南 开 大 学

计算机与网络空间安全学院

网络技术与应用课程报告

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**第二次实验报告**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

学号：2012911

姓名：马永田

年级：2020级

专业：计算机科学与技术

2022年10月26日

1. **实验内容说明**

**（一）IP数据报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。
6. **实验准备**

**（一）了解NPcap的架构。**

Npcap是用于Windows操作系统的数据包捕获和网络分析的体系结构，由软件库和网络驱动程序组成。Npcap 基于 WinPcap 4.1.3 源码基础上开发，支持 32 位和 64 位架构，在 Windows Vista 以上版本的系统中，采用 NDIS 6 技术的 Npcap 能够比原有的 WinPcap 数据包（NDIS 5）获得更好的抓包性能，并且稳定性更好。

Npcap独具以下特点：

1.支持 NDIS 6 技术；

2.支持“只允许管理员 Administrator”访问 Npcap；

3.支持与 WinPcap 兼容或并存两种模式；

4.支持 Windows 平台的回环（Loopback）数据包采集；

5.支持 Windows 平台的回环（Loopback）数据包发送；

大多数网络应用程序都通过广泛使用的操作系统原语(例如套接字)访问网络。用这种方法很容易访问网络上的数据，因为操作系统可以外理低级细节(协议外理，数据包重组等)，并提供类似于用干读取和写入文件的界面。但是，有时候，“简便方法”不能胜任这项工作，因为某些应用程序需要直接访问网络上的数据包。也就是说，他们需要访问网络上的“原始”数据，而无需操作系统介入协议处理。

Npcap的目的是使这种对Windows应用程序的访问。

它提供以下功能:

1.捕获原始数据包，包括发往运行机器的原始数据包和与其他主机(在共享媒体上)交换的原始数据包

2.根据用户指定的规则对数据包进行过滤

3.再将其分发给应用程序

4.将原始数据包发送到网络

5.收集统计信息在网络上的流量

1. **配置实验环境并构建项目**

安装 Npcap1.71、npcap-sdk-1.13,并对Visual Studio的项目进行如下配置：

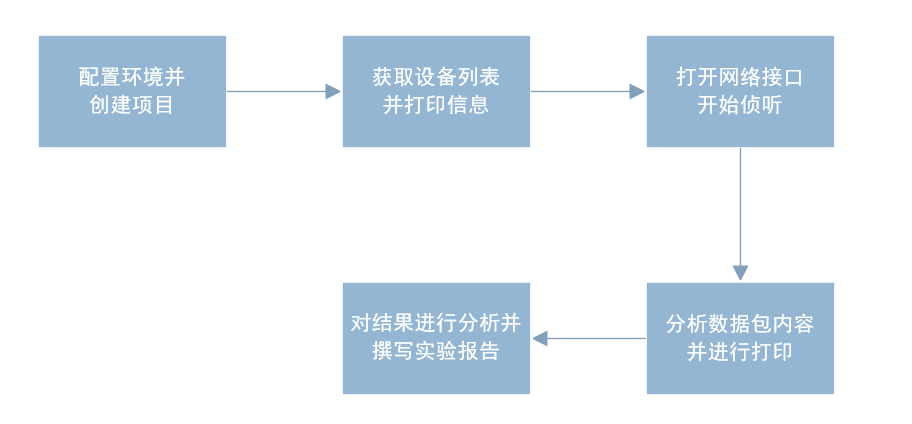
• 在 C/C++ 中添加附加包含目录：D:\npcap\Include;

• 在预处理器中添加预处理器定义：WPCAP;HAVE\_REMOTE;

• 在链接器-> 常规中添加附加库目录：D:\npcap\Lib;

• 在链接器-> 输入中添加附加依赖项：Packet.lib;wpcap.lib

1. **实验过程**
2. **项目设计思路**



1. **关键代码分析**
2. **获取设备列表并打印设备信息**

第一步首先调用 WinPcap 提供的 pcap\_findalldevs\_ex()函数来获取网络接口设备即网卡列表。 pcap\_findalldevs\_ex()函数原型为：

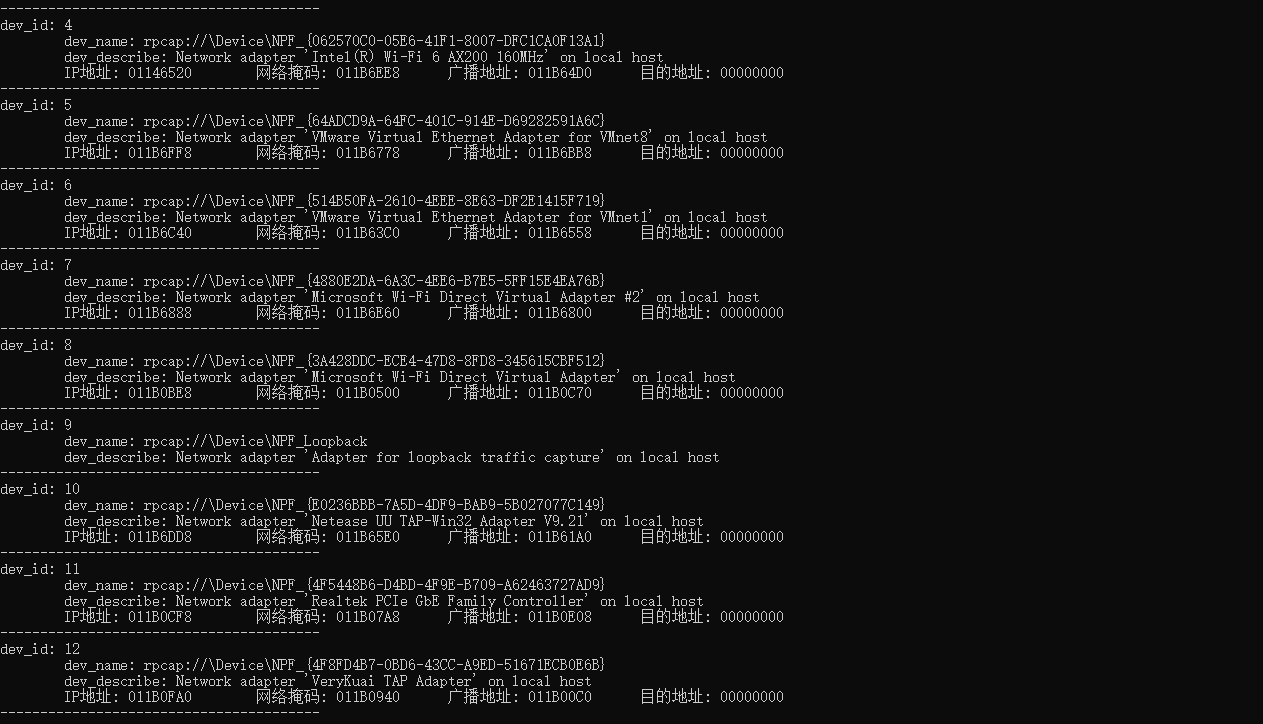
int [pcap\_findalldevs\_ex](http://www.ferrisxu.com/WinPcap/html/group__wpcapfunc.html" \l "g98f36e62c95c6ad81eaa8b2bbeb8f16e) (char \*source, struct [pcap\_rmtauth](http://www.ferrisxu.com/WinPcap/html/structpcap__rmtauth.html) \*auth, [pcap\_if\_t](http://www.ferrisxu.com/WinPcap/html/structpcap__if.html) \*\*alldevs, char \*errbuf) 返回值：0表示查找成功。-1表示查找失败

参数介绍：

1. “ source”指定是本地适配器或者远程适配器，它告诉函数必须在哪里进行查找，并且使用与pcap\_open（）相同的语法。
2. [struct](https://so.csdn.net/so/search?q=struct&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/explormt/article/details/_blank) [pcap\_rmtauth](http://www.ferrisxu.com/WinPcap/html/structpcap__rmtauth.html)  \*auth  (auth参数可以为NULL.)
3. [pcap\_if\_t](http://www.ferrisxu.com/WinPcap/html/structpcap__if.html) \*\*alldevs  该参数用于存放获取的适配器数据。如果查找失败，alldevs的值为NULL.
4. char \*errbuf 该参数存放查找失败的信息。

|  |
| --- |
| pcap\_if\_t\* alldevs;  pcap\_if\_t\* dev;  pcap\_addr\_t\* a;  char errbuf[PCAP\_ERRBUF\_SIZE];  //获取设备列表  if (pcap\_findalldevs\_ex(PCAP\_SRC\_IF\_STRING, NULL, &alldevs, errbuf) != -1) {  cout << "获取设备列表成功" << endl;  }  else {  cout << "获取设备列表失败" << endl;  return 0;  }  //显示获取的设备列表  int devid = 0;  for (dev = alldevs; dev != NULL; dev = dev->next) {  cout << "dev\_id: " << ++devid << endl;  cout << " dev\_name: " << dev->name << endl;  cout << " dev\_describe: " << dev->description << endl;  //获取该网络接口设备的IP地址信息  for(a = dev->addresses; a !=NULL; a = a->next){  //判断该地址是否为IP地址  if (a->addr->sa\_family == AF\_INET) {  cout << " IP地址: " << a->addr << endl;  cout << " 网络掩码: " << a->netmask << endl;  cout << " 广播地址: " << a->broadaddr << endl;  cout << " 目的地址: " << a->dstaddr << endl;  }  }  cout << "----------------------------------------" << endl;  } |

运行后结果如下图，可以看到获取的网卡的相关信息。



1. **打开网络接口并捕获数据包**

得到网络接口设备列表（即 alldevs）之后，, 需要打开网络接口卡并对其网络流量进行监听，打开网络接口设备需要使用 pcap\_open() 函数。之后使用pcap\_next\_ex()函数捕获数据包。

pcap\_open() 函数定义如下：

pcap\_t\* pcap\_open(

const char\* source,

int snaplen,

int flags,

int read\_timeout,

struct pcap\_rmtauth\* auth,

char\* errbuf

)

**参数介绍:**

1.source：以/0终止的字符串，其中包含要打开的源名称。

2.snaplen: 必须保留的包的长度。对于由过滤器接收的每个数据包，只有第一个“snaplen”字节存储在缓冲区中并传递给用户应用程序。

3.flags: 保留捕获数据包可能需要的几个标志。

4.read\_timeout: 以毫秒为单位读取超时。

5.auth: 指向“结构pcap\_rmtauth”的指针，它保留在远程机器上验证用户所需的信息。如果这不是远程捕获，则该指针可以设置为NULL。

6.errbuf: 指向用户分配的缓冲区的指针，该缓冲区将在该函数失败的情况下包含错误。

pcap\_next\_ex()函数定义如下

pcap\_next\_ex(

pcap\_t \* p, struct p

cap\_pkthdr\*\* pkt\_header,

const u\_char \* \*pkt\_data

)

**功能:** 从interface或离线记录文件获取一个报文

**参数:**

1.p: 已打开的捕捉实例的描述符

2.pkt\_header : 报文头

3.pkt\_data : 报文内容

**返回值:**

1 : 成功

0 : 获取报文超时

- 1 : 发生错误

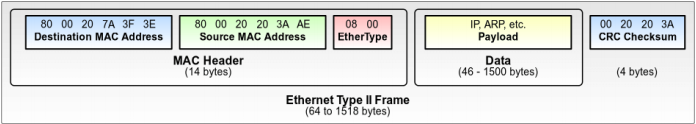
- 2 : 获取到离线记录文件的最后一个报文

|  |
| --- |
| cout << "开始侦听:" << endl;  devid = 0;  for (pcap\_if\_t\* dev = alldevs; dev != NULL; dev = dev->next) {  //打开网络接口  pcap\_t\* handle = pcap\_open(dev->name, 655340, PCAP\_OPENFLAG\_PROMISCUOUS, 2000, NULL, errbuf);  //数据包头  pcap\_pkthdr\* pkt\_header;  //数据包  const u\_char\* pkt\_data;  cout << "dev\_id: " << ++devid << endl;  //捕获数据包  int retvalue = pcap\_next\_ex(handle, &pkt\_header, &pkt\_data);  cout << " 侦听长度: " << pkt\_header->len << endl;  PrintFrameHeader(pkt\_data);  PrintIPHeader(pkt\_data);  if (retvalue == 0) {  cout << " 获取报文超时" << endl;  }  cout << "----------------------------------------" << endl;  } |

1. **分析数据包结构并进行打印**

**相关结构体定义**

在以太网中，数据通信的基本单位是以太网帧 (frame)，由头部 (header )、数据 (data) 以及校验和 (checksum) 三部分构成：



以太网帧头部：

• 目的地址，用于标记数据由哪台机器接收，6 字节

• 源地址，用于标记数据由哪台机器发送，6 字节

• 类型，用于标记数据该如何处处理，2 字节

网络中传输的数据包是经过封装的，每一次封装都会增加相应的首部，由于 Npcap 在数据链路层捕获数据包，因此，在以太网中利用 pcap\_next\_ex 获得的数据都包含以太网帧头的信息，并且由于利用 pcap\_next\_ex 捕获到的数据包保存在一个无结构的缓冲区中，因此在实际编程过程中通常需要定义一些有关首部的数据结构。根据上述对以太网帧的介绍，定义如下结构体：

|  |
| --- |
| #pragma pack (1)  //进入字 节对齐方式  typedef struct FrameHeader\_t {  BYTE DesMAC[6];  // 目的地址  BYTE SrcMAC[6];  //源地址  WORD FrameType;  //帧类型  }FrameHeader\_t;  typedef struct IPHeader\_t {  //IP首部  BYTE Ver\_HLen;  BYTE TOS;  WORD TotalLen;  WORD ID;  WORD Flag\_Segment;  BYTE TTL;  BYTE Protocol;  WORD Checksum;  ULONG SrcIP;  ULONG DstIP;  } IPHeader\_t;  typedef struct Data\_t {  //包含帧首部和IP首部的数据包  FrameHeader\_t FrameHeader;  IPHeader\_t IPHeader;  } Data\_t;  #pragma pack() //恢复缺省对齐方式 |

**打印捕获数据包信息：**

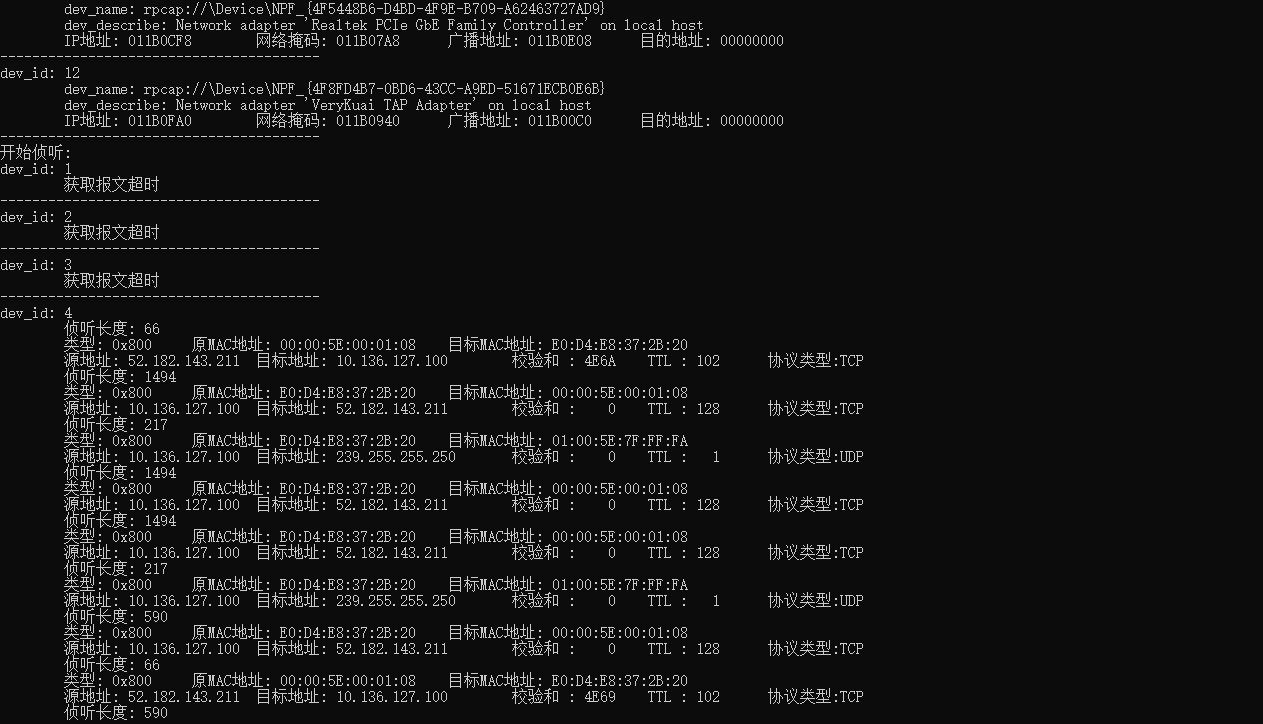
该部分封装在void PrintFrameHeader(const u\_char\* packetData)函数与void PrintIPHeader(const u\_char\* packetData)函数中，用于打印捕获到以太网帧头部的信息。

PrintFrameHeader部分打印以太网类型、原始MAC地址、目标MAC地址， 首先初始化上一步定义的 FrameHeader\_t，传入捕获到的 packetData，将 16 位数由网络字节顺序转换为主机字节顺序后，打印以太网类型。

PrintIPHeader部分打印源IP地址、目的IP地址、校验和以及协议类型，+14 跳过数据链路层后进行类型转换得到IPHeader\_t类型指针，之后将二进制存储的IP地址转换为我们常见的形式，对相关内容进行打印，代码如下：

|  |
| --- |
| void PrintFrameHeader(const u\_char\* packetData){  struct FrameHeader\_t\* protocol;  protocol = (struct FrameHeader\_t\*)packetData;  u\_short ether\_type = ntohs(protocol->FrameType); // 以太网类型  u\_char\* ether\_src = protocol->SrcMAC; // 以太网原始MAC地址  u\_char\* ether\_dst = protocol->DesMAC; // 以太网目标MAC地址  printf(" 类型: 0x%x \t", ether\_type);  printf("原MAC地址: %02X:%02X:%02X:%02X:%02X:%02X \t",  ether\_src[0], ether\_src[1], ether\_src[2], ether\_src[3], ether\_src[4], ether\_src[5]);  printf("目标MAC地址: %02X:%02X:%02X:%02X:%02X:%02X \n",  ether\_dst[0], ether\_dst[1], ether\_dst[2], ether\_dst[3], ether\_dst[4], ether\_dst[5]);  }  void PrintIPHeader(const u\_char\* packetData) {  struct IPHeader\_t\* ip\_protocol;  // +14 跳过数据链路层  ip\_protocol = (struct IPHeader\_t\*)(packetData + 14);  SOCKADDR\_IN Src\_Addr, Dst\_Addr = { 0 };  u\_short check\_sum = ntohs(ip\_protocol->Checksum);  int ttl = ip\_protocol->TTL;  int proto = ip\_protocol->Protocol;  Src\_Addr.sin\_addr.s\_addr = ip\_protocol->SrcIP;  Dst\_Addr.sin\_addr.s\_addr = ip\_protocol->DstIP;  char buff1[17];  ::inet\_ntop(AF\_INET, (const void\*)&Src\_Addr.sin\_addr, buff1, 17);  printf(" 源地址: %s \t", buff1);  char buff2[17];  ::inet\_ntop(AF\_INET, (const void\*)&Dst\_Addr.sin\_addr, buff2, 17);  printf("目标地址: %s \t", buff2);  printf("校验和 :%5X \t TTL :%4d \t", check\_sum, ttl);  printf("协议类型:");  switch (ip\_protocol->Protocol)  {  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 89: printf("OSPF \n"); break;  default: printf("None \n"); break;  }  } |

**4.实验结果展示**



1. **特殊现象分析**

最初在实验中发现，程序运行到pcap\_open函数时会报错，提示0xC0000005：写入位置0x00000000时发生访问冲突，此外显示无法打开wpcap.lib，各种调试无果，最终选择将npcap的版本从1.6.3更新到最新的1.7.1后问题解决。