9.1 网络监听

- 在一个共享式网络，可以听取所有的流量
- 是一把双刃剑
  - 管理员可以用来监听网络的流量情况
  - 开发网络应用的程序员可以监视程序的网络情况
  - 黑客可以用来刺探网络情报
- 目前有大量商业的、免费的监听工具，俗称嗅探器（**Sniffer**）

9.1.1 以太网络的工作原理

- 载波侦听/冲突检测(**CSMA/CD, carrier sense multiple access with collision detection**)技术
  - 载波侦听：是指在网络中的每个站点都具有同等的权利，在传输自己的数据时，首先监听信道是否空闲
    - 如果空闲，就传输自己的数据
    - 如果信道被占用，就等待信道空闲
  - 而冲突检测则是为了防止发生两个站点同时监测到网络没有被使用时而产生冲突
- 以太网采用了**CSMA/CD**技术，由于使用了广播机制，所以，所有与网络连接的工作站都可以看到网络上传递的数据

以太网卡的工作模式

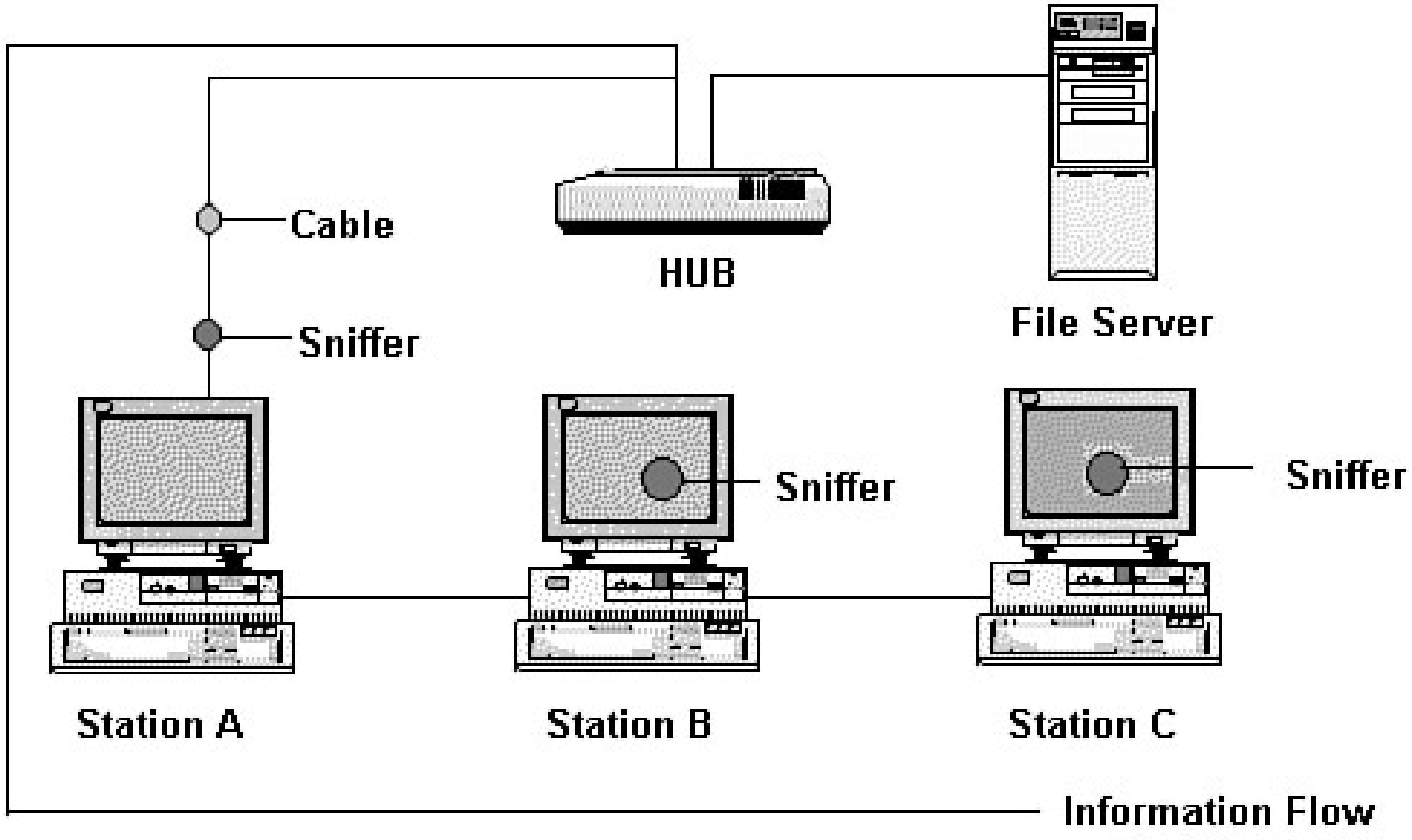
- 网卡的**MAC**地址**(48**位**)**
  - 通过**ARP**来解析**MAC**与**IP**地址的转换
  - 用**ipconfig/ifconfig**可以查看**MAC**地址
- 正常情况下，网卡应该只接收这样的包
  - **MAC**地址与自己相匹配的数据帧
  - 广播包
- 网卡完成收发数据包的工作，两种接收模式
  - 混杂模式：不管数据帧中的目的地址是否与自己的地址匹配，都接收下来
  - 非混杂模式：只接收目的地址相匹配的数据帧，以及广播数据包(和组播数据包)
- 为了监听网络上的流量，必须设置为混杂模式

9.1.2 共享网络和交换网络

- 共享式网络
  - 通过网络的所有数据包发往每一个主机
  - 最常见的是通过HUB连接起来的子网

- 交换式网络
  - 通过交换机连接网络
  - 由交换机构造一个“MAC地址-端口”映射表
  - 发送包的时候，只发到特定的端口上

共享式网络示意图


Cable
HUB
File Server
Sniffer
Sniffer
Sniffer
Station A
Station B
Station C
Information Flow

9.1.3 应用程序抓包的技术

- **UNIX系统提供了标准的API支持**
  - **Packet socket**
  - **BPF**

- **Windows平台上通过驱动程序来获取数据包**
  - 驱动程序
  - **WinPcap**

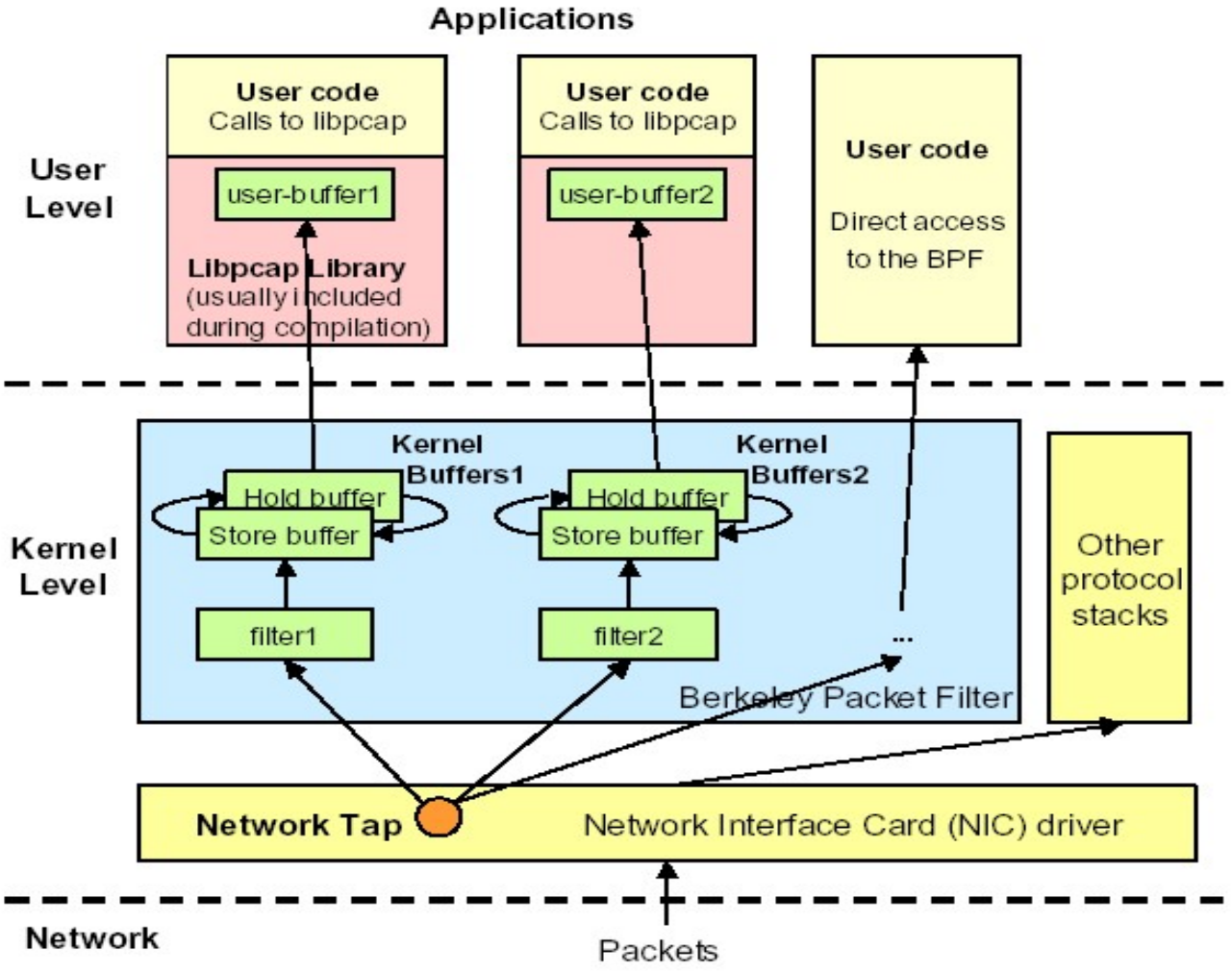
Packet socket

- 设置混杂模式
  - 用ioctl()函数可以设置
- 打开一个packet socket
  - packet_socket = socket(PF_PACKET, int socket_type, int protocol);
  - 以前的做法，socket(PF_INET, SOCK_PACKET, protocol)
- 不同的UNIX或者Linux版本可能会有不同的函数调用，本质上
  - 打开一个socket(或者通过open打开一个设备)
  - 通过ioctl()或者setsockopt()设置为混杂模式

BPF(Berkeley Packet Filter)

- BSD抓包法
  - BPF是一个核心态的组件，也是一个过滤器
  - Network Tap接收所有的数据包
  - Kernel Buffer，保存过滤器送过来的数据包
  - User buffer，用户态上的数据包缓冲区
- Libpcap(一个抓包工具库)支持BPF
  - Libpcap是用户态的一个抓包工具
  - Libpcap几乎是系统无关的
- BPF是一种比较理想的抓包方案
  - 在核心态，所以效率比较高，
  - 但是，只有少数OS支持(主要是一些BSD操作系统)

BPF和libpcap



关于libpcap

- 用户态下的packet capture
- 系统独立的接口，C语言接口
- 目前最新为0.7版本
- 广泛应用于：
  - 网络统计软件
  - 入侵检测系统
  - 网络调试
- 支持过滤机制，BPF

Libpcap介绍

- 为捕获数据包做准备的几个函数
  - **char *pcap_lookupdev(char *errbuf);**
    返回一个指向网络设备的指针，这个指针下面用到
  - **pcap_t *pcap_open_live(char *device, int snaplen, int promisc, int to_ms, char *ebuf);**
    用来获取一个**packet capture descriptor**；
    snaplen指定了抓取数据包的最大长度
  - **pcap_dumper_t *pcap_dump_open(pcap_t *p, char *fname);**
    打开一个**savefile**文件，用于**dump**
  - **pcap_t *pcap_open_offline(char *fname, char *ebuf);**
    打开一个**savefile**，从中读取数据包

Libpcap：dump文件格式

- 文件头：

```
struct pcap_file_header {
    bpf_u_int32 magic;
          // 0xa1b2c3d4
    u_short version_major;
    u_short version_minor;
    bpf_int32 thiszone;
    bpf_u_int32 sigfigs;
    bpf_u_int32 snaplen;
    bpf_u_int32 linktype;
};
```

- 然后是每一个包的包头和数据

```
struct pcap_pkthdr {
        struct timeval ts;
        bpf_u_int32 caplen;
        bpf_u_int32 len;
};
```

其中数据部分的长度为caplen

Libpcap: 设置filter

- 设置过滤器用到的函数
  - **int pcap_lookupnet(char *device, bpf_u_int32 *netp, bpf_u_int32 *maskp, char *errbuf)**
    获得与网络设备相关的网络号和掩码
  - **int pcap_compile(pcap_t *p, struct bpf_program *fp, char *str, int optimize, bpf_u_int32 netmask)**
    把字符串**str**编译成一个过滤器程序
  - **int pcap_setfilter(pcap_t *p, struct bpf_program *fp)**
    设置一个过滤器

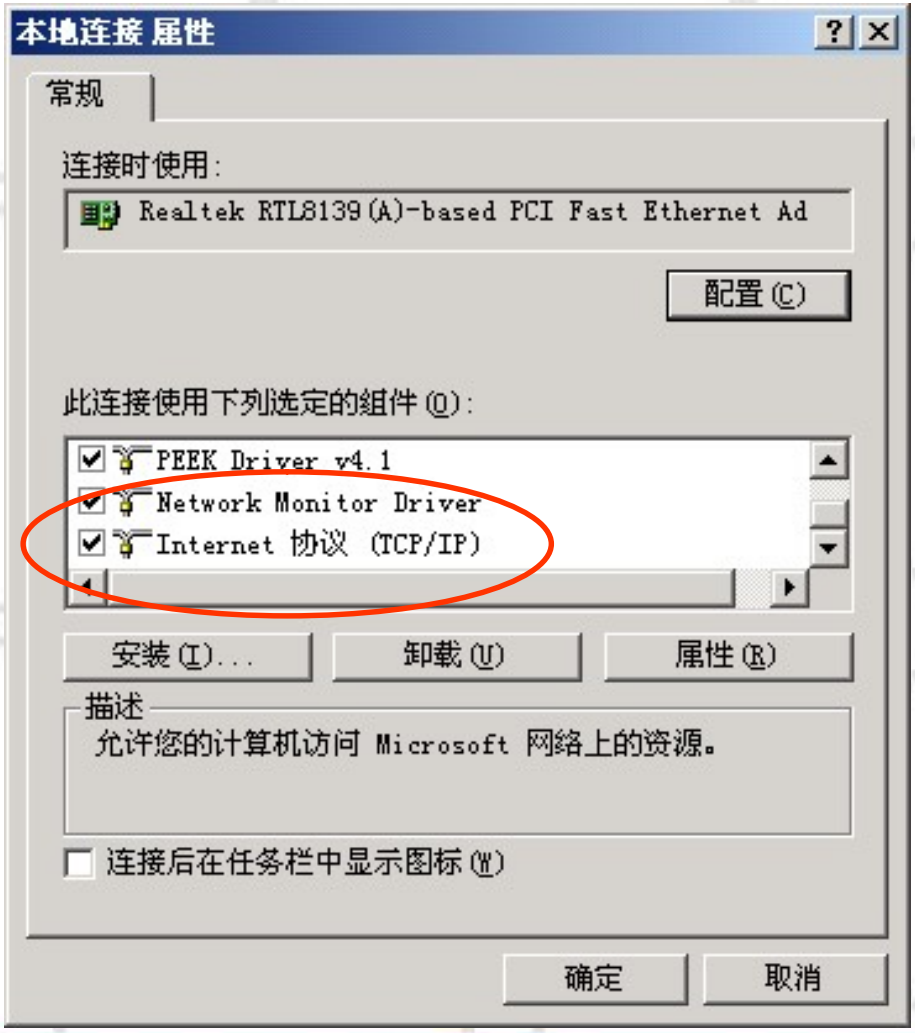
Libpcap: 捕获数据

- 捕获数据用到的两个函数
  - **int pcap_dispatch(pcap_t *p, int cnt, pcap_handler callback, u_char *user)**
  - **int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user)**
  - 参数含义：
    - **cnt**指定了捕获数据包的最大数目
    - **pcap_handler**是一个回调函数
  - 二者区别在于**pcap_loop**不会因为**read**操作超时而返回。
  - 另一个函数：**void pcap_dump(u_char *user, struct pcap_pkthdr *h, u_char *sp)**
    把数据包写到一个由**pcap_dump_open()**打开的文件中

Windows平台下的抓包技术

- 内核本身没有提供标准的接口
- 通过增加一个驱动程序或者网络组件来访问内核网卡驱动提供的数据包
  - 在Windows不同操作系统平台下有所不同
- 不同sniffer采用的技术不同
  - WinPcap是一个重要的抓包工具，它是libpcap的Windows版本

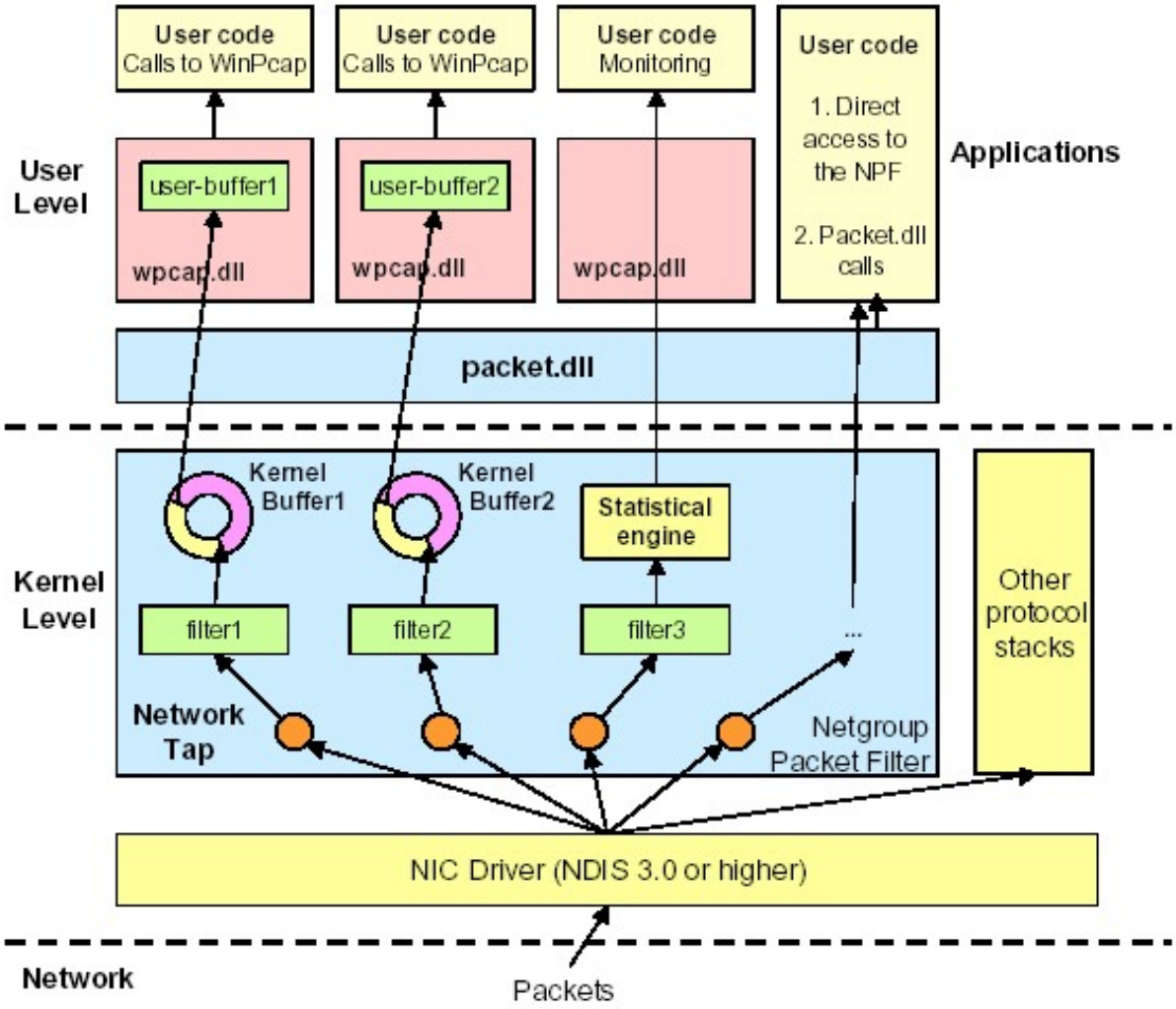
Windows 2000下抓包组件



WinPcap

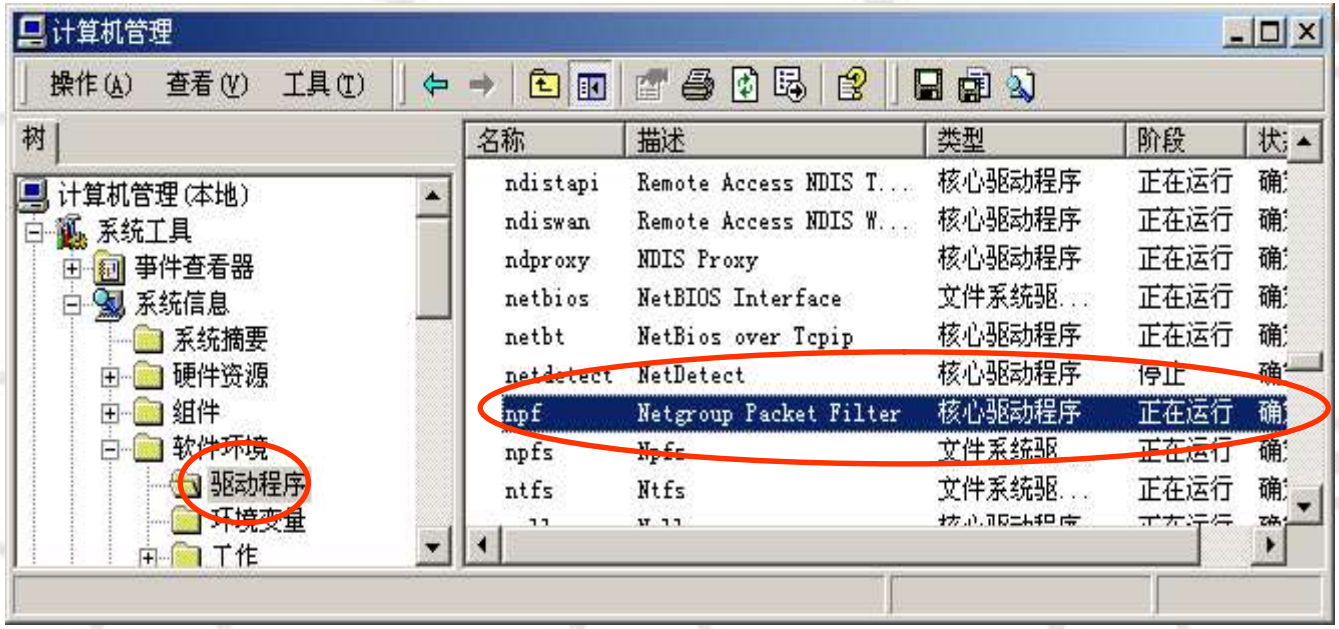
- **WinPcap包括三个部分**
  - 第一个模块**NPF(Netgroup Packet Filter)**，是一个虚拟设备驱动程序文件。它的功能是过滤数据包，并把这些数据包原封不动地传给用户态模块，这个过程中包括了一些操作系统特有的代码
  - 第二个模块**packet.dll**为**win32**平台提供了一个公共的接口。不同版本的**Windows**系统都有自己的内核模块和用户层模块。**Packet.dll**用于解决这些不同。调用**Packet.dll**的程序可以运行在不同版本的**Windows**平台上，而无需重新编译
  - 第三个模块 **Wpcap.dll**是不依赖于操作系统的。它提供了更加高层、抽象的函数。
- **packet.dll和Wpcap.dll**
  - **packet.dll**直接映射了内核的调用
  - **Wpcap.dll**提供了更加友好、功能更加强大的函数调用

WinPcap和NPF



Windows的网络结构

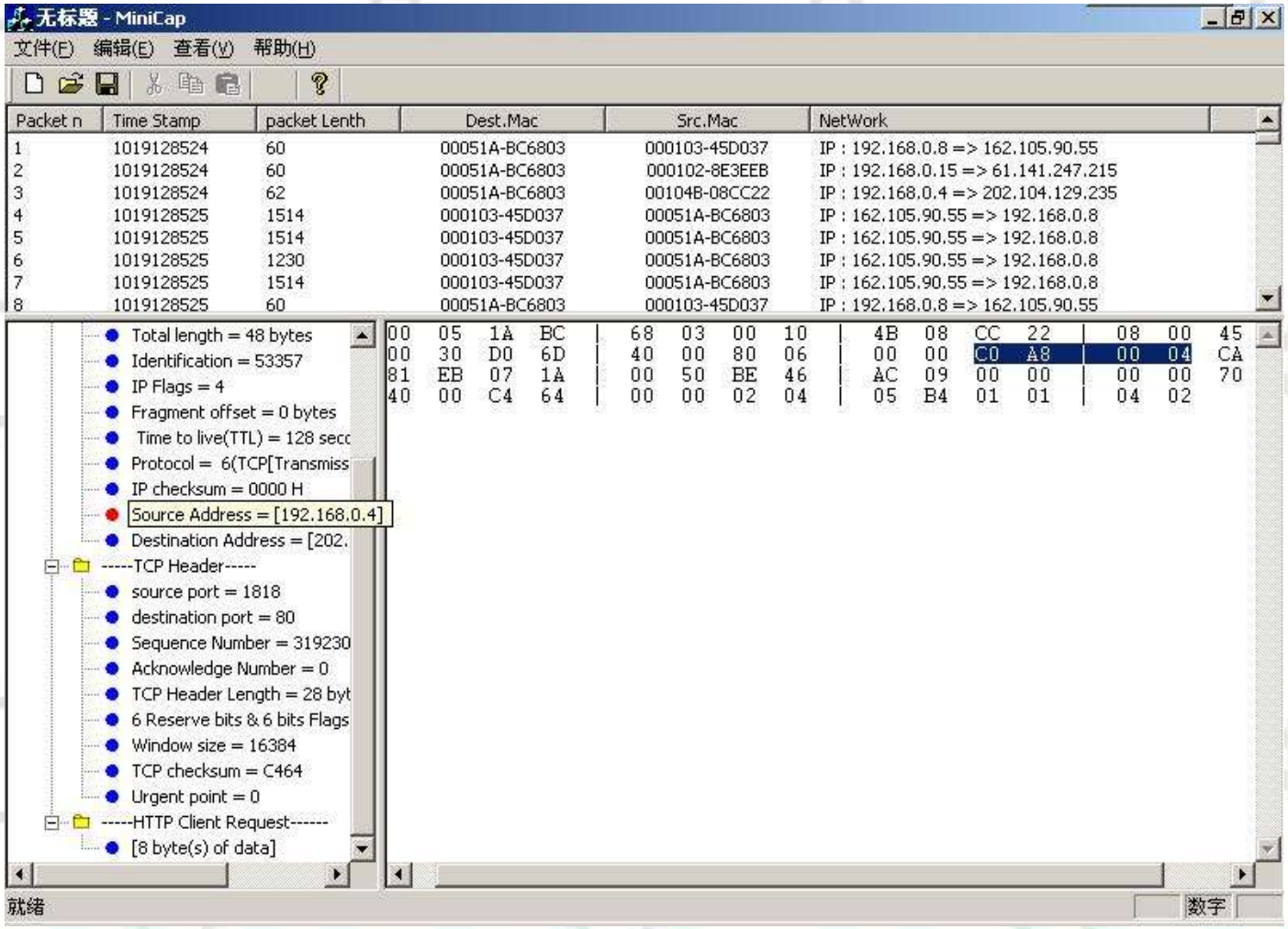
- **NDIS(Network Driver Interface Specification，网络驱动接口规范)描述了网络驱动与底层网卡之间的接口规范，以及它与上层协议之间的规范**
- **NPF作为一个核心驱动程序而提供的**



WinPcap的优势

- 提供了一套标准的抓包接口
  - 与libpcap兼容，可使得原来许多UNIX平台下的网络分析工具快速移植过来
  - 便于开发各种网络分析工具
- 除了与libpcap兼容的功能之外，还有
  - 充分考虑了各种性能和效率的优化，包括对于NPF内核层次上的过滤器支持
  - 支持内核态的统计模式
  - 提供了发送数据包的能力

用WinPcap开发自己的Sniffer



Windows平台下一些Sniffer工具

- **Buttsniffer**
  - 简单，不需要安装，可以在**Windows NT**下运行，适合于后台运作
- **NetMon**
  - **Windows 2000**自带**(Microsoft SMS**也带**)**
  - 友好的图形界面，分析功能强
- **NetXRay**
  - 界面友好，统计分析功能强，过滤器功能
- 基于**WinPcap**的工具
  - **WinDump**
  - **Analyzer**

UNIX/Linux平台下的一些Sniffer工具

- dsniff
- linux_sniffer
- Snort
- tcpdump
- sniffit
- ……

9.1.3 与sniffer有关的两项技术

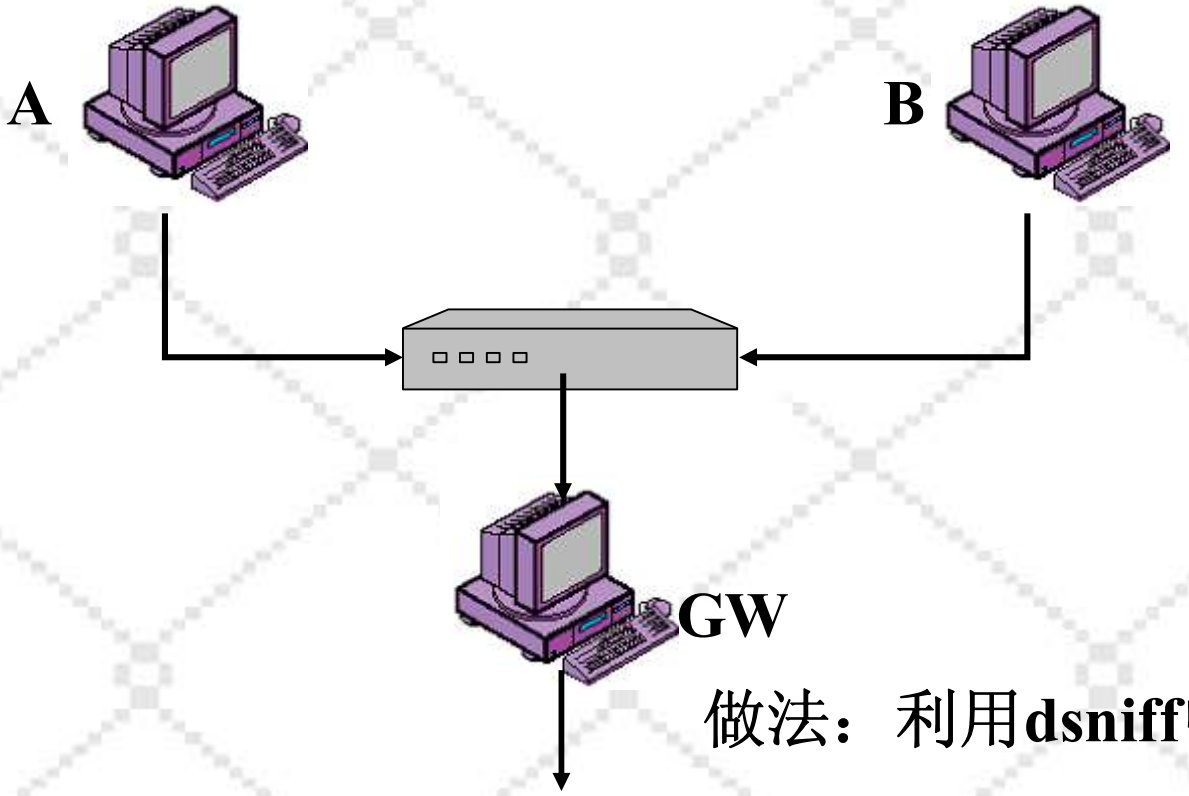
- 检测处于混杂模式的节点
- 如何在交换网络上实现**sniffer**

检测处于混杂模式的节点

- 网卡和操作系统对于是否处于混杂模式会有一些不同的行为，利用这些特征可以判断一个机器是否运行在混杂模式下
- 一些检测手段
  - 根据操作系统的特征
    - **Linux**内核的特性：正常情况下，只处理本机**MAC**地址或者以太广播地址的包。在混杂模式下，许多版本的Linux内核只检查 数据包中的IP地址以确定是否送到IP堆栈。因此，可以构造无效以太地址而IP地址有效的ICMP ECHO请求，看机器是否返回应答包(混杂模式)，或忽略(非混杂模式)。
    - **Windows 9x/NT**：在混杂模式下，检查一个包是否为以太广播包时，只看**MAC**地址前八位是否为**0xff**。
  - 根据网络和主机的性能
    - 根据响应时间：向本地网络发送大量的伪造数据包，然后，看目标主机的响应时间，首先要测得一个响应时间基准和平均值
- **L0pht的AntiSniff**产品，参考它的技术文档

在交换式网络上监听数据包

- ARP重定向技术，一种中间人攻击



1 B打开IP转发功能

2 B发送假冒的arp包给A,声称自己是GW的IP地址

3 A给外部发送数据，首先发给B

4 B再转发给GW

做法：利用dsniff中的arpredirect工具

发送数据包——Libnet

- 利用Libnet构造数据包并发送出去
- 关于Libnet
  - 支持多种操作系统平台
  - 提供了50多个C API函数，功能涵盖
    - 内存管理(分配和释放)函数
    - 地址解析函数
    - 各种协议类型的数据包构造函数
    - 数据包发送函数(IP层和链路层)
    - 一些辅助函数，如产生随机数、错误报告等

使用Libnet的基本过程

- 数据包内存初始化
- 网络接口初始化
- 构造所需的数据包
- 计算数据包的校验和
- 发送数据包
- 关闭网络接口
- 释放数据包内存

```
libnet_init_packet(…);
libnet_open_raw_sock(…);
libnet_build_ip(…);
libnet_build_tcp(…);
libnet_do_checksum(…);
libnet_write_ip(…);
libnet_close_raw_sock(…);
libnet_destroy_packet(…);
```