南开大学

计算机学院

网络技术与应用课程报告

第2次实验报告

学号：2010519
姓名：卢麒萱
年级：2020
专业：计算机科学与技术

第1节 实验内容说明

（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。

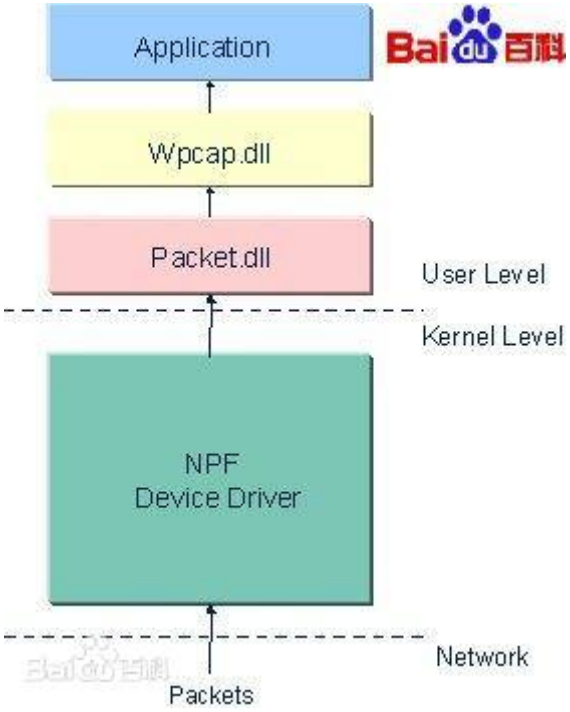
（4）捕获的数据报不要求硬盘存储，但应以简单明了的方式在屏幕上显示。必显字段包括源MAC地址、目的MAC地址和类型/长度字段的值。

（5）编写的程序应结构清晰，具有较好的可读性。

第2节 实验准备

winpcap简介

winpcap(windows packet capture)是windows平台下一个免费，公共的网络访问系统。它用于windows系统下的直接的网络编程。Winpcap是针对Win32平台上的抓包和网络分析的一个架构。它包括一个核心态的包过滤器，一个底层的动态链接库（packet.dll）和一个高层的不依赖于系统的库（wpcap.dll）。主要组成部分如下：



它有如下几个**功能**：

1. 捕获原始数据包，包括在共享网络上各主机发送/接收的以及相互之间交换的数据；
2. 在数据包发往应用程序之前，按照自定义的规则将某些特殊的数据包过滤掉；
3. 在网络上发送原始的数据包；
4. 收集网络通信过程中的统计信息。

首先，为了访问网络上传输的原始数据，一个捕获系统需要绕过操作系统的协议栈。 这需要一部分程序运行于操作系统的内核中，来与网络接口驱动直接交互。该部分与操作系统密切相关，WinPcap的解决方案是实现一个叫做Netgroup Packet Filter（NPF）的设备驱动程序，并对Windows 95、Windows 98、Windows ME、Windows NT 4、Windows 2000 与Windows XP等不同操作系统提供不同版本的驱动程序。这些驱动程序提供了数据包捕获与发送的基本特性，同时也提供诸如一个可编程的过滤系统与一个监控引擎之类的更高级特性。第一个特性可用于限制一个捕获会话，只捕获特定的网络数据包（比如，可以只捕获一个特定主机生成的ftp数据包）。第二个特性提供了一个强大但简单的方式，来获取网络流量的统计信息 (比如，可以获取网络负载或两个主机间所交换数据的数量)。

其次，捕获系统必须导出一个接口，使得用户层应用程序可使用内核驱动所提供的特性。WinPcap提供两个不同的库：packet.dll与wpcap.dll。

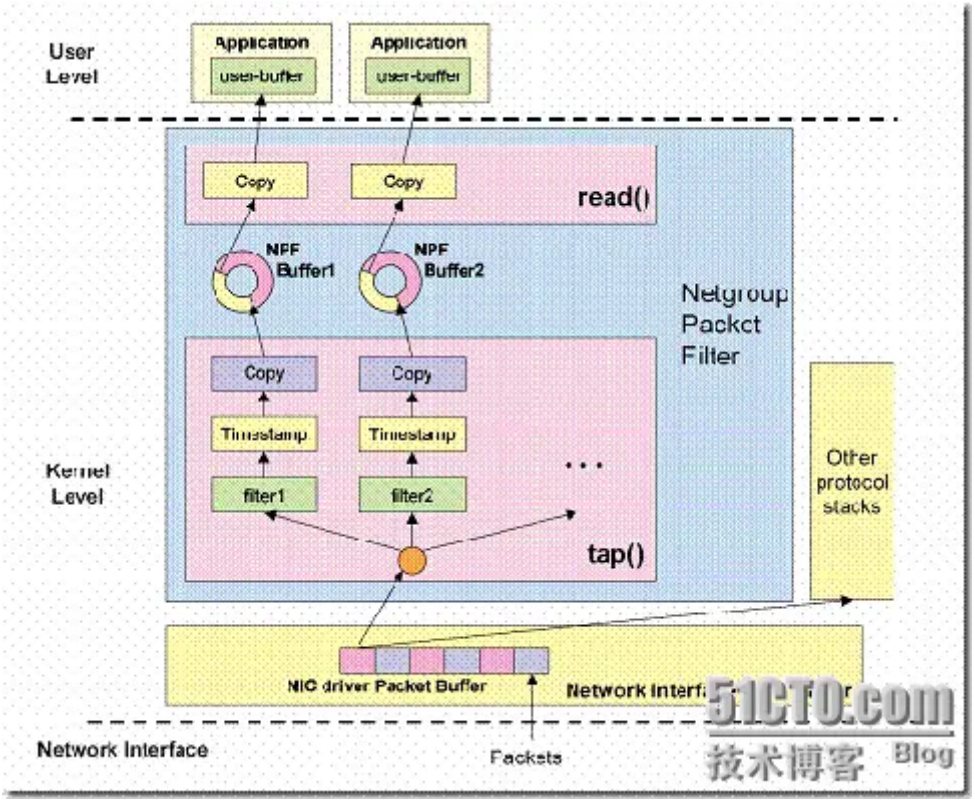
第一个库提供一个底层的API，可用来直接访问驱动程序的函数，提供一个独立于微软的不同操作系统的编程接口。

第二个库导出了更强大的、更高层的捕获函数接口，并提供与UNIX捕获库libpcap的兼容性。

这些函数使得数据包的捕获能独立于底层网络硬件与操作系统。

数据包捕获的基本过程

WinPcap从网络上捕获一个数据包，然后递送给应用程序，所调用的组件如图3所示。



Npcap简介

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 还独具以下**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；

环境配置

安装`npcap`，新建`CMake`工程，在`CMakeLists.txt`文件中添加如下代码：

```
cmake_minimum_required(VERSION 3.12)
project(main LANGUAGES C CXX)
set(CMAKE_INCLUDE_CURRENT_DIR on)
set(CMAKE_C_STANDARD 11)
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_BUILD_TYPE Release)
include_directories(inc)
link_directories(lib)
add_compile_definitions(UNICODE _UNICODE)
add_executable(${PROJECT_NAME} src/main.c)
target_link_libraries(${PROJECT_NAME} x64/wpcap ws2_32)
```

将库文件和头文件添加到工程。

第3节 实验过程

基本函数

- 注：`pcap_if`和`pcap_if_t`相同。

```
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 */
};
```

该结构体有五个元素。第一是一个`pcap_if`的链表指向下一个设备接口；第二个是设备的实际的名字，这个名字是机器能识别的名字，供`pcap_open_live()`调用；第三个是设备的文本描述符，这个描述符就是人们能够识别的文本符号；第四个是一个地址指针，指向的是一系列接口的第一个指针；第五个是一个标志位，表示目前这个标志位主要是不是`loopback`设备。

用户定义了两个类型：

```
struct pcap_addr {
  struct pcap_addr* next;
  struct sockaddr*  addr;      /* address */
  struct sockaddr*  netmask;   /* netmask for that address */
  struct sockaddr*  broadaddr; /* broadcast address for that address */
  struct sockaddr*  dstaddr;   /* P2P destination address for that address */
};
```

该结构体表示接口地址。

`typedef struct pcap pcap_t;`：一个已打开的捕获实例描述符，这个结构体对用户来说是不透明的，他提供wpcap.dll的函数来维护他的内容。

```
struct pcap_pkthdr {
 struct timeval ts; /* time stamp */
 bpf_u_int32 caplen; /* length of portion present */
 bpf_u_int32 len; /* length this packet (off wire) */
};
```

该结构体表示一个数据包在堆文件中的文件头，包括时间戳，目前部分的长度和数据包的长度。

获取设备列表

通常，编写基于`WinPcap`应用程序的第一件事情，就是获得已连接的网络适配器列表。`WinPcap`提供了`pcap_findalldevs()`函数来实现这个功能：这个函数返回一个`pcap_if_t`结构的链表， 每个这样的结构都包含了一个适配器的详细信息。数据域`name`和`description`表示一个适配器名称和一个可以让人们理解的描述。

```
static char errbuf[PCAP_ERRBUF_SIZE];
pcap_if_t*  alldevs;
/* Retrieve the device list */
if (pcap_findalldevs(&alldevs, errbuf) == -1) {
```

```
  fprintf(stderr, "Error in pcap_findalldevs: %s\n", errbuf);
  return -1;
}

pcap_if_t* d;
int i = 0;
/* Print the list */
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;
}
```

选择设备并跳转打开设备

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

if (inum < 1 || inum > i) {
  printf("\nInterface number out of range.\n");
  /* Free the device list */
  pcap_freealldevs(alldevs);
  return -1;
}

/* Jump to the selected adapter */
for (d = alldevs, i = 0; i < inum - 1; d = d->next, i++)
  ;

pcap_t* adhandle;
/* Open the adapter */
if ((adhandle =
       pcap_open_live(d->name,  // name of the device
                      65536,    // portion of the packet to capture.
                                // 65536 grants that the whole packet will
be
                                // captured on all the MACs.
                      1,      // promiscuous mode (nonzero means
promiscuous)
                      1000,   // read timeout
                      errbuf  // error buffer
                      ))
    == NULL) {
  fprintf(stderr,
          "\nUnable to open the adapter. %s is not supported by WinPcap\n",
          d->name);
  /* Free the device list */
  pcap_freealldevs(alldevs);
  return -1;
```

```
}
printf("\nlistening on %s...\n", d->description);
/* At this point, we don't need any more the device list. Free it */
pcap_freealldevs(alldevs);
```

输出以太网帧各字段

```
int                 res;
struct pcap_pkthdr* header;
const u_char*       pkt_data;
/* Retrieve the packets */
while ((res = pcap_next_ex(adhandle, &header, &pkt_data)) >= 0) { //
pkt_data保存以太网帧部分信息
  if (res == 0) continue; /* Timeout elapsed */

  /* convert the timestamp to readable format */
  time_t     local_tv_sec = header->ts.tv_sec;//捕获数据包的时间
  struct tm* ltime        = localtime(&local_tv_sec);//格式转换
  char       timestr[16];
  strftime(timestr, sizeof timestr, "%H:%M:%S", ltime);//格式化时间

  printf("[Timestamp] %s,%.6d\n", timestr, header->ts.tv_usec);//打印时间
  printf("[Capture length] %u\n", header->caplen);//ncap捕获的长度
  printf("[Total length] %u\n", header->len);//数据包总长度
  const etherheader_t* eh = pkt_data;
  printf("[Source MAC] %x:%x:%x:%x:%x:%x\n", eh->saddr.byte[0],
         eh->saddr.byte[1], eh->saddr.byte[2], eh->saddr.byte[3],
         eh->saddr.byte[4], eh->saddr.byte[5]);//saddr.byte储存mac源地址
  printf("[Destination MAC] %x:%x:%x:%x:%x:%x\n", eh->daddr.byte[0],
         eh->daddr.byte[1], eh->daddr.byte[2], eh->daddr.byte[3],
         eh->daddr.byte[4], eh->daddr.byte[5]);//daddr.byte储存mac目的地址
  printf("[EtherType] 0x%04hx\n", ntohs(eh->type));//type为以太类型
  puts("--------");
}
```

运行后部分输出：

```
PS C:\Users\Administrator\Desktop\course3-1\internet_technique\ex2\winpcap-ether-cap> .\build\main.exe
1. \Device\NPF_{1F44D925-B944-48C9-8C90-FCEC23F0355A} (Microsoft)
2. \Device\NPF_{BCC5DEBD-A82B-4103-86EE-58BF3A73D744} (VMware Virtual Ethernet Adapter)
3. \Device\NPF_{6A756ECE-FEE3-4D5E-92A8-8C90CB0B4A3C} (VMware Virtual Ethernet Adapter)
4. \Device\NPF_{E6BF2888-8F80-4712-B2A5-E40842E007B3} (Microsoft)
5. \Device\NPF_{C9ED2896-7DD5-4545-92B4-88E973F22BA9} (Microsoft)
6. \Device\NPF_{D36E40BC-CFD5-4BB6-AC94-FAAB358F2215} (Microsoft)
7. \Device\NPF_{A1034C70-14F2-4AA0-88C0-1885AD40FE3A} (Netease UU TAP-Win32 Adapter V9.21)
8. \Device\NPF_{70651D16-25B4-4B0B-A52B-983072C05C86} (Realtek PCIe GbE Family Controller)
Enter the interface number (1-8):6

listening on Microsoft...
[Timestamp] 20:25:28,811828
[Capture length] 324
[Total length] 324
[Source MAC] 0:0:5e:0:1:8
[Destination MAC] 64:6e:e0:fc:3c:ee
[EtherType] 0x0800
--------
[Timestamp] 20:25:28,811921
[Capture length] 66
[Total length] 66
[Source MAC] 64:6e:e0:fc:3c:ee
[Destination MAC] 0:0:5e:0:1:8
[EtherType] 0x0800
--------
[Timestamp] 20:25:30,109620
[Capture length] 55
[Total length] 55
[Source MAC] 64:6e:e0:fc:3c:ee
[Destination MAC] 0:0:5e:0:1:8
[EtherType] 0x0800
--------
```