPcap 的库，允许现有使用 WinPcap 的应用程序在不修改代码的情况下使用 NPcap。
3. **NPF Service：**NPF 服务是一个用户模式进程，它提供了控制 NPcap 的命令行工具。用户可以使用这些工具配置和管理 NPcap。
4. **Packet Capture API：**NPcap 支持 Packet Capture API，这是一种通用的抓包 API，让应用程序能够访问捕获和分析网络数据包。

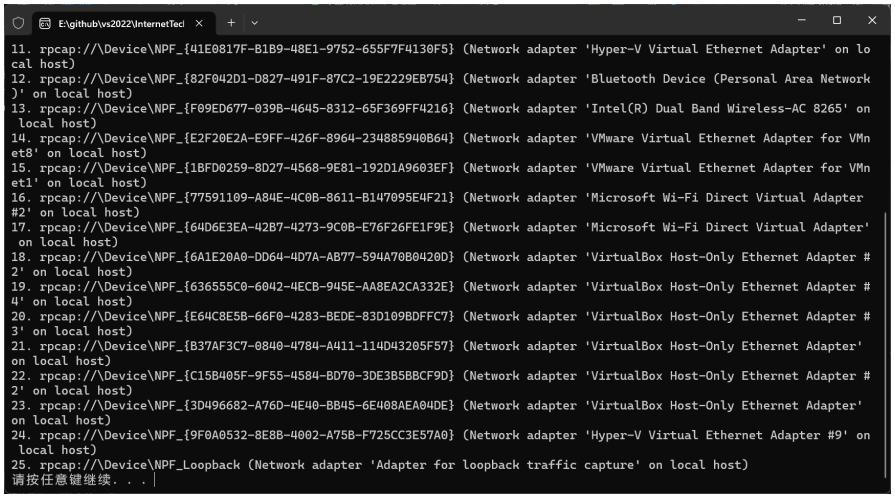
SDK 配置：




```
#include "pcap.h"
```

```
int main(){
    pcap_if_t* alldevs;
    pcap_if_t* d;
    int i = 0;
    char errbuf[PCAP_ERRBUF_SIZE];
    /* Retrieve the device list from the local machine */
    if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL /* auth is not needed */,
&alldevs, errbuf) == -1){
        fprintf(stderr, "Error in pcap_findalldevs_ex: %s\n", errbuf);
        exit(1);
    }
    /* Print the list */
    for (d = alldevs; d != NULL; 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 Npcap is installed.\n");
        return 0;
    }
    /* We don't need any more the device list. Free it */
    pcap_freealldevs(alldevs);
    system("pause");
    return 0;
}
```

测试：获得网络适配器列表→成功



(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。

设备列表获取方法:

NpcapDeviceList.h

```
#pragma once
#ifndef NPCAPDEVICELIST_H
#define NPCAPDEVICELIST_H

#include <vector>
#include <string>

struct NetworkDevice {
    std::string name;
    std::string description;
};

std::vector<NetworkDevice> GetNpcapDeviceList();

#endif
```

NpcapDeviceList.cpp

```
#include "NpcapDeviceList.h" // 头文件
#include <pcap.h>
#include <iostream>

std::vector<NetworkDevice> GetNpcapDeviceList() {
    std::vector<NetworkDevice> deviceList;
    pcap_if_t* alldevs;
    char errbuf[PCAP_ERRBUF_SIZE];

    // 未找到网络设备
    if (pcap_findalldevs(&alldevs, errbuf) == -1) {
        std::cerr << "没有找到网络设备\n";
        return deviceList;
    }

    // 遍历网络设备
    for (pcap_if_t* dev = alldevs; dev != nullptr; dev = dev->next) {
        NetworkDevice device;
        device.name = dev->name;
```

```
        device.description = (dev->description) ? dev->description : "N/A";
        deviceList.push_back(device);
    }

    // 释放网络设备的内存
    pcap_freealldevs(alldevs);
    return deviceList;
}
```

网卡设备打开方法：

NpcapDeviceControl.h

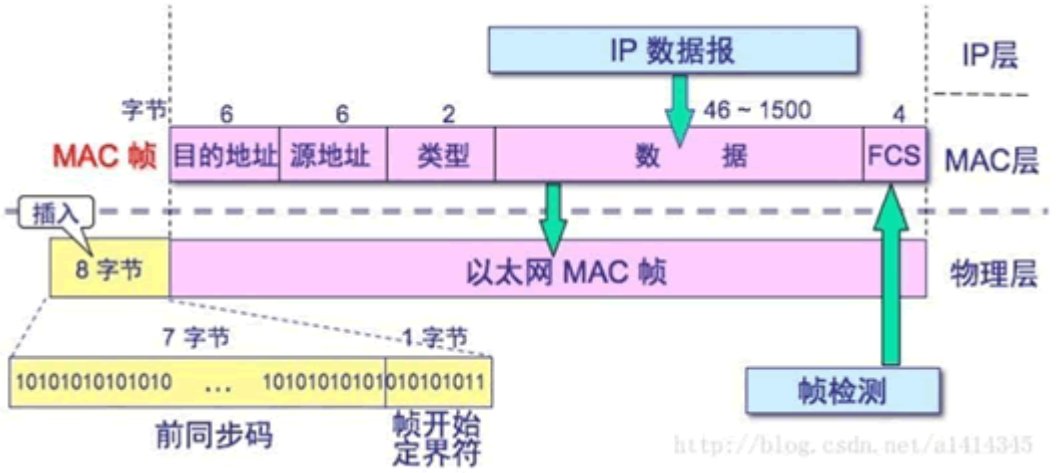
```
#pragma once
#ifndef NPCAPDEVICECONTROL_H
#define NPCAPDEVICECONTROL_H

#include <pcap.h>

pcap_t* OpenNpcapDevice(const char* deviceName);

#endif
```

NpcapDeviceControl.cpp

```
#include "NpcapDeviceControl.h"
#include <iostream>

pcap_t* OpenNpcapDevice(const char* deviceName) {
    char errbuf[PCAP_ERRBUF_SIZE];
    pcap_t* handle = pcap_open(deviceName, 65536, PCAP_OPENFLAG_PROMISCUOUS, 1000,
nullptr, errbuf);

    if (handle == nullptr) {
        std::cerr << "打开设备错误 " << deviceName << ": " << errbuf << std::endl;
    }

    return handle;
}
```

数据包捕获方法：

数据包结构:



1. 帧首部（前 16 字节）：

DesMAC：目的 MAC 地址，占 6 个字节（48 位）。

SrcMAC：源 MAC 地址，占 6 个字节（48 位）。

TrameType：帧类型，占 2 个字节（16 位）。通常用于指示以太网帧中携带的协议类型（如 IPv4 或 IPv6）。

2. IP 首部：

Ver_HLen：版本号和首部长度，占 1 个字节。通常包含 IP 协议版本号和 IP 首部的长度。

TOS：服务类型，占 1 个字节。用于指示服务质量的相关信息。

TotalLen：总长度，占 2 个字节。表示整个 IP 数据报的长度（包括 IP 首部和数据部分）。

ID：标识，占 2 个字节。通常用于分片和重组数据报。

Flag_Segment：标志和片偏移，占 2 个字节。用于处理分片和重组。

TTL：生存时间，占 1 个字节。表示数据报在网络上可以存在的时间（跳数）。

Protocol：协议，占 1 个字节。指示数据报中的上层协议类型（如 TCP 或 UDP）。

Checksum：校验和，占 2 个字节。用于校验 IP 首部的完整性。

SrcIP：源 IP 地址，占 4 个字节（32 位）。

DstIP：目的 IP 地址，占 4 个字节（32 位）。

3. 包含帧首部和 IP 首部的数据包：

FrameHeader：帧首部，类型为 FrameHeader_t，包含了 MAC 地址和帧类型信息。

IPHeader：IP 首部，类型为 IPHeader_t，包含了 IP 数据报的各种信息。

```
#include "NpcapPacketCapture.h"
#include <iostream>

pcap_t* InitializePacketCapture(const char* device) {
    char errbuf[PCAP_ERRBUF_SIZE];
    pcap_t* handle = pcap_open_live(device, BUFSIZ, 1, 1000, errbuf);
    if (handle == nullptr) {
        std::cerr << "句柄为空\n";
    }
    return handle;
}

void CapturePackets(pcap_t* handle, int numPackets, pcap_handler callback) {
    if (pcap_loop(handle, numPackets, callback, nullptr) == -1) {
        std::cerr << "捕获数据包失败\n";
    }
    struct pcap_pkthdr header;
    const u_char* packet;

    while (int returnValue = pcap_next_ex(handle, &header, &packet) >= 0) {
        // 检查以太网帧是否有足够的长度
        if (header.len < 14) {
            continue;
        }

        // 从以太网帧中提取源MAC地址和目的MAC地址
        unsigned char* sourceMac = (unsigned char*)(packet);
        unsigned char* destMac = (unsigned char*)(packet + 6);

        // 提取类型/长度字段的值
        unsigned short etherType = ntohs(*(unsigned short*)(packet + 12));
```

```
        // 在屏幕上显示数据包信息
        printf("Source MAC: %02X:%02X:%02X:%02X:%02X:%02X\n", sourceMac[0],
sourceMac[1], sourceMac[2], sourceMac[3], sourceMac[4], sourceMac[5]);
        printf("Dest MAC: %02X:%02X:%02X:%02X:%02X:%02X\n", destMac[0], destMac[1],
destMac[2], destMac[3], destMac[4], destMac[5]);
        printf("Type/Length: 0x%04X\n", etherType);
        printf("\n");
    }
}
```

（3）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。

获取网络设备列表： 使用 NPcap API 列出全部可用的网络设备并对每一个网络设备进行标号，通过窗口输入标号选择其中一个；

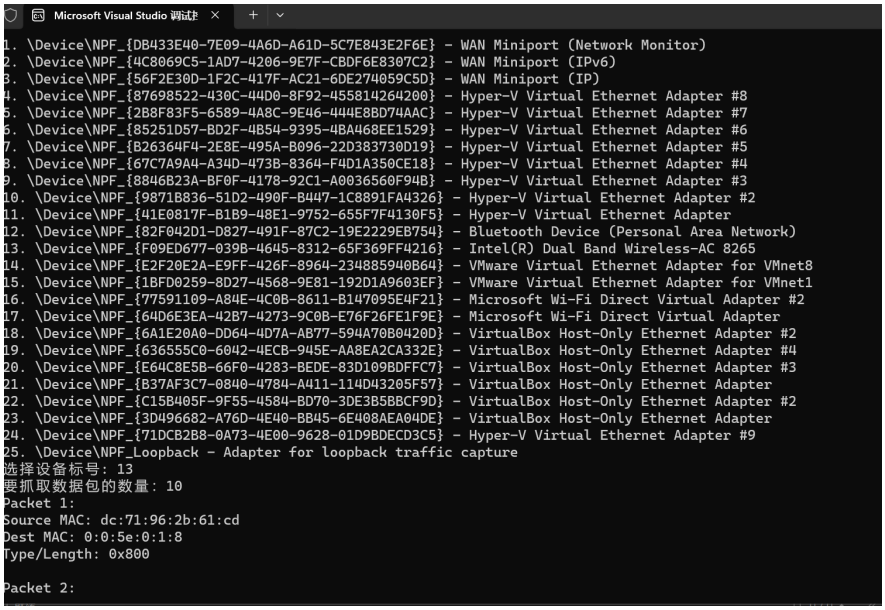
打开网络设备： 使用 NPcap API 打开对应标号的网络设备；

捕获数据包： 根据窗口的输入数量，捕获对应数量的数据包；

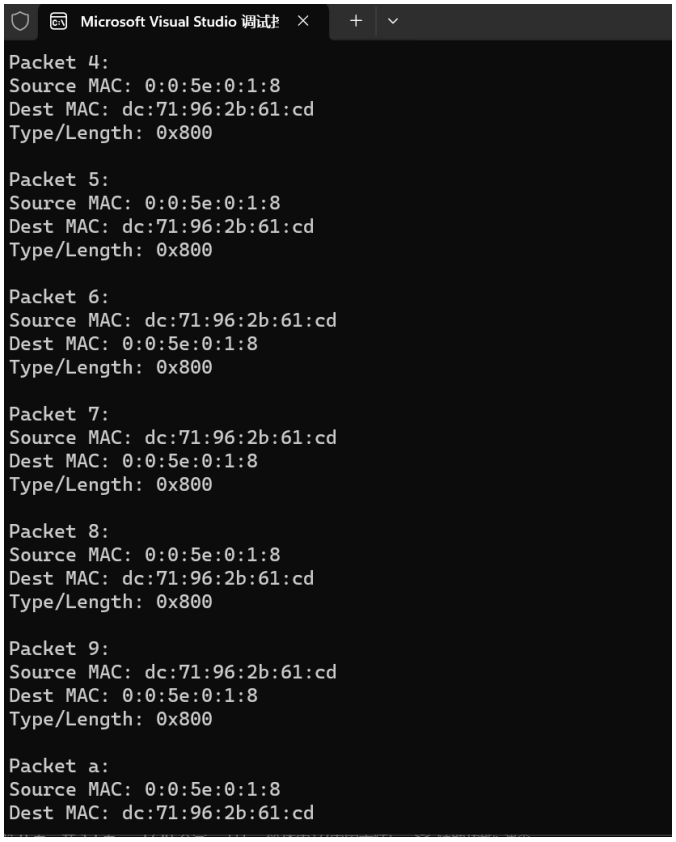
解析数据包： 捕获数据包，根据数据包的结构，解析以太网帧头部，提取源 MAC 地址、目的 MAC 地址以及类型/长度字段的值；

打印数据：使用标准输出来将解析后的数据显示在屏幕上，源 MAC 地址、目的 MAC 地址和类型/长度字段的值；

关闭设备：捕获数据包完成后，关闭网络设备。



（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。

```
#include <iostream>
#include <pcap.h>
#include <vector>

// 用于存储网络设备信息
struct NetworkDevice {
    std::string name;
    std::string description;
};
// 获取本机的所有网络设备并标号存储
std::vector<NetworkDevice> GetNpcapDeviceList();
// 打开选定的设备
pcap_t* OpenNpcapDevice(const char* deviceName);
// 捕获数据包并解析
void CaptureAndDisplayPackets(pcap_t* handle, int numPackets);

int main() {
    std::vector<NetworkDevice> devices = GetNpcapDeviceList();
    if (devices.empty()) {
        std::cerr << "No devices found. Exiting." << std::endl;
        return 1;
    }
    std::cout << "选择设备标号: ";
    int deviceIndex;
    std::cin >> deviceIndex;
    if (deviceIndex < 1 || deviceIndex > devices.size()) {
        std::cerr << "Invalid device selection. Exiting." << std::endl;
        return 1;
    }
    // 打开选定的设备
    pcap_t* handle = OpenNpcapDevice(devices[deviceIndex - 1].name.c_str());
    if (handle != nullptr) {
        std::cout << "要抓取数据包的数量: ";
        int numPackets;
        std::cin >> numPackets;
        CaptureAndDisplayPackets(handle, numPackets);
        pcap_close(handle);
    }
    return 0;
}
// 获取本机的所有网络设备并标号存储
std::vector<NetworkDevice> GetNpcapDeviceList() {
```

```
    std::vector<NetworkDevice> deviceList;
    pcap_if_t* alldevs;
    char errbuf[PCAP_ERRBUF_SIZE];

    if (pcap_findalldevs(&alldevs, errbuf) == -1) {
        std::cerr << "Error in pcap_findalldevs: " << errbuf << std::endl;
        return deviceList;
    }
    int deviceIndex = 1;
    for (pcap_if_t* dev = alldevs; dev != nullptr; dev = dev->next) {
        NetworkDevice device;
        device.name = dev->name;
        device.description = (dev->description) ? dev->description : "N/A";
        deviceList.push_back(device);
        std::cout << deviceIndex << ". " << device.name << " - " << device.description
<< std::endl;
        deviceIndex++;
    }

    pcap_freealldevs(alldevs);
    return deviceList;
}
// 打开选定的设备
pcap_t* OpenNpcapDevice(const char* deviceName) {
    char errbuf[PCAP_ERRBUF_SIZE];
    pcap_t* handle = pcap_open_live(deviceName, BUFSIZ, 1, 1000, errbuf);
    if (handle == nullptr) {
        std::cerr << "Error opening device: " << errbuf << std::endl;
    }
    return handle;
}
// 捕获数据包并解析
void CaptureAndDisplayPackets(pcap_t* handle, int numPackets) {
    struct pcap_pkthdr header;
    const u_char* packet;
    for (int i = 0; i < numPackets; ++i) {
        packet = pcap_next(handle, &header);
        if (packet == nullptr) {
            std::cerr << "No more packets to capture." << std::endl;
            break;
        }
        // 解析以太网帧
        unsigned char* sourceMac = (unsigned char*)(packet);
        unsigned char* destMac = (unsigned char*)(packet + 6);
```

```
        unsigned short etherType = (packet[12] << 8) + packet[13];
        // 显示结果
        std::cout << "Packet " << i + 1 << ":\n";
        std::cout << "Source MAC: ";
        for (int j = 0; j < 6; ++j) {
            std::cout << std::hex << (int)sourceMac[j];
            if (j < 5) std::cout << ':';
        }
        std::cout << "\nDest MAC: ";
        for (int j = 0; j < 6; ++j) {
            std::cout << std::hex << (int)destMac[j];
            if (j < 5) std::cout << ':';
        }
        std::cout << "\nType/Length: 0x" << std::hex << etherType << "\n\n";
    }
}
```