网络程序设计

WinPcap编程

1. 什么是WinPcap？

当应用程序需要访问原始数据包，即没有被操作系统利用网络协议处理过的数据包时，**socket**无法满足需要，**WinPcap**为**Win32**应用程序提供这种访问方式。

- **WinPcap**提供了以下功能
  - 捕获原始数据包，无论它是发往本机的，还是在其他设备（共享媒介）上交互的
  - 在数据包递交给某应用程序前，根据用户指定的规则过滤数据包
  - 将原始数据包通过网络发送出去
  - 收集并统计网络流量信息

1. 什么是WinPcap？

- 基于**WinPcap**的典型应用
  - 网络与协议分析器 **(network and protocol analyzers)**
  - 网络监视器 **(network monitors)**
  - 网络流量记录器 **(traffic loggers)**
  - 网络流量发生器 **(traffic generators)**
  - 用户级网桥及路由 **(user-level bridges and routers)**
  - 网络入侵检测系统 **(network intrusion detection systems (NIDS))**
  - 网络扫描器 **(network scanners)**
  - 安全工具 **(security tools)**

1. 什么是WinPcap？

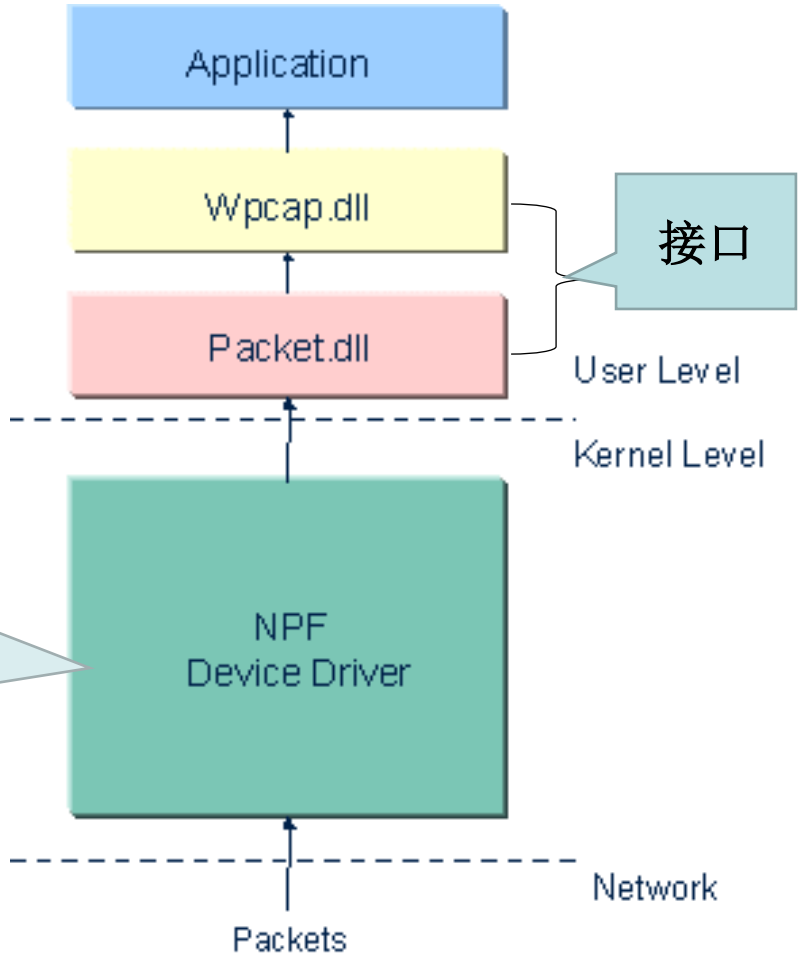
- 什么是**WinPcap**做不到的？

  WinPcap不能阻止、过滤或操纵同一机器上的其他应用程序的通讯：它仅仅能简单地“监视”在网络上传输的数据包。所以，它不能提供以下支持：

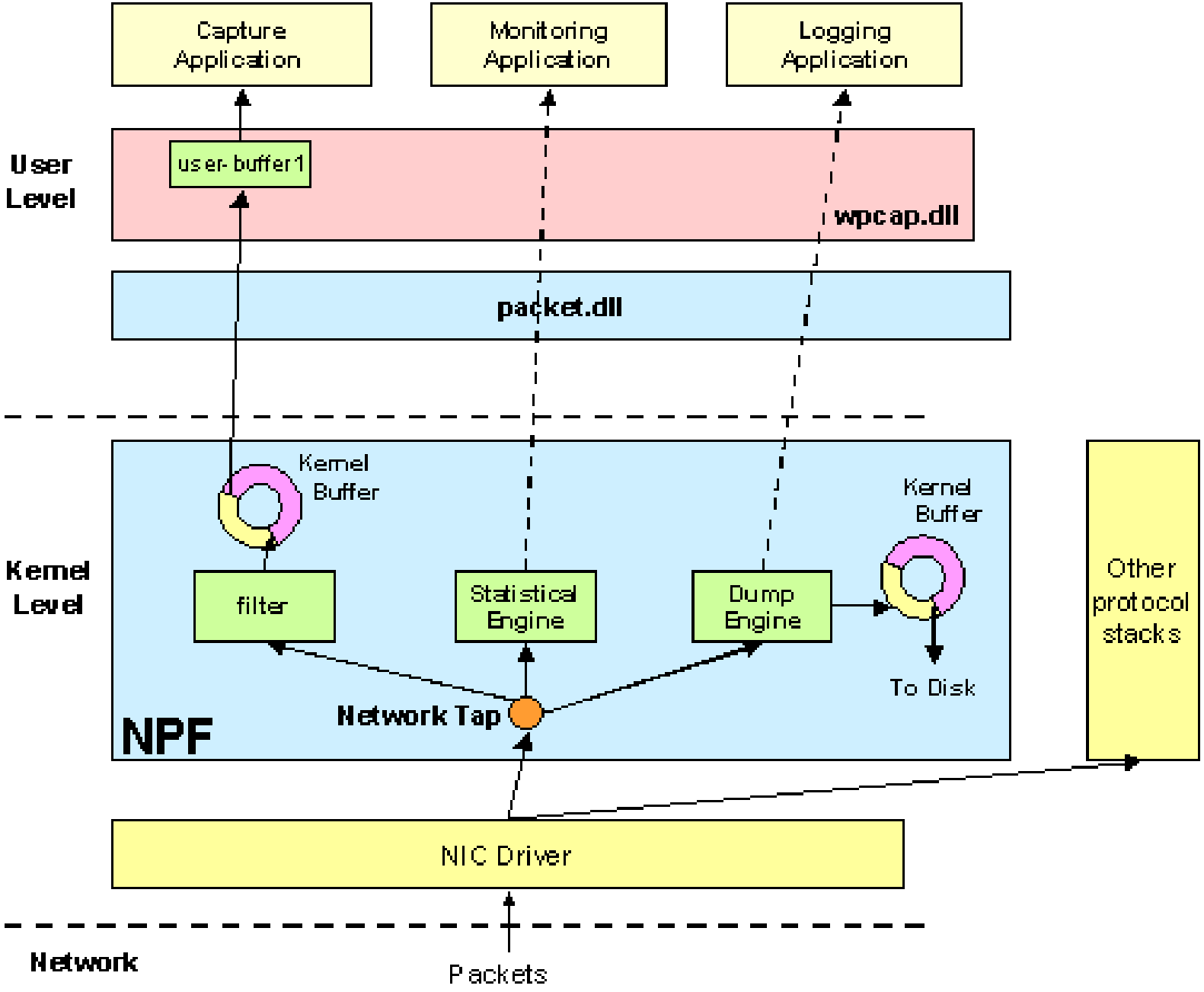
  – 网络流量控制
  – 服务质量调度
  – 个人防火墙

3.1 WinPcap组成

WinPcap is an architecture for packet capture and network analysis for the Win32 platforms. It includes a kernel-level packet filter(NPF), a low-level dynamic link library (packet.dll), and a high-level and system-independent library (wpcap.dll).



设备驱动
---数据捕获
---数据发送
---可编程的过滤系统
---监听引擎
---……


Capture Application
Monitoring Application
Logging Application
User Level
user-buffer1
wpcap.dll
packet.dll
Kernel Level
Kernel Buffer
Kernel Buffer
filter
Statistical Engine
Dump Engine
To Disk
Other protocol stacks
NPF
Network Tap
NIC Driver
Network
Packets

WinPcap体系结构

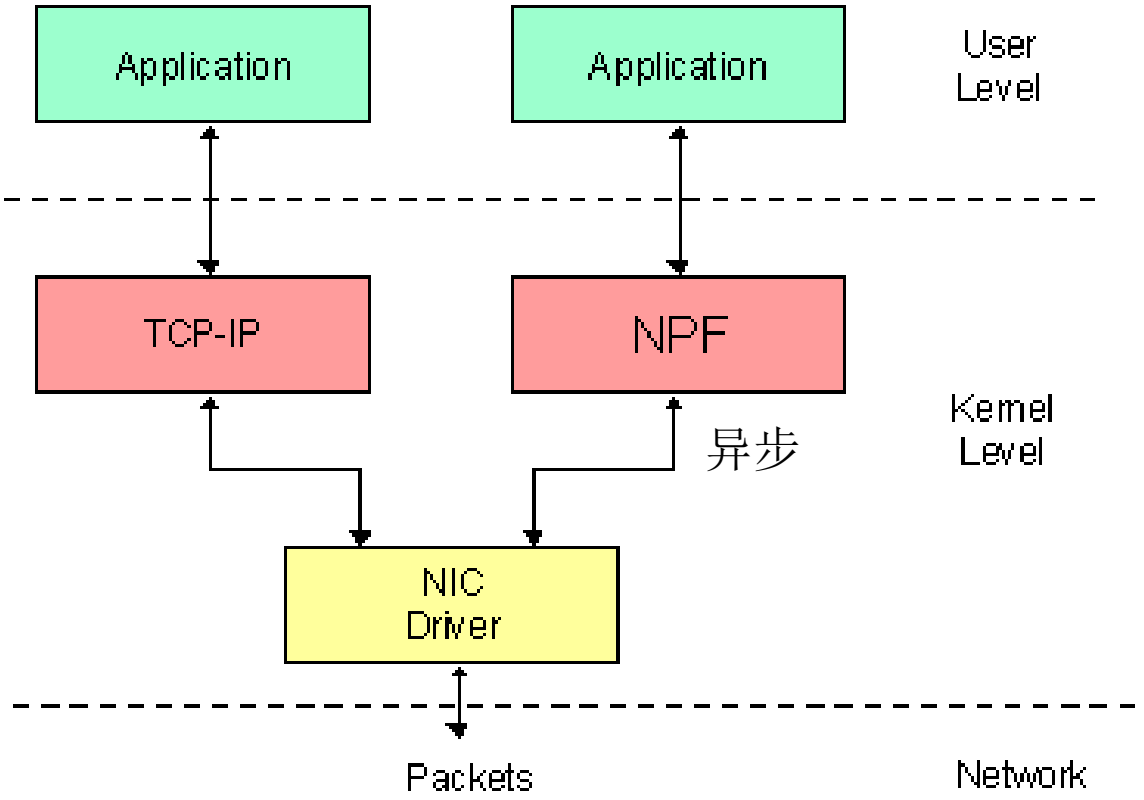
- 网络：网络中传输的数据帧
- 核心层：NPF(Netgroup Packet Filter)模块、NIC(Network Interface Card)驱动程序
- 用户层：Packet.dll, wpacp.dll

网络驱动程序接口规范

- NDIS(Network Driver Interface Specification, 网络驱动程序接口规范)是定义网络接口卡和协议驱动(TCP/IP实现)之间的通信标准。
- NDIS位于网卡和协议层之间，通过一套基元指令为上层的协议驱动提供服务，同时屏蔽了下层各种网卡技术上的差别。
- NDIS支持以下三种类型的网络驱动程序
  - 网络驱动程序
  - 中间层驱动程序
  - 传输驱动程序或协议驱动程序（比如NPF）

3.1 WinPcap组成

- NPF的位置
  - NPF is implemented as a protocol driver

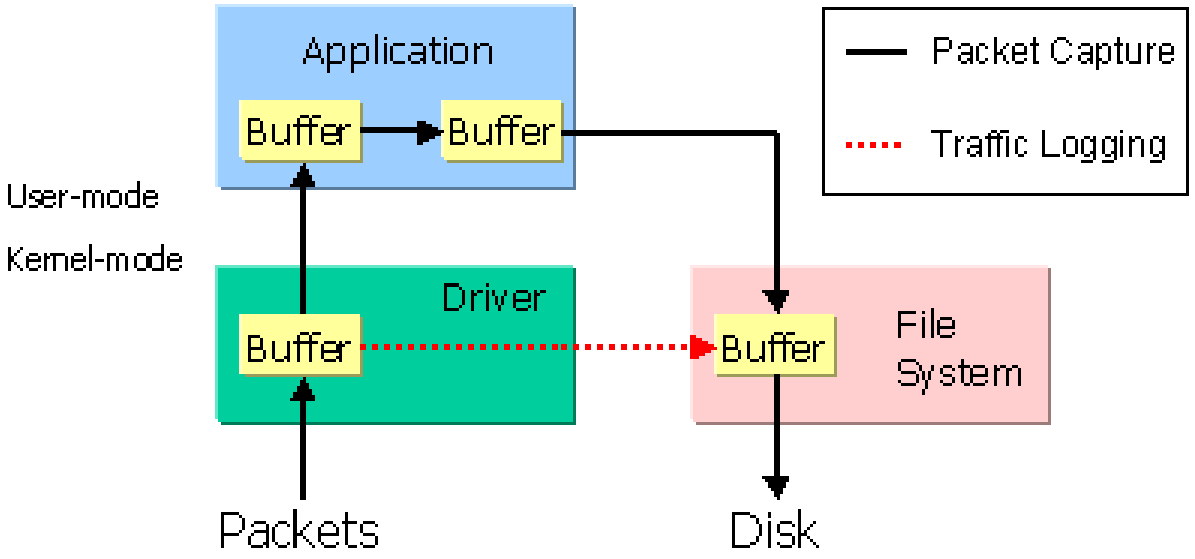


NPF的结构

- 捕获数据帧是NPF最重要的操作。数据捕获过程依赖以下4个主要组件
- 1)数据帧过滤器
- 2)虚拟处理器:可以执行伪汇编书写的用户级过滤程序
- 3)核心缓冲区
- 4)网络分流器

NPF功能

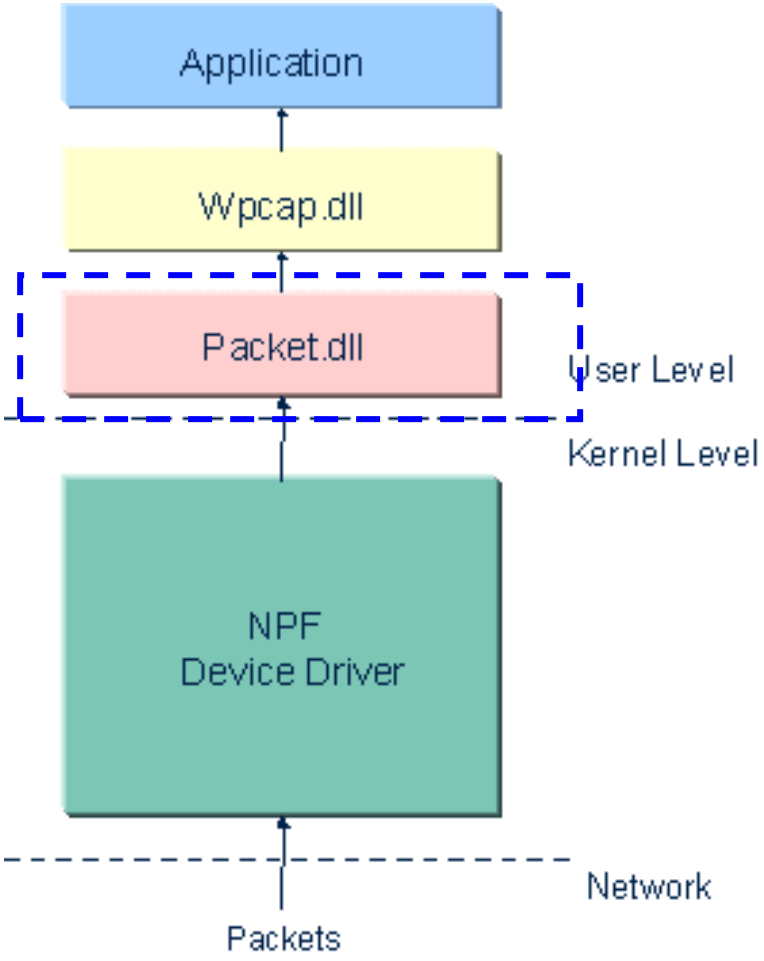
- **NPF功能:**
  - 数据捕获
  - 流量监测
  - 数据发送
  - **dump to disk**



Packet.dll

- **packet.dll (Packet Driver API)**

  提供了一个底层的API访问接口，可以直接访问网卡，为win32平台提供了一个公共的接口。



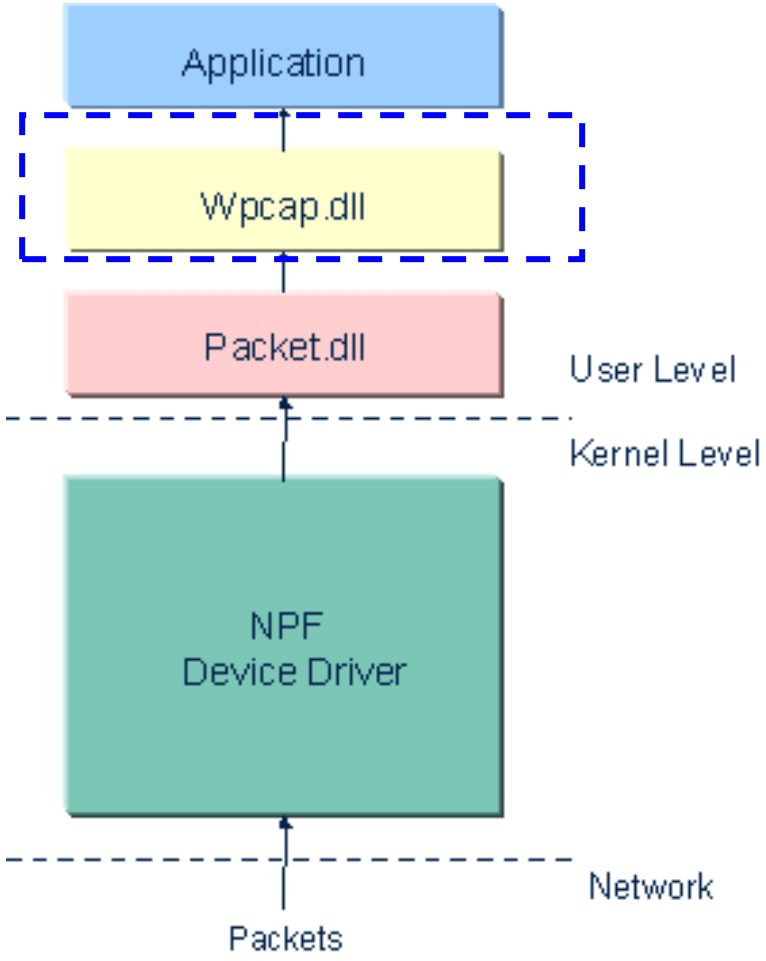
Packet.dll

- 功能
  - 安装，启动和停止NPF设备驱动
  - 从NPF驱动接收数据包
  - 通过NPF驱动发送数据包
  - 获取可用的网络适配器列表
  - 获取适配器的不同信息，比如设备描述，地址列表和掩码
  - 查询并设置一个低层的适配器参数
- two versions of packet.dll
  - the first one runs under Windows 95/98/ME
  - the second one is for Windows NT/2000/XP.
- 说明：
  - 以系统独立的方式访问网卡，开发的应用程序在不同的Windows系统上运行不需重新编译（向后兼容）
  - 透明处理NPF驱动

wpcap.dll

- **Wpcap.dll(wpcap,libpcap)**

  不依赖于操作系统，与libpcap兼容，提供了更加高层、抽象的函数。



wpcap.dll

- 功能
  - 获取网络适配器列表
  - 获取网络适配器的不同信息，比如网卡描述和地址的列表
  - 捕获数据包
  - 发送数据
  - 有效保存数据包到磁盘
  - 创建一个数据包过滤器把它们应用到数据捕获中去

WinPcap环境配置

- VS2019 环境下的配置
  - 1. 到http://www.winpcap.org/devel.htm 下载安装包 和 Developer's Pack.
  - 2.安装驱动
  - 3.打开vs平台，设置环境；

安装WinPcap,下载解压WinPcap开发包

WinPcap: Developer Resources

The latest stable WinPcap version is 4.1.3

At the moment there is no development version of WinPcap. For the list of changes, refer to the changelog.

Download
Developer Pack

Download WinPcap 4.1.2 Developer's Pack

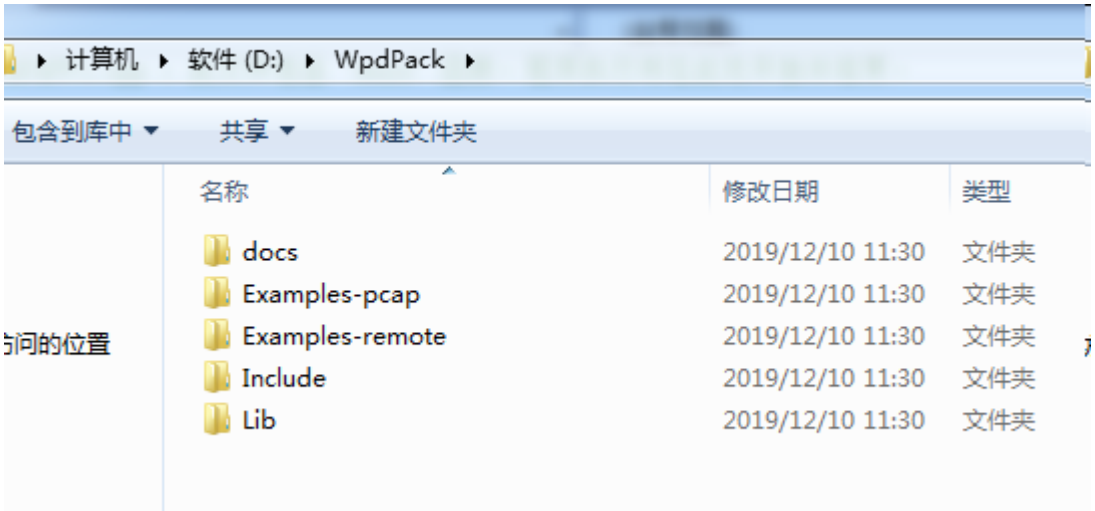
MD5 Checksum: bae2236af062b0900ad1416b2c4878b9
SHA1 Checksum: f5c80885bd48f07f41833d0f65bf85da1ef1727a

This ZIP compressed file contains all the files needed
to create WinPcap-based applications: libraries, include files,
documentation and a complete set of example programs.

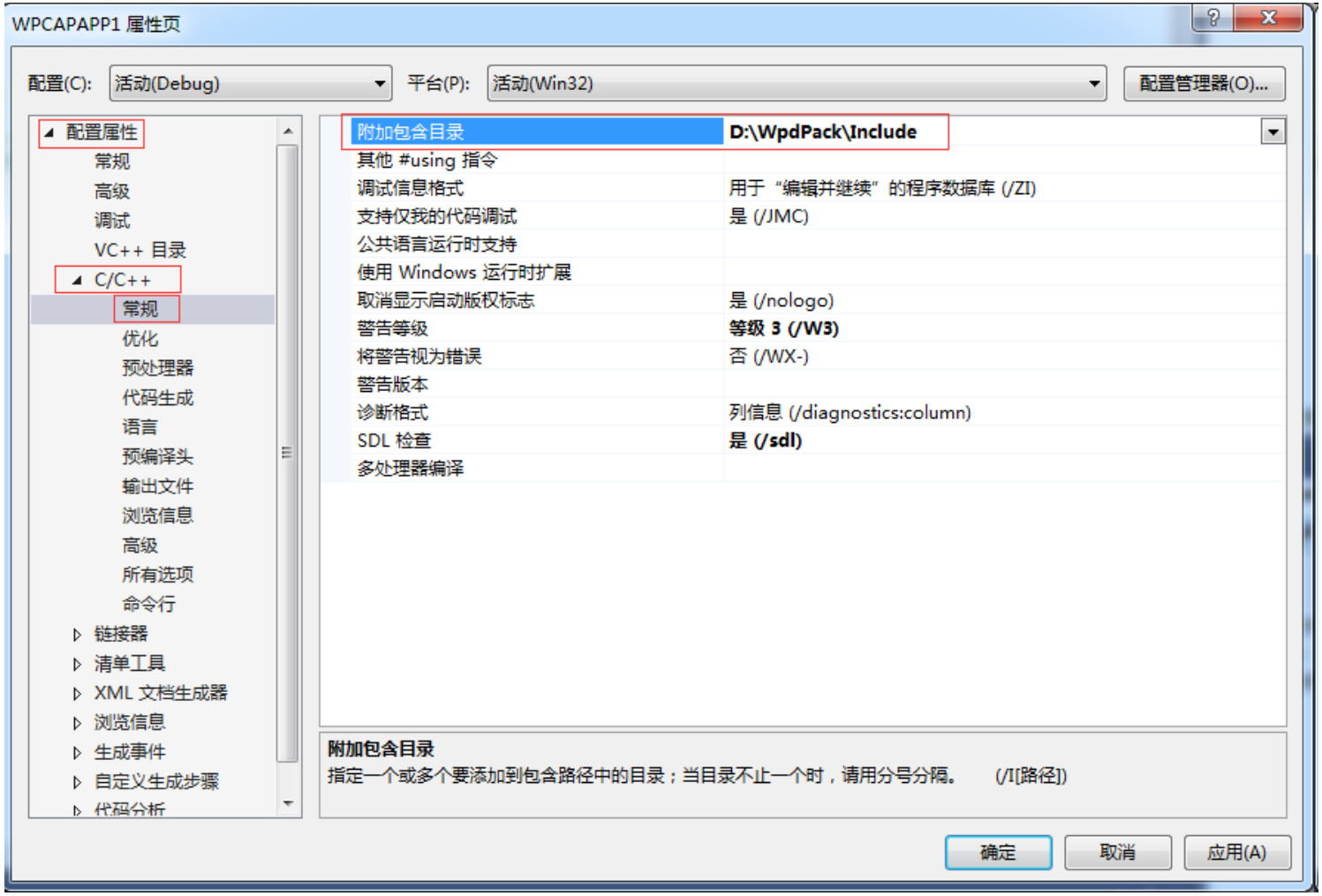
Instructions

1. download the ZIP archive containing the developer's pack
2. uncompress it to the desired folder

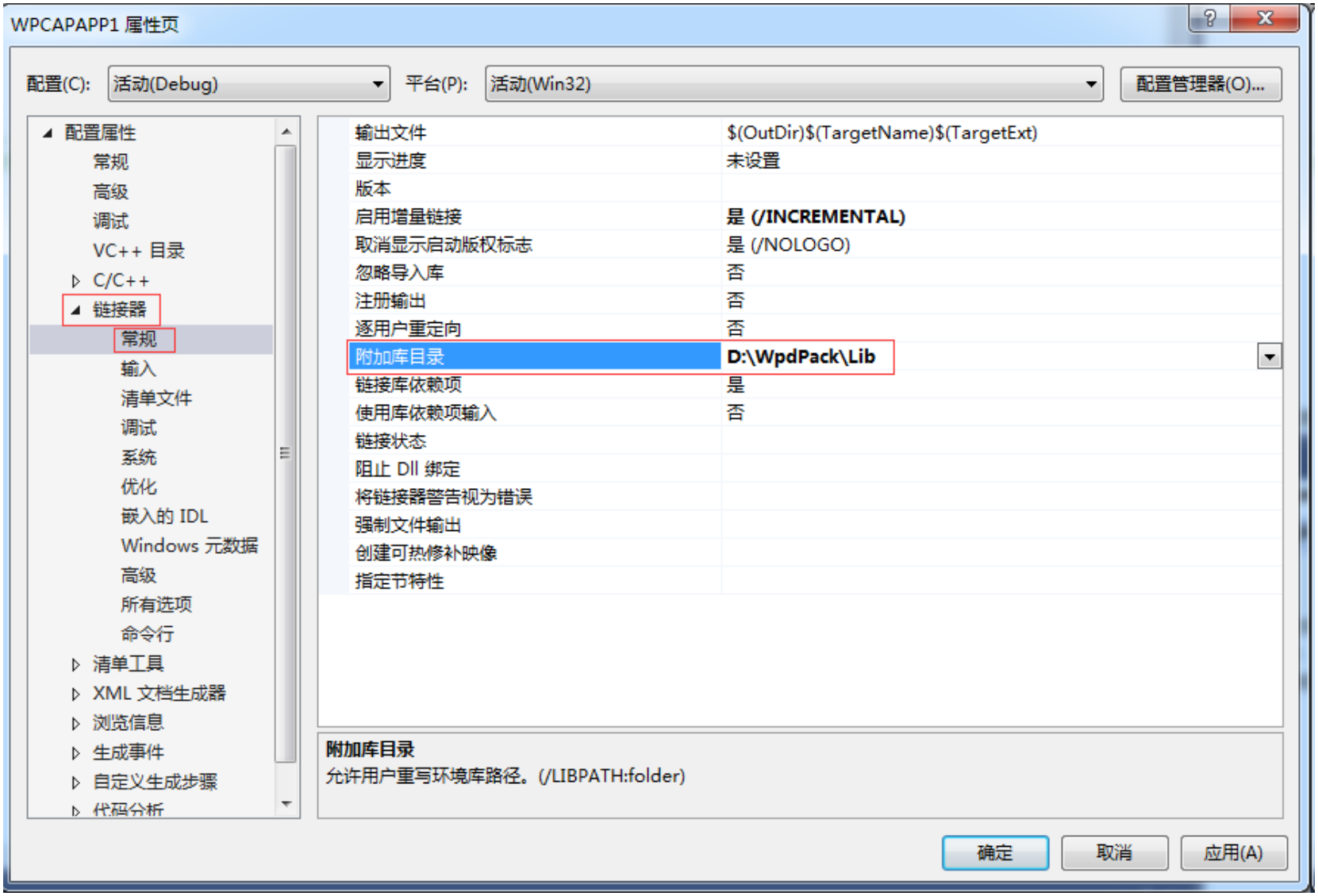
NOTE: there is no Developer's package specific for WinPcap 4.1.3. The current 4.1.2 package is compatible with WinPcap 4.1.3.



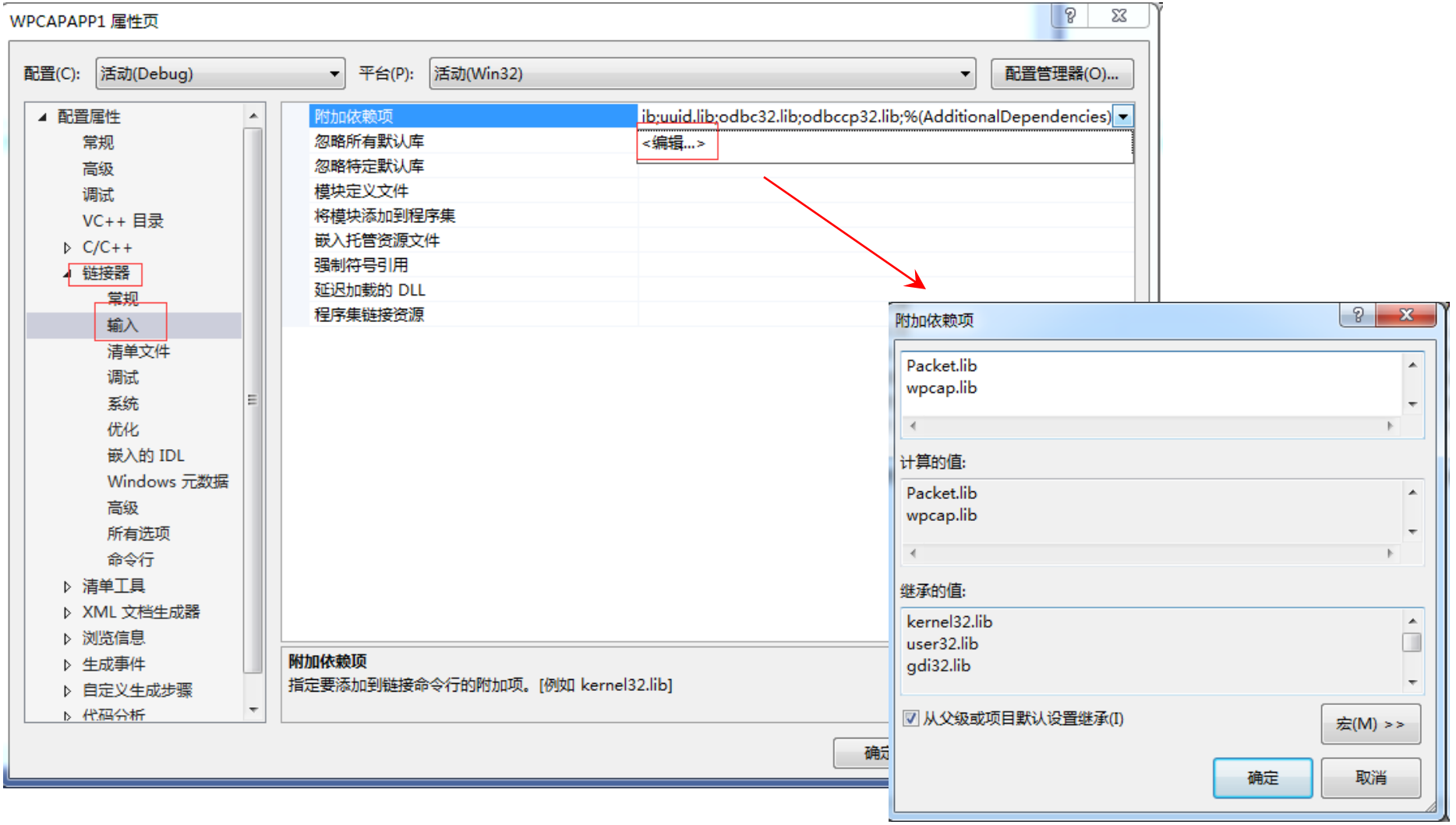
附加WinPcap的Include目录



附加WinPcap的Lib目录



引入常用的库文件


WPCAPAPP1 属性页
配置(C): 活动(Debug)
平台(P): 活动(Win32)
配置管理器(O)...
配置属性
常规
高级
调试
VC++ 目录
C/C++
链接器
常规
输入
清单文件
调试
系统
优化
嵌入的 IDL
Windows 元数据
高级
所有选项
命令行
清单工具
XML 文档生成器
浏览信息
生成事件
自定义生成步骤
代码分析
附加依赖项
忽略所有默认库
忽略特定默认库
模块定义文件
将模块添加到程序集
嵌入托管资源文件
强制符号引用
延迟加载的 DLL
程序集链接资源
ib;uuid.lib;odbc32.lib;odbccp32.lib;%(AdditionalDependencies)
<编辑...>
附加依赖项
指定要添加到链接命令行的附加项。[例如 kernel32.lib]
确定
附加依赖项
Packet.lib
wpcap.lib
计算的值:
Packet.lib
wpcap.lib
继承的值:
kernel32.lib
user32.lib
gdi32.lib
从父级或项目默认设置继承(I)
宏(M) >>
确定
取消

添加对头文件的声明
在使用wpcap.dll进行编译时，需要包含的头文件是pcap.h
在使用Packet.dll进行编译时，需要包含的头文件是packet32.h

WinPcap英文帮助文档

WinPcap中文帮助文档(仅支持到4.0.1)

wpcap.dll常用数据结构

① 接口地址信息

```
struct pcap_addr {
    struct pcap_addr *next; //指向下一个地址
    struct sockaddr *addr;          //指向sockaddr类型的结构
    struct sockaddr *netmask;       //指向addr相应的掩码
    struct sockaddr *broadaddr;     //addr相应的广播地址
    struct sockaddr *dstaddr;       //与addr对应的目标地址，如非点到点，
    则为NULL
};
```

在pcap_findalldevs()中被使用

wpcap.dll常用数据结构

②网卡接口信息

```
typedef struct pcap_if  pcap_if_t
struct pcap_if {
    struct pcap_if *next; //指向下一个接口结构
    char *name;         // 设备名，打开设备时使用
        char *description;      //设备的描述信息
        struct pcap_addr *addresses;//指向地址链表的第一个地址
        u_int32 flags;  //接口标识
};
```

wpcap.dll常用数据结构

转储文件头结构**pcap_file_header**
保存转储文件首部信息

Data Fields

bpf_u_int32	**magic**	//标识位
u_short	**version_major** *Libpcap major version.*	//主版本号
u_short	**version_minor** *Libpcap minor version.*	//次版本号
bpf_int32	**thiszone** *gmt to local correction*	//本地时间
bpf_u_int32	**sigfigs** *accuracy of timestamps*	//时间戳
bpf_u_int32	**snaplen** *max length saved portion of each pkt*	//所捕获数据帧的最大长度
bpf_u_int32	**linktype** *data link type (LINKTYPE_*)*	//数据链路类型

wpcap.dll常用数据结构

④包描述头：dump文件中每个数据包的头部描述信息。

```
struct pcap_pkthdr {
    struct timeval ts;   //时间戳
    bpf_u_int32 caplen; //数据包保存在pcap文件中的实际长度，以字节为单位
    bpf_u_int32 len;    //所抓获的数据包的真实长度，如果文件中保存不是完整的数据包，那么这个值可能要比前面的数据包长度的值大。};
struct timeval{
   DWORD      GMTtime; //秒计时：32位,数据包抓获的时间
   DWORD      microTime; //毫秒计时：32位，抓取数据包时的毫秒值}
```

wpcap.dll常用数据结构

统计数据结构：**pcap_stat**
保存数据监测的统计信息

Data Fields

u_int	**ps_recv** *number of packets transited on the network*	//网上已传送的数据帧数
u_int	**ps_drop** *number of packets dropped by the driver*	//删除的数据帧数
u_int	**ps_ifdrop** *drops by interface, not yet supported*	//接口拒绝的帧数，暂不支持
u_int	**bs_capt** ***Win32 specific.*** *number of packets captured, i.e number of packets that are accepted by the filter, that find place in the kernel buffer and therefore that actually reach the application. For backward compatibility,* ***pcap_stats()*** *does not fill this member, so use* ***pcap_stats_ex()*** *to get it.*	//win32专用，捕获的帧数

wpcap.dll常用数据结构

用户认证信息结构：**pcap_rmtauth**
保存远程主机上的用户认证信息

Data Fields

int	**type**	//身份认证类型 *Type of the authentication required.*
char *	**username**	//用户名 *Zero-terminated string containing the username that has to be used on the remote machine for authentication.*
char *	**password**	//口令 *Zero-terminated string containing the password that has to be used on the remote machine for authentication.*

wpcap.dll常用数据结构

捕捉实例

typedef struct pcap pcap_t

一个已打开的捕捉实例的描述符。这个结构体对用户来说是不透明的，它通过wpcap.dll提供的函数，维护了它的内容。

在打开网卡后的一系列操作中作为本次捕获的标识。

wpcap.dll的常用函数

int **pcap_findalldevs** (**pcap_if_t** **alldevsp, char *errbuf)
*构造一个可打开的网络设备的列表 **pcap_open_live()***

char * **pcap_lookupdev** (char *errbuf) X
返回系统中第一个合法的设备

int **pcap_lookupnet** (const char *device, **bpf_u_int32** *netp, **bpf_u_int32** *maskp, char *errbuf)
返回接口的子网和掩码

pcap_dumper_t * **pcap_dump_open** (**pcap_t** *p, const char *fname)
打开一个文件来写入数据包

pcap_t * **pcap_open_live** (const char *device, int snaplen, int promisc, int to_ms, char *ebuf)
在网络中打开一个活动的捕获

pcap_t * **pcap_open_offline** (const char *fname, char *errbuf)
打开一个 tcpdump/libpcap 格式的存储文件，来读取数据包

int **pcap_compile** (**pcap_t** *p, struct bpf_program *fp, char *str, int optimize, **bpf_u_int32** netmask)
编译数据包过滤器，将程序中高级的过滤表达式，转换成能被内核级的过滤引擎所处理的东西。（参见 过滤表达式语法）

int **pcap_setfilter** (**pcap_t** *p, struct bpf_program *fp)
在捕获过程中绑定一个过滤器

int **pcap_dispatch** (**pcap_t** *p, int cnt, **pcap_handler** callback, u_char *user)
收集一组数据包 非阻塞

int **pcap_loop** (**pcap_t** *p, int cnt, **pcap_handler** callback, u_char *user)
收集一组数据包 阻塞

u_char * **pcap_next** (**pcap_t** *p, struct **pcap_pkthdr** *h)
返回下一个可用的数据包

void **pcap_close** (**pcap_t** *p)
关闭一个和p关联的文件，并释放资源

int **pcap_setbuff** (**pcap_t** *p, int dim)
设置与当前适配器关联的内核缓存大小 //window平台专用

int **pcap_setmode** (**pcap_t** *p, int mode)
将接口p的工作模式设置为mode //window平台专用

wpcap.dll的常用函数

int **pcap_stats** (**pcap_t** *p, struct **pcap_stat** *ps)
返回当前捕获的统计信息

int **pcap_sendpacket** (**pcap_t** *p, u_char *buf, int size)
发送一个原始数据包

FILE * **pcap_file** (**pcap_t** *p)
返回一个脱机捕获文件的标准流

wpcap.dll工作流程
1.捕获数据帧

- 1)调用pcap_lookupdev()函数获得主机上的网络设备，该函数返回一个指向主机上的网络设备的指针;
- 2)选择待捕获数据帧的网络设备，调用pcap_open_live()函数打开这个网络设备，该函数根据捕获需求设置网卡的工作模式，并返回一个数据帧捕获描述符pcap_t
- 3)调用pcap_compile()函数编译过滤规则，并使用pcap_setfilter()来设置过滤器规则；(本步骤可选)
- 4)调用pcap_loop()或pcap_dispatch()函数捕获网络上所有的数据帧
- 5)调用pcap_close()函数关闭库

wpcap.dll工作流程
2.发送数据帧

- 1)调用pcap_lookupdev()函数获得主机上的网络设备（pcap_lookupdev()在新版中已废弃，建议用pcap_findalldevs()）
- 2)选择待发送数据帧的网络设备，调用pcap_open_live()函数打开这个网络设备，并返回一个数据帧捕获描述符pcap_t
- 3)构造一个原始数据帧(链路帧)
- 4)调用pcap_sendpacket()函数发送数据帧
- 5)调用pcap_close()函数关闭库

wpcap.dll工作流程
3.统计网络流量

- 1)调用pcap_lookupdev()函数获得主机上的网络设备
- 2)选择待统计流量的网络设备，通过read_timeout来设置统计的事件间隔，调用pcap_open_live()函数打开这个网络设备
- 3)调用pcap_compile()函数编译过滤规则，并使用pcap_setfilter()函数设置过滤规则(可选)
- 4)调用pcap_setmode()函数设置设备为统计模式
- 5)调用pcap_loop()或pcap_dispatch()函数统计网络上所有的数据帧，并在回调函数中进行统计计算
- 6)调用pcap_close()函数关闭库

wpcap.dll工作流程
4.转储网络流量

- 1)调用pcap_lookupdev()函数获得主机上的网络设备
- 2)选择转储流量的网络设备，调用pcap_open()函数打开这个网络设备
- 3)调用pcap_compile()函数编译过滤规则，并使用pcap_setfilter()函数设置过滤规则(可选)
- 4)调用pcap_dump_open()函数打开转储文件
- 5)调用pcap_loop()或pcap_dispatch()函数捕获网络上所有的数据帧，并在回调函数中调用pcap_dump()函数进行原始数据的转储操作。
- 6)调用pcap_close()函数关闭库

使用pcap.dll编程

1、获取设备列表

- int pcap_findalldevs_ex (char * source, struct pcap_rmtauth * auth, pcap_if_t ** alldevs, char * errbuf)
- 功能：函数返回一个 pcap_if 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。
- 举例 ：使用WPcap获得设备列表

使用pcap.dll编程

2、打开适配器并捕获数据包

- pcap_t* pcap_open (const char * *source*, int *snaplen*, int *flags*, int *read_timeout*, struct pcap_rmtauth * *auth*, char * *errbuf*)

  功能：打开网卡，设置混杂模式

- int pcap_loop (pcap_t * *p*, int *cnt*, pcap_handler *callback*, u_char * *user*)

  功能：抓包，每抓到包后调用callback函数处理。

- 举例:使用回调函数打开适配器并捕获数据包

使用pcap.dll编程

3、过滤数据包

- int pcap_compile (pcap_t * *p*, struct bpf_program * *fp*, char * *str*, int *optimize*, bpf_u_int32 *netmask*)

  将一个高层的布尔过滤表达式编译成一个能够被过滤引擎所解释的低层的字节码。

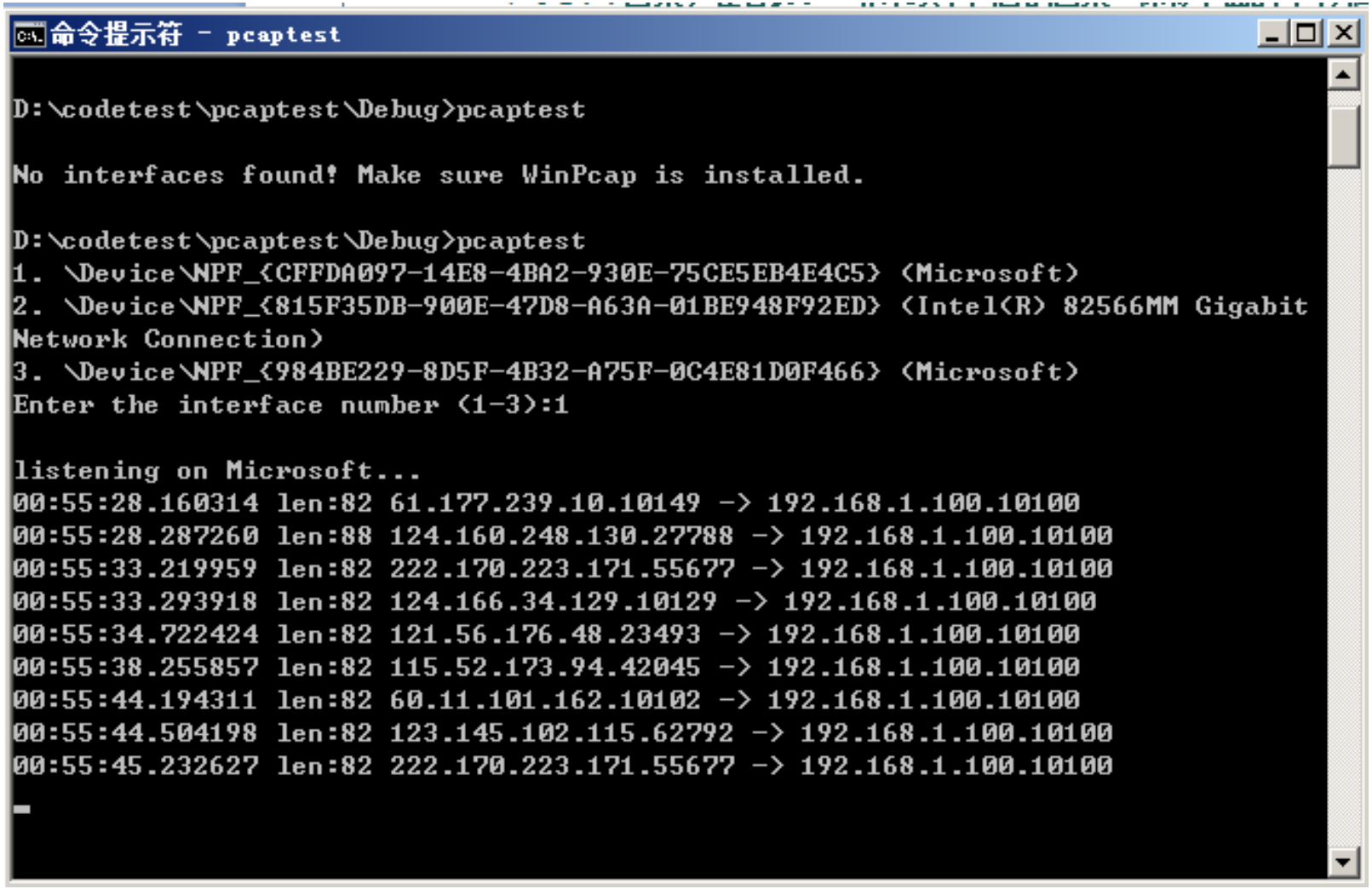
- int pcap_setfilter (pcap_t * *p*, struct bpf_program * *fp*)

  将一个过滤器与内核捕获会话关联。

- 举例：过滤并分析UDP数据包

使用pcap.dll编程

3、



使用pcap.dll编程

4、发送原始数据

int pcap_sendpacket (pcap_t * *p*, u_char * *buf*, int *size*)

功能：发送一个原始数据包到网络中。

- p：设备标识
- buf：待发送的数据包
- size：缓冲区buf的长度

举例：发送原始数据包

WinPcap的过滤规则

- 1) 表达式支持逻辑操作符，可以使用关键字 and、or、not对子表达式进行组合，同时支持使用小括号。
- 2) 基于协议的过滤：使用协议限定符：ip、arp、rarp、tcp、udp等。
- 3) 基于MAC地址的过滤要使用限定符ether（代表以太网地址）
  - 仅作为源地址时： ether src mac_addr
  - 仅作为目的地址:ether dst mac_addr
  - 既作为源地址又作为目的地址:ether host mac_addr
  - mac_addr应该遵从00:E0:4C:E0:38:88的格式

WinPcap的过滤规则

- 4）基于IP地址的过滤应该使用限定符host （代表主机地址）
  - 仅作为源地址：src host ip_addr
  - 仅作为目的地址：dst host ip_addr
  - 既作为源地址又作为目的地址：host ip_addr
- 5) 基于端口的过滤应使用限定符 port。
- 举例：
  - 仅接收80端口的数据包：port 80。
  - 只捕获arp或icmp数据包：arp or （ip and icmp）
  - 捕获主机192.168.1.23与192.168.1.28之间传递的所有UDP数据包：（ip and udp）and（ host 192.168.1.23 or host 192.168.1.28）

UDP流量捕获程序

```
#include <pcap.h>
#include <tchar.h>

/* IP地址结构 */
typedef struct ip_address
{
    u_char byte1;
    u_char byte2;
    u_char byte3;
    u_char byte4;
}ip_address;

/* IPv4首部结构 */
typedef struct ip_header {
    u_char  ver_ihl;        // 版本 (4 bits) + 首部长度 (4 bits)
    u_char  tos;            // 服务类型(Type of service)
    u_short tlen;           // 总长(Total length)
    u_short identification; // 标识(Identification)
    u_short flags_fo;       // 标志位(Flags) (3 bits) + 段偏移量(Fragment offset) (13 bits)
    u_char  ttl;            // 存活时间(Time to live)
    u_char  proto;          // 协议(Protocol)
    u_short crc;            // 首部校验和(Header checksum)
    ip_address  saddr;      // 源地址(Source address)
    ip_address  daddr;      // 目的地址(Destination address)
    u_int   op_pad;         // 选项与填充(Option + Padding)
}ip_header;


/* UDP首部*/
typedef struct udp_header {
    u_short sport;          // 源端口(Source port)
    u_short dport;          // 目的端口(Destination port)
    u_short len;            // UDP数据包长度(Datagram length)
    u_short crc;            // 校验和(Checksum)
}udp_header;
```

```c
/* 对回调函数packet_handler的声明 */
void packet_handler(u_char* param, const struct pcap_pkthdr* header, const u_char* pkt_data);


int _tmain(int argc, _TCHAR* argv[])
{
    pcap_if_t* alldevs;
    pcap_if_t* d;
    int inum;
    int i = 0;
    pcap_t* adhandle;
    char errbuf[PCAP_ERRBUF_SIZE];
    u_int netmask;
    char packet_filter[] = "ip and udp";
    struct bpf_program fcode;

    /* 获取设备列表 */
    if (pcap_findalldevs(&alldevs, errbuf) == -1)
    {
        fprintf(stderr, "pcap_findalldevs函数调用错误：%s\n", errbuf);
        return 1;;
    }
```

```
/* 打印设备列表 */
for (d = alldevs; d; d = d->next)
{
    printf("%d. %s", ++i, d->name);
    if (d->description)
        printf(" (%s)\n", d->description);
    else
        printf(" (没有可用的描述符)\n");
}

if (i == 0)
{
    printf("\n无法找到网络接口!请确认WinPcap已正确安装.\n");
    return -1;
}

printf("请待捕获数据的输入网卡编号 (1-%d):", i);
scanf_s("%d", &inum);

/* 检查用户输入的网卡编号是否合法 */
if (inum < 1 || inum > i)
{
    printf("\nAdapter number out of range.\n");

    /* 释放设备列表 */
    pcap_freealldevs(alldevs);
    return -1;
}
```

```
/* 跳转到用户选择的网卡 */
for (d = alldevs, i = 0; i < inum - 1; d = d->next, i++);

/* 打开网卡 */
adhandle = pcap_open_live(d->name,                    // 设备名
    65536,                          // 保证能捕获到不同数据链路层上的每个数据包的全部内容
    1,                              // 混杂模式
    1000,                           // 读取超时时间
    errbuf                          // 错误缓冲池
);
if (adhandle == NULL)
{
    fprintf(stderr, "\n无法打开网络适配器. WinPcap不支持%s \n", d->name);
    // 释放设备列表
    pcap_freealldevs(alldevs);
    return -1;

}

/* 检查链接层.本程序只支持以太网. */
if (pcap_datalink(adhandle) != DLT_EN10MB)
{
    fprintf(stderr, "\n本程序仅在以太网环境下可用.\n");
    /* 释放设备列表 */
    pcap_freealldevs(alldevs);
    return -1;
}

if (d->addresses != NULL)
    /* 获得第一个接口地址的网络掩码 */
    netmask = ((struct sockaddr_in*)(d->addresses->netmask))->sin_addr.S_un.S_addr;
else
    /* 如果接口没有地址，那么我们假设一个C类的掩码 */
    netmask = 0xffffff;
```

```
    //编译过滤器
    if (pcap_compile(adhandle, &fcode, packet_filter, 1, netmask) < 0)
    {
        fprintf(stderr, "\n无法编译过滤规则，请检查语法的正确性.\n");
        // 释放设备列表
        pcap_freealldevs(alldevs);
        return -1;
    }

    //设置过滤器
    if (pcap_setfilter(adhandle, &fcode) < 0)
    {
        fprintf(stderr, "\n设置过滤器错误.\n");
        // 释放设备列表
        pcap_freealldevs(alldevs);
        return -1;
    }


    printf("\nlistening on %s...\n", d->description);

    /* 不需要其他的设备列表信息，释放资源 */
    pcap_freealldevs(alldevs);

    /* 开始流量捕获 */
    pcap_loop(adhandle, 0, packet_handler, NULL);

    return 0;
}
```

```c
/* 每当有数据包进入，则以下回调函数被WinPcap调用 */
void packet_handler(u_char* param, const struct pcap_pkthdr* header, const u_char* pkt_data)
{
    struct tm time1;
    struct tm* ltime=&time1;
    char timestr[16];
    ip_header* ih;
    udp_header* uh;
    u_int ip_len;
    u_short sport, dport;
    time_t local_tv_sec;
    /* 将时间戳转换成可识别的格式 */
    local_tv_sec = header->ts.tv_sec;
    localtime_s(ltime,&local_tv_sec);
    strftime(timestr, sizeof timestr, "%H:%M:%S", ltime);
    /* 打印数据包的时间戳和长度 */
    printf("%s.%.6d len:%d ", timestr, header->ts.tv_usec, header->len);
    /* 获得IP数据包头部的位置 */
    ih = (ip_header*)(pkt_data + 14); //以太网头部长度
    /* 获得UDP首部的位置 */
    ip_len = (ih->ver_ihl & 0xf) * 4;
    uh = (udp_header*)((u_char*)ih + ip_len);
    /* 将网络字节序列转换成主机字节序列 */
    sport = ntohs(uh->sport);
    dport = ntohs(uh->dport);
    /* 打印IP地址和UDP端口 */
    printf("%d.%d.%d.%d.%d -> %d.%d.%d.%d.%d\n",
        ih->saddr.byte1,
        ih->saddr.byte2,
        ih->saddr.byte3,
        ih->saddr.byte4,
        sport,
        ih->daddr.byte1,
        ih->daddr.byte2,
        ih->daddr.byte3,
        ih->daddr.byte4,
        dport);
}
```

关于Packet.dll

非常重要的注意事项，请仔细阅读！

Packet.dll 的源代码是开放的，并且有完整的文档。 然而，packet.dll应该被认为是核心API，因为它建立在WinPcap内部的目的，就是为真正的公共API:wpcap.dll建立一层"隔离墙"。

由于应用程序使用WinPcap的**常规(normal)方式和推荐(suggested)方式**，是通过wpcap.dll，所以，**我们不保证packet.dll的API不会在未来的WinPcap的发行版中被修改，并且，我们不提供这个API的支持。**由于这个原因，**在这个指南中，不会含有有关Packet.dll的文档。**用户可以自行使用Doxygen来创建它们，或者阅读代码中的注释。

不建议使用Packet.dll。