实验二：数据包捕获与分析

一、实验内容

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。

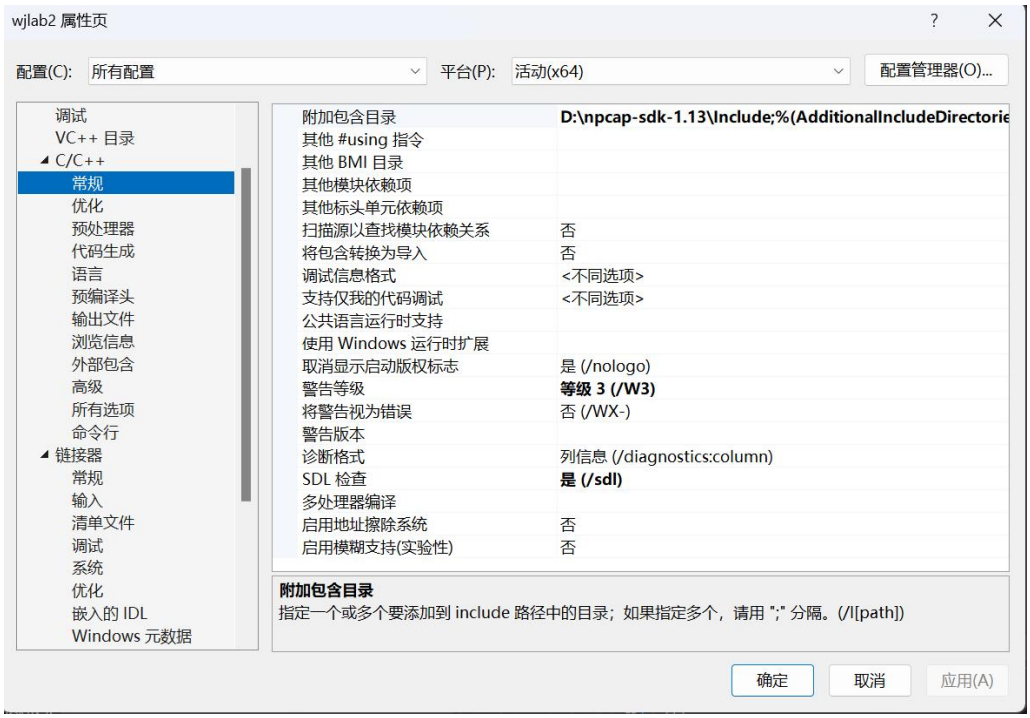
二、实验步骤

1.Npcap 是一个在 Windows 操作系统上进行网络数据包捕获的工具，是 WinPcap 的改进版,Npcap 的核心部分是其网络驱动程序 NDIS,它与 Windows 网络协议栈集成，允许它在网络协议层捕获数据包。其中还包括 NPF，它是一种网络数据包捕获和过滤技术。Npcap 还附带一些实用程序和示例应用程序，如 wireshark 等。Npcap 中还包含一些动态链接库文件，如 packet.dll，Wpcap.dll。packet.dll 包含了与网络数据包捕获相关的函数和方法，允许应用程序与 NPcap 驱动程序进行交互，以进行数据包捕获和分析。为应用程序提供了捕获和处理数据包的接口。

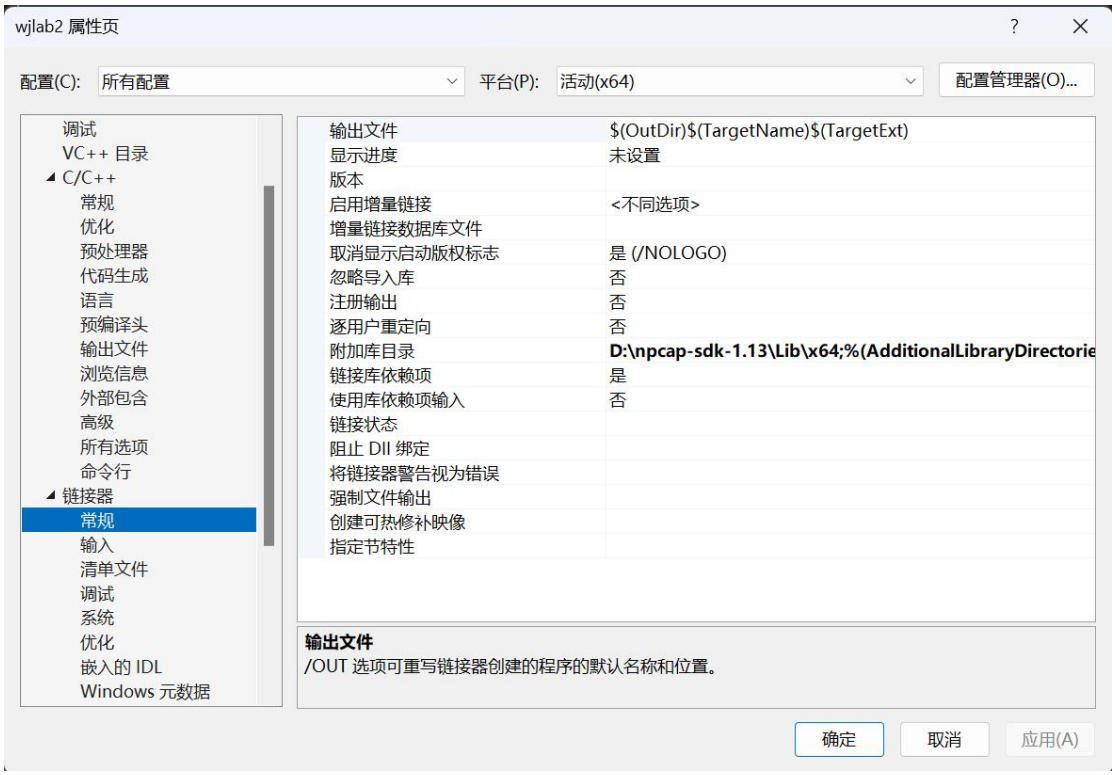
Wpcap.dll 包含了与网络数据包捕获和处理相关的功能，为应用程序提供了用于配置捕获参数、设置网络适配器、捕获数据包等功能。

2.使用 Npcap 前需要配置相关软件包以及头文件。

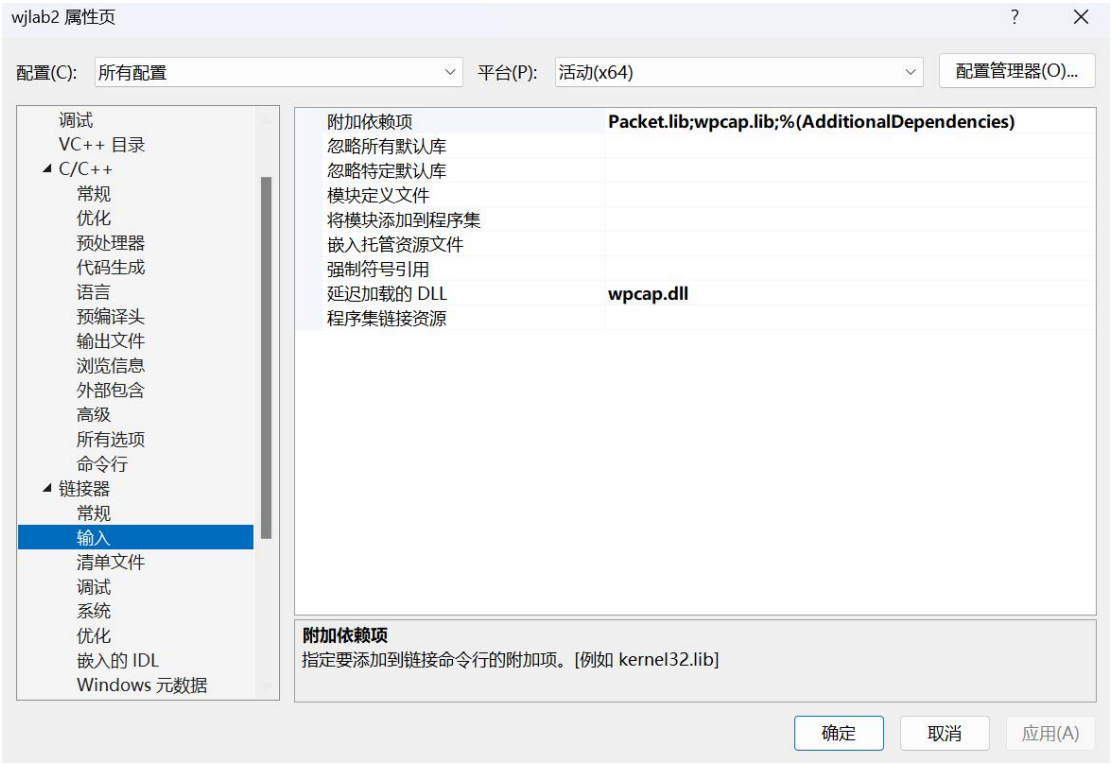
（1）首先需要配置附加包含目录，即 npcap-sdk 文件中的 include 文件夹。



（2）配置附加库目录，即 npcap-sdk 文件中的 Lib 文件夹。



（3）配置附加依赖项，也就是 Lib 文件夹中的两个动态链接库 Packet.dll 和 wpcap.dll。



（4）获取设备列表可以使用 Npcap 中的函数时需要包含头文件 "pcap.h"，然后调用 pcap_findalldevs 函数来获取网络设备列表。打开网卡设备可以使用 pcap_open 或 pcap_open_live 函数来打开所选的设备以进行数据包捕获。数据包的捕获则可以通过使用 pcap_next_ex 函数来捕获一个数据包，

3. 编写一段代码捕获本机的数据包，其中通过 pcap_open_live，pcap_findalldevs_ex，pcap_loop 等函数对数据包进行抓取和分析，使用 ip_protocol_packet_callback 函数进行 IP 地址的计算，并且使用了 epp_callback 函数对数据链路层进行解析，从而分析出 MAC 源地址和目的地址。最后使用 Catch 函数抓取本机的所有网卡，并对整个抓取结果进行合理化输出。

```
ID 1  Name: Network adapter 'Microsoft' on local host
ID 2  Name: Network adapter 'VMware Virtual Ethernet Adapter' on local host
ID 3  Name: Network adapter 'VMware Virtual Ethernet Adapter' on local host
请输入要获取数据包的网卡号
1
正在监听Network adapter 'Microsoft' on local host
请输入想要捕获的数据包个数:
2

第[ 1 ]个IP数据包被捕获
----------链路层协议---------
以太网的类型为 :0800
网络层使用的是IPv4协议
Mac源地址:      38:d5:7a:e0:13:55:
Mac目的地址:    00:00:5e:00:01:08:
-----------解析IP层数据包-----------
IP版本:IPv4
IP协议首部长度:20
服务类型:0
数据包总长度:59
标识:20518
片偏移:0
生存时间:128
首部检验和:0
(计算所得)首部检验和:51334
源IP地址:10.136.77.244
目的IP:222.30.45.41
协议号:17
传输层协议是:UDP
```

以上是程序的抓取结果，可以看到其中的 IPv4 地址与本机的地址相同，可见抓取成功。

```
无线局域网适配器 WLAN:

   连接特定的 DNS 后缀 . . . . . :
   IPv6 地址 . . . . . . . . . . . . : 2001:250:401:6571:615b:efbc:2b14:9648
   临时 IPv6 地址. . . . . . . . . . : 2001:250:401:6571:6182:f450:e7c:650a
   本地链接 IPv6 地址. . . . . . . . : fe80::3986:6cf4:65bc:2c81%6
   IPv4 地址 . . . . . . . . . . . . : 10.136.77.244
   子网掩码  . . . . . . . . . . . . : 255.255.128.0
   默认网关. . . . . . . . . . . . . : fe80::865b:12ff:fe5e:360c%6
                                       10.136.0.1
```

三、实验感想与研讨

通过数据包捕获与分析实验，我深入了解了网络通信是如何在物理层、数据链路层、网络层和传输层等不同层次上工作的。并了解了数据包如何在网络中传输，以及不同协议（如 TCP 和 UDP）的作用。通过分析捕获的数据包，我学会了如何解读不同协议的数据包头部，以及如何识别数据包中的重要信息。掌握了协议和数据包结构的解析技能，提升了网络安全意识，并提升了实际应用的技能。

四、实验代码

```
#include <Winsock2.h>
#include<Windows.h>
#include<iostream>
#include <ws2tcpip.h>
#include "pcap.h"
#include "stdio.h"
#include<time.h>
#include <string>
#pragma comment(lib, "Packet.lib")
#pragma comment(lib,"wpcap.lib")
#pragma comment(lib,"ws2_32.lib")
#pragma warning( disable : 4996 )
#define _WINSOCK_DEPRECATED_NO_WARNINGS
using namespace std;
#pragma pack(1)
//帧的首部
struct e_head
{
    uint8_t ether_dst[6];
    uint8_t ether_src[6];
    uint16_t ether_type;
};
//IP 的首部
struct ip_head
{
    uint8_t ip_header_length : 4,ip_version : 4;
    uint8_t tos;
    uint16_t total_length;
    uint16_t ip_id;
    uint16_t ip_offset;
    uint8_t ttl;
    uint8_t ip_protocol;
    uint16_t ip_checksum;
    uint16_t cal_checksum();
    struct in_addr  ip_source_address;
    struct in_addr ip_destination_address;
};

uint16_t ip_head::cal_checksum()
{
    uint32_t cal_checksum = 0;
    uint16_t var1 = (((this->ip_version << 4) + this->ip_header_length)
```

```
<< 8) + this->tos;
    uint16_t var2 = (this->ttl << 8) + this->ip_protocol;
    uint16_t var3 = ntohl(this->ip_source_address.S_un.S_addr) >> 16;
    uint16_t var4 = ntohl(this->ip_source_address.S_un.S_addr);
    uint16_t var5 = ntohl(this->ip_destination_address.S_un.S_addr) >>
16;
    uint16_t var6 = ntohl(this->ip_destination_address.S_un.S_addr);
    cal_checksum = cal_checksum + var1 + ntohs(this->total_length) +
ntohs(this->ip_id) + ntohs(this->ip_offset) + var2 + var3 + var4 + var5
+ var6;
    cal_checksum = (cal_checksum >> 16) + (cal_checksum & 0xffff);
    cal_checksum += (cal_checksum >> 16);
    return (uint16_t)(~cal_checksum);
}
//分析 IP 数据包
void  ip_protocol_packet_callback(u_char*  argument,  const  struct
pcap_pkthdr* packet_header, const u_char* packet_content)
{
    ip_head* ip_protocol;
    uint32_t  head_length;
    uint16_t  offset;
    uint8_t  tos;
    uint16_t checksum;
    ip_protocol = (struct ip_head*)(packet_content + 14);
    checksum = ntohs(ip_protocol->ip_checksum);
    head_length = ip_protocol->ip_header_length * 4;
    tos = ip_protocol->tos;
    offset = ntohs(ip_protocol->ip_offset);
    cout << "-----------解析 IP 层数据包----------- " << endl;
    printf("IP 版本:IPv%d\n", ip_protocol->ip_version);
    cout << "IP 协议首部长度:" << head_length << endl;
    printf("服务类型:%d\n", tos);
    cout <<"数据包总长度:" <<ntohs(ip_protocol->total_length) <<endl;
    cout << "标识:" << ntohs(ip_protocol->ip_id) << endl;
    cout << "片偏移:" << (offset & 0x1fff) * 8 << endl;
    cout << "生存时间:" << int(ip_protocol->ttl) << endl;
    cout << "首部检验和:" << htons(checksum) << endl;
    cout  <<  "(计算所得)首部检验和:"  <<
htons(ip_protocol->cal_checksum()) << endl;
    char src[17];
    ::inet_ntop(AF_INET, (const void*)&ip_protocol->ip_source_address,
src, 17);
    cout << "源 IP 地址:" << src << endl;
    char dst[17];
```

```
    ::inet_ntop(AF_INET,                                    (const
void*)&ip_protocol->ip_destination_address, dst, 17);
    cout << "目的 IP:" << dst << endl;
    printf("协议号:%d\n", ip_protocol->ip_protocol);
    cout << "传输层协议是:";
    switch (ip_protocol->ip_protocol)
    {
    case 1:
        cout << "ICMP" << endl;
        break;
    case 2:
        cout << "IGMP" << endl;
        break;
    case 3:
        cout << "GGP" << endl;
        break;
    case 6:
        cout << "TCP" << endl;
        break;
    case 8:
        cout << "EGP" << endl;
        break;
    case 17:
        cout << "UDP" << endl;
        break;
    case 89:
        cout << "OSPF" << endl;
        break;
    default:break;
    }
}
//解析数据链路层，获取 MAC 地址
void epp_callback(u_char* argument, const pcap_pkthdr* packet_header,
const u_char* packet_content)
{
    uint16_t e_type;
    e_head* e_protocol = (e_head*)packet_content;
    uint8_t* mac_src;
    uint8_t* mac_dst;
    static int packet_number = 1;
    e_type = ntohs(e_protocol->ether_type);
    e_protocol = (e_head*)packet_content;
    mac_src = e_protocol->ether_src;
    mac_dst = e_protocol->ether_dst;
```

```
    cout << endl;
    printf("第[ %d ]个 IP 数据包被捕获\n", packet_number);
    cout << "----------链路层协议---------" << endl;;
    printf("以太网的类型为 :%04x\n", e_type);
    switch (e_type)
    {
    case 0x0800:
        cout << "网络层使用的是 IPv4 协议" << endl;
        break;
    case 0x0806:
        cout << "网络层使用的是 ARP 协议" << endl;
        break;
    case 0x8035:
        cout << "网络层使用的是 RARP 协议" << endl;
        break;
    default: break;
    }
    printf("Mac 源地址:\t%02x:%02x:%02x:%02x:%02x:%02x:\n", *mac_src,
*(mac_src + 1), *(mac_src + 2), *(mac_src + 3), *(mac_src + 4), *(mac_src
+ 5));//X 表示以十六进制形式输出 02 表示不足两位，前面补 0 输出
    printf("Mac 目的地址:\t%02x:%02x:%02x:%02x:%02x:%02x:\n", *mac_dst,
*(mac_dst + 1), *(mac_dst + 2), *(mac_dst + 3), *(mac_dst + 4), *(mac_dst
+ 5));
    switch (e_type)
    {
    case 0x0800:
        ip_protocol_packet_callback(argument,            packet_header,
packet_content);
        break;
    default:
        cout << "不是 IP 数据包，不进行解析" << endl;
        break;
    }
    packet_number++;
}
void Catch()
{
    pcap_if_t* allAdapters;
    pcap_if_t* ptr;
    pcap_t* pcap_handle;
    int index = 0;
    int num = 0;
    int i = 0;
    char errbuf[PCAP_ERRBUF_SIZE];
```

```
    int flag = 0;
    char packet_filter[40] = "";
    struct bpf_program fcode;
    u_int netmask;
    if  (pcap_findalldevs_ex(PCAP_SRC_IF_STRING,  NULL,  &allAdapters,
errbuf) != -1)
    {
        for (ptr = allAdapters; ptr != NULL; ptr = ptr->next)
        {
            ++index;
            if (ptr->description)
                printf("ID %d  Name: %s \n", index, ptr->description);
        }
    }
    if (index == 0)
    {
        cout <<"没有找到接口，请确认是否安装了 Npcap 或 WinPcap" <<endl;
    }
    cout << "请输入要获取数据包的 ID" << endl;
    cin >> num;
    if (num < 1 || num > index)
    {
        cout << "ID 不在上述列表中" << endl;
        pcap_freealldevs(allAdapters);
    }
    for (ptr = allAdapters, i = 0; i < num - 1; ptr = ptr->next, i++);
    if                          ((pcap_handle                          =
pcap_open_live(ptr->name,65536,PCAP_OPENFLAG_PROMISCUOUS,1000,errbuf))
== NULL)
    {
        cout << "无法打开适配器,Npcap 不支持" << endl;
        pcap_freealldevs(allAdapters);
        exit(0);
    }
    cout << "正在监听" << ptr->description << endl;
    pcap_freealldevs(allAdapters);
    int cnt = -1;
    cout << "请输入想要捕获的数据包个数:" << endl;
    cin >> cnt;
    pcap_loop(pcap_handle, cnt, epp_callback, NULL);
    cout << "解析 ip 数据包结束" << endl;
}
int main()
{
```

```
    Catch();
    return 0;
}
```