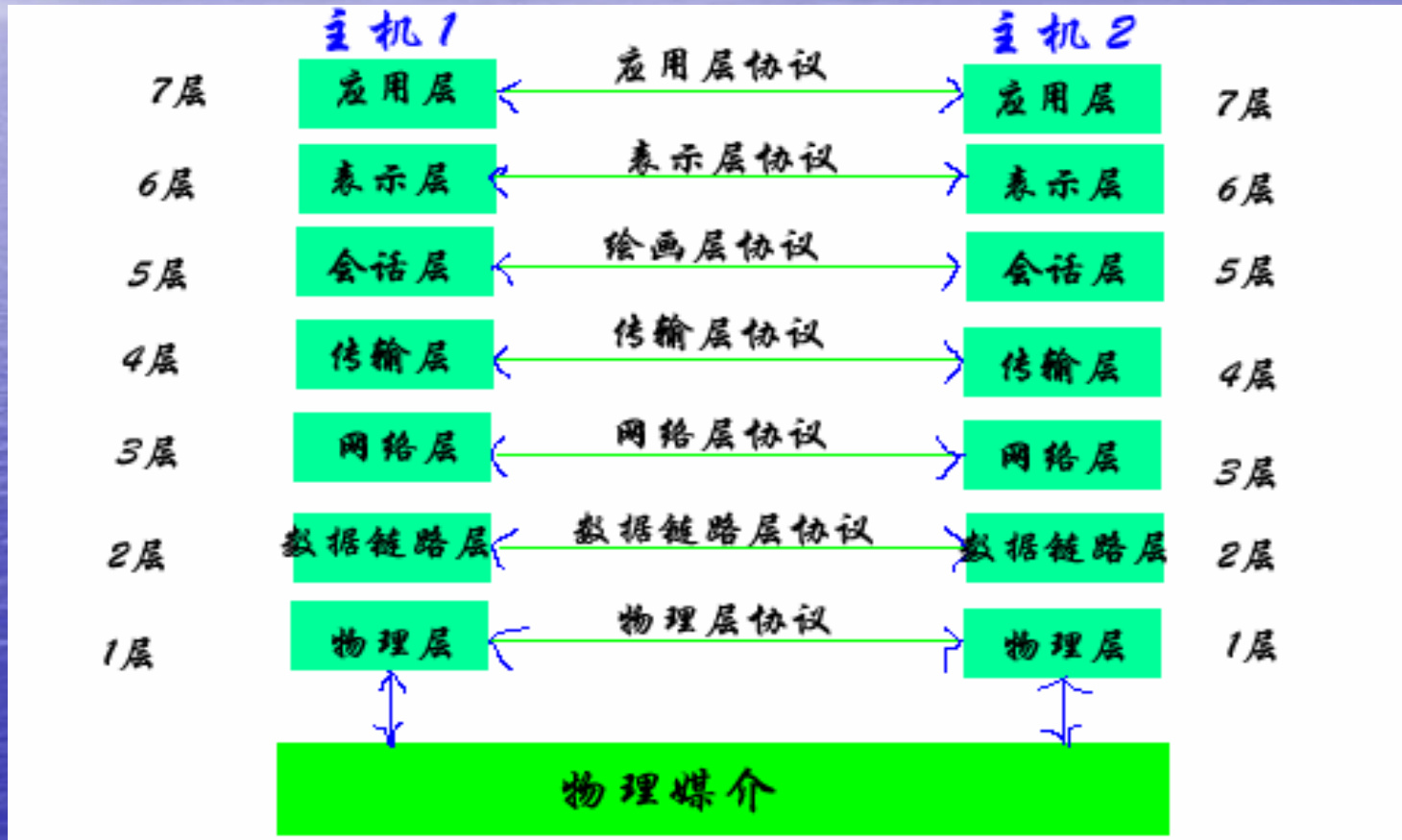
第 7 章　网络

1. 网络基本原理
2. Windows 2000网络体系结构
3. Windows 2000的层次化网络服务

1.网络基本原理

- OSI参考模型
- TCP/IP参考模型

OSI参考模型

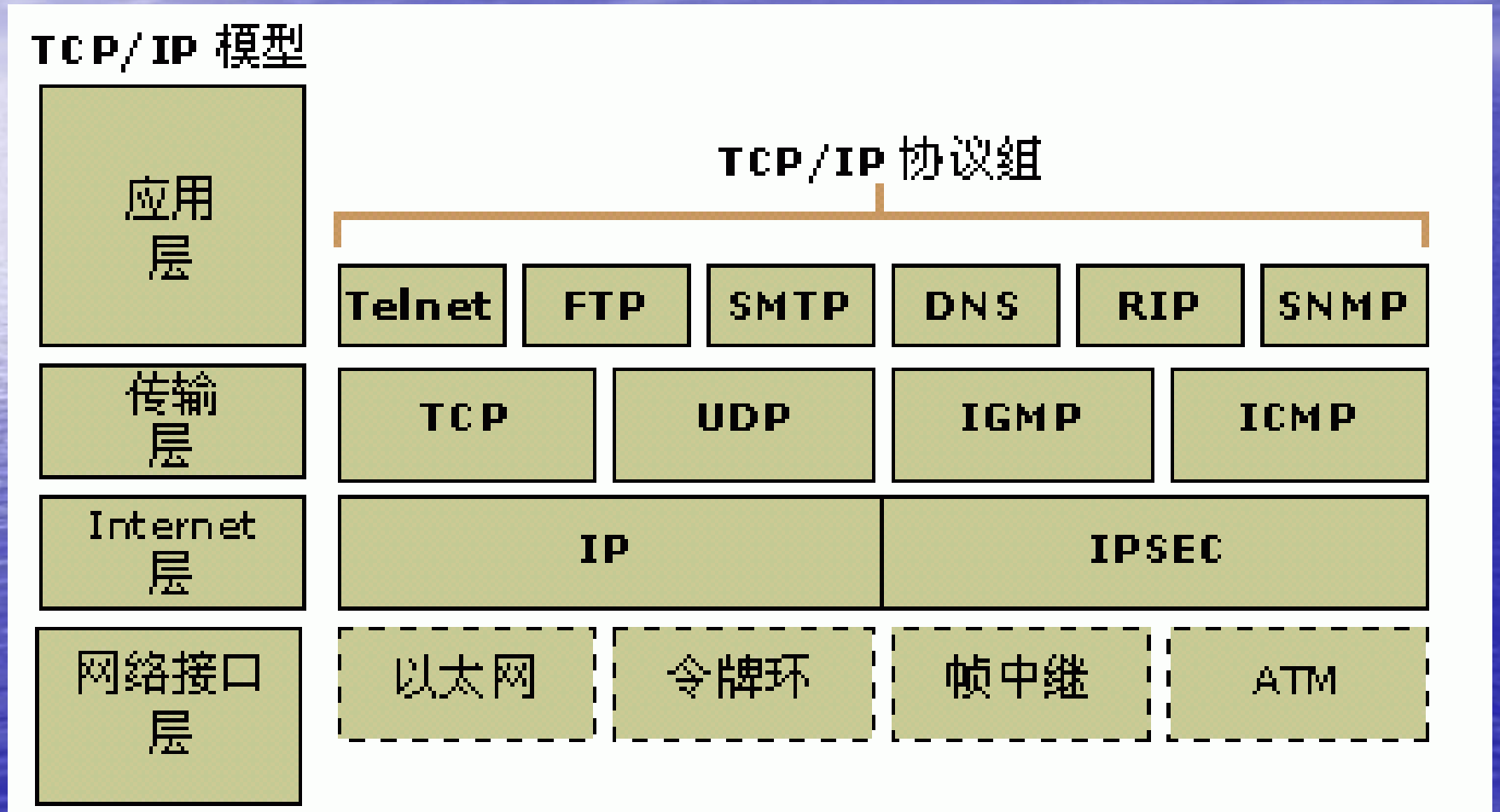


OSI参考模型图

OSI 参考模型

- 应用层　　处理两个网络应用程序之间的信息传输。
- 表示层　负责所传输消息的语法和语义的分析，处理数据的格式化。
- 会话层　管理相互协作的应用程序之间的连接。
- 传输层　从会话层接受数据，传递给网络层，并确保到达对方的信息正确无误。
- 网络层　负责建立分组头，处理路由，拥塞控制，　　以　及网络互连。
- 数据链路层　（DLL）　发送和接收帧。
- 物理层　负责传送比特流。

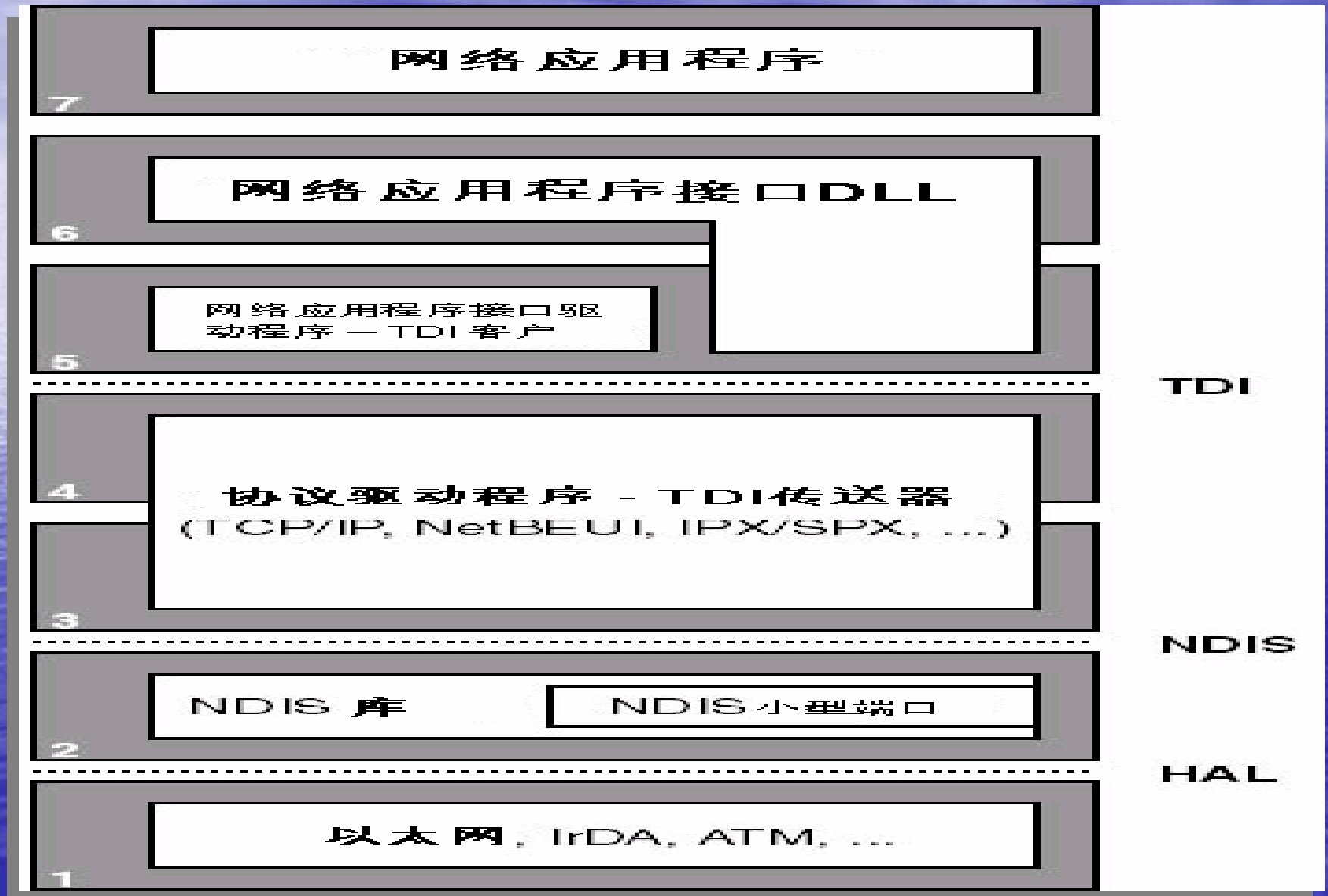
TCP/IP 参考模型


TCP/IP 模型
应用层
传输层
Internet层
网络接口层
TCP/IP 协议组
Telnet
FTP
SMTP
DNS
RIP
SNMP
TCP
UDP
IGMP
ICMP
IP
IPSEC
以太网
令牌环
帧中继
ATM

TCP/IP 参考模型

- 互联网络层
  - 互联网络协议（IP：Internet Protocol ）
  - 地址解析协议（ARP：Address Resolution Protocol）
  - 反向地址解析协议（RARP：Reverse Address Resolution Protocol）
- 传输层
  - 传输控制协议 （TCP 面向连接的数据传输）
  - 用户数据报协议（UDP 面向无连接的数据传输）

2.Windows 2000网络体系结构



2. Windows 2000网络体系结构

- 网络API
- 传输驱动程序接口（TDI）客户
- TDI传送器
- NDIS库
- NDIS小端口驱动程序

NDIS协议驱动程序设计

- 网络驱动程序是硬件与软件的中介
- NDIS驱动程序原理
- 操作系统中数据包传输机制
- NDIS协议驱动程序和上层应用程序

NDIS协议驱动程序设计

- 驱动程序Driver在操作系统的知识体系中属于设备处理这一部分
- 它的功能就是接收—上层用户态程序发宋的抽象请求，如read或write命令，再将它们转换为对设备的具体要求，并且发送给设备控制器，随后设备被启动去执行具体操作。
- 驱动程序也将设备控制器发送的信号传送给上层软件。
- 设备驱动程序就是在I／O(输入／输出)进程与设备控制器之间的通信程序。

NDIS协议驱动程序设计

- 分类：

硬件驱动程序：如显卡驱动程序和网卡驱动程序.

软件驱动程序：TCP/IP驱动程序等。

Windows驱动程序标准和规范有：

WDM Windows驱动程序模型

VXD 虚拟设备驱动程序

NDIS

驱动程序是操作系统与I／O设备之间的桥梁。系统程序员的主要工作就是编写驱动程序主要目的当然是为了驱动真正的硬件

NDIS协议驱动程序设计

- Windows2000的系统分成用户态和核心态，可以与保护模式下的ring3和ring0级别相对应。
- 用户态应用程序对Windows子系统进行Win32 API调用，这个调用由系统服务接口作用到I／O管理器(进行必要的参数匹配和操作安全性检查)，然后由这个请求构造出合适的IRP(I／O请求包)，并把此IRP传给驱动程序。
- 驱动程序直接执行这个请求包，并与硬件打交道，从而完成I／O请求工作，最后由I／O管理器将执行结果返回给用户态程序。

NDIS协议驱动程序设计

- 学习NDIS驱动程序的开发方法，了解其工作原理。
- 掌握DDK(DeviceDriversKit，设备驱动程序开发包)的使用方法，能够编译生成相应的驱动程序并在系统中安装。

NDIS协议驱动程序设计

- NDIS协议驱动程序用DDK作为编译环境，源代码用C语言完成，源文件包括packet．c、openclos．c、read．c、write．c、packet．rc，最后的底层代码编译出一个的packet．sys文件；手写一个packet．inf安装文件；
- 上层应用程序代码也是用C语言完成，源文件包括analyze．c、childwin．c、packet32．c和testapp．c，最后编译出一个testapp．exe文件。packet．sys、packetinf和testapp．exe三者为用户提供了一个具备方便的安装条件的底层驱动与上层应用程序。

Windows中的网络体系结构

- Microsoft Windows 2000网络体系结构依据国际标准化组织开发的7层网络模型。
- 开放系统互联(OSI)参考模型于1978年被引进，用来描述网络是一个—系列的协议层，每个协议层上都有其特定的功能。
- 处于两个相邻层的界面定义了下一个协议层为上一个协议层提供何种服务及如何实现这些服务的内容。

NDIS协议驱动程序设计

- Windows 2000下的网络驱动程序实现网络体系结构中自下而上的4个协议层：
- 物理层
- 数据链路层
- 网络层
- 传输层

NDIS协议驱动程序设计

- 1)物理层。物理层是OSI模式中最低的一层。这一层通过一个物理媒介接收和转发无结构的原始比特流。物理层涉及的主要问题包括电气的、机械的和功能的接口。在Windows 2000下，物理层由网络接口卡(NIC)来实现

NDIS协议驱动程序设计

- 2)数据链路层。

LLC(逻辑链路控制子层)和MAC(介质访问控制子层)。LLC层提供将数据帧从一个节点无错误传输到另一个节点。LLC层建立/终止逻辑链路，控制帧传输、帧排序，LLC层利用帧确认和重传来通过链路来向上一层提供最终的无错误的传输。MAC层管理存取网络媒介，检查帧错误，并管理接收帧地址确认。在Windows 2000网络体系结构中，LLC子层是由传输驱动程序实现的，而MAC子层是由网络接口卡(NIC)来实现。NIC由一个被称为NIC驱动程序的软件来控制。Windows 2000装有可用于多种常用NIC的NIC驱动程序。

NDIS协议驱动程序设计

- 3)网络层。该层控制子网的运行。它决定了数据将要传送的路径，其根据如下：网络条件，服务优先级，以及路径、流量控制、帧分解和重组、逻辑至物理地址映射等其他因素
- 4)传输层。该层确保信息传输无误，并且是按顺序的。

NDIS驱动程序

- 在Windows 2000网络体系结构中，网络层和传输层是由我们所熟知的传输驱动程序(transport driver)软件来实现的。有时这种驱动程序也指协议驱动程序或协议模组。Windows2000装有TCP／IP、IPX／SPX、NetBEUI和语音传输驱动程序。

NDIS驱动程序

- 网络驱动程序接口规范(NDIS)抽象了低层次的硬件来提供给高层次网络上的网络管理，如网络传输。
- 网络驱动程序接口规范(NDIS)使各种协议驱动程序编写独立于各种网卡，复杂网络驱动程序编写提供一组api函数，通过NDIS.SYS这个接口（另一组API）来调用

NDIS驱动程序

- NDIS程序库(NDIS．sys)提供了一个接口，网卡驱动程序与协议层驱动程序及操作系统通过这个接口进行通信。NDIS库还参与管理操作系统中的与网络有关的特定任务，管理所有底层的NIC驱动程序的绑定与状态信息。

NDIS支持网络驱动程序形式

- 小型端口驱动程序
- 中间驱动程序
- 协议驱动程序

NDIS驱动程序

- 1．NDIS小型端口驱动程序

一个NDIS小型端口驱动程序(也称为一小型端口NIC驱动程序)有两种基本功能：

管理一个网络接口卡(NIC)，包括通过NIC发送和接收数据。

与高级驱动程序接口，例如，与中间驱动程序和传输协议驱动程序接口。

- 一个小型端口NIC驱动程序通过NDIS库和它的NIC及高层驱动程序相互通信。

NDIS库导出一个完全的函数集合(NdisXXX函数)来装入小型端口需要调用的操作系统函数。然后，小型端口必须导出一套MiniportXxx函数的实体指针，可供NDIS自己使用或代替高层驱动程序访问小型端口。下面用发送和接收操作作为例子说明了小型端NIC驱动程序与NDIS和高层驱动程序之间的交互。

NDIS驱动程序

- 当一个传输驱动程序需传输一个数据包时，它调用一个由NDIS库导出的NdisXxx函数。然后NDIS通过调用适当的NdisXxx函数将这个数据包传至小型端口。接着小型端口通过调用适当的NdisXxx函数将数据包传至NIC来传输。当一个NIC接收到一个数据包时，它可以发布一个硬件中断让NDIS或NIC的小型端口来进行处理。NDIS通过调用适当的MiniportXxx函数来通知NIC的小型端口。

NDIS驱动程序

- 小型端口对来自NIC的数据建立传输，然后通过调用适当的NdisXxx函数标识接收到的数据
- 来绑定到高层驱动程序上。

1 对于无连接环境和面向连接的环境，NDIS都支持小型端口驱动程序。无连接小型端口控制NIC用于无连接网络媒体，如以太网、FDDI和令牌环。

2 面向连接的小型端口控制NIC用于面向连接的网络媒体，如ATM和ISDN。

小型端口和外设的通信是通过发送输入／输出(I／O)请求包(IRP)

NDIS驱动程序

- 2．NDIS中间驱动程序

中间驱动程序是典型的处于小型端口驱动程序和传输协议驱动程序之间的驱动程序。

中间驱动程序通过ProtocolXxx函数让NDIS调用来完成与底层小型端口的通信请求。

中间驱动程序通过MiniportXxx函数让NDIS调用来和一个或多个上面的协议驱动程序进行通信

中间驱动程序通常用下面方式：

- 在不同的网络媒体中起翻译的作用。（将以太网和令牌环数据包映射给ATM数据包）
- 过滤数据包。
- 平衡数据包传送。

NDIS驱动程序

- 3 NDIS协议驱动程序
- 一个在NDIS驱动程序层级中处于最高级的网络协议像TCP／IP等各种协议。传输协议驱动程序分配、拷贝来自应用程序申请发送的数据包，协议驱动程序同时也提供协议接口来接收从下一个到达驱动程序中的包，一个传输协议驱动程序将接收到的数据传输给适当的客户端应用程序。

NDIS驱动程序的应用

- 目前计算机系统的互联基本上采用TCP／IP协议，由于历史和技术等原因TCP／IP在提出、实施并成为行业标准的过程中，基本上没有考虑数据传输的安全性问题。

利用NDIS来编写自己的网络驱动程序

1 可以不使用不安全的TCP/IP，而用一个自己的协议程序来代替，在协议中加入加密和认证功能。

2 利用NDIS开发中间驱动能够截获所有的网络数据包(如果是以太网那就是以太帧)。它不仅可用于实现个人防火墙，还可以用来实现VPN(虚拟个人网络)、NAT(网络地址转换)、PPPOverEthemet(基于局域网的点对点通信协议)以及VLAN(虚拟局域网)。Windows DDK提供了两个著名的中间层驱动例子：Passthm和Mux。开发人员可以在Passthru的基础上开发包过滤防火墙，Mux则实现了VLAN功能。

NDIS驱动程序的应用

- 对于一个基于TCP／IP协议的网络系统，其数据在网络传输过程中，数据包易从网络上截获、篡改，仿冒。可以专门设计用于满足应用系统安全的网络驱动程序，该网络驱动程序可以采用NDIS驱动程序的形式。可在通信双方节点网络设备的NDIS层中对于待发送和截获的数据包施加高强度的加密、解密算法和认证算法，进行加密、解密和认证的处理。

实验环境

- DDK(驱动程序开发包）
  for windows2000

编译环境默认选用vc

安装DDK后，有checked和free两个编译驱动程序。

实验环境

- DDK安装后，可以在DDK的安装目录下看到若干子目录，其中bin子目录下存放的是可执行文件，help子目录下存放着关于DDK的各种帮助文件，inc子目录下存放的是编译驱动程序所需的各种头文件，src子目录下存放的是各种驱动程序例子的源代码等。
- DDK在src子目录下包含了几十个真实的驱动程序的源代码

实验环境

- 驱动程序的编译

驱动程序的编译由DDK提供BUILD工具控制，该工具位于DDK的bin子目录下。一

般而言，在编译一个驱动程序时需要在源文件的目录下提供三个文件：makefile、sources

和dirs。这三个文件均没有扩展名。BUILD根据这三个文件的内容对目标驱动程序进行编

译，并且创建build．10g、build．err等文件作为输出。如果正常，执行BUILD的结果是创建

驱动程序的可执行版本，其文件类型是．sys。

实验环境

- 顺次选择“开始”—>“程序”—>Developmentkits->Windows 2000DDK->CheckedBuildEnvironment，将出现一个控制台窗口。使用cd命令进入D：\programsLNTDDK\src\network\ndislpacket\driver目录(即驱动程序所在的目录)，键入build命令开始编译

NDIS协议驱动程序设计

- 1．NDIS接口与层次

NDIS提供了四层软件接口，通过这四层接口我们可以很好地了解NDIS协议驱动的工作原理

这四层接口分别为：

. 传输驱动程序上边界接口。

. 传输驱动程序下边界接口。

. 网络接口卡上边界接口。

. 网络接口卡下边界接口

- NDIS驱动程序通常需要考虑以下几个问题

驱动程序初始化。

数据传输处理。

数据包消息接收。

中断处理。

传输终结处理

- 2 函数

ProtocolXXX各种协议函数

- 3．数据包的组织与管理

在NDIS中，数据的接收与发送都是以数据包为单位进行的。

数据包由以下几部分组成：

- 一个数据包描述符：

其所含信息包括整个数据包所占用的物理页面的数量、数据包的长度、指向第一个和最后一个缓冲区描述符的指针以及数据包池的句柄等等。

- 一组缓冲区描述符：

每个缓冲区描述符用来描述一片存储区域，其中包括起始虚拟地址、偏移量、该存储区域的大小以及指向下一个缓冲区描述符的指针等信息。

- 由缓冲区描述符所描述的虚拟存储区域，该区域可能横跨几个页面。这些页面最终被映射到物理内存中。

- NdisAllocateBufferPool：请求分配一个缓冲池
- NdisAllocatePacketPool:请求分配一个数据包描述符
- NdisChainBufferAtBack:链接一个或是多个缓冲区描述符
- NdisAllocateMemory:得到缓冲区描述符所描述的存储区域

NDIS协议驱动设计思想

- 1．网络上截获数据包的思想

每当上层应用程序请求读取数据时，就为该请求生成一个相应的IRP，并将此IRP置于一个读取等待队列之中。当NIC从网络上接收到数据包时，由NDIS负责调用相应的接收函数。接收函数从读取等待队列的首部取出一个IRP，并将从网络上接收到的数据包拷贝到由该IRP所指定的缓冲区中。至此，上层应用程序便成功接收到了它所需要的数据。那么程序大体是这样的，当用户需要一个包时，调用一个上层函数进行读取，下层接到这个读请求，把包送到上层

- 2．解决丢包的策略

如果上层应用程序一直没有读取数据的请求，那么NIC从网络上接收到的数据包就会直接被丢弃。要想解决丢包这一问题，就必须对接收方式进行彻底的改变。改造后的程序的设计思想主要是增加了缓冲区以及对该缓冲区实施有效的管理。协议驱动程序负责维护一个接收缓冲区，该缓冲区以队列的形式组织。当NIC通知NDIS已从网络上接收到数据包时，我们可以先将这些数据缓存起来。

当上层应用程序需要读取数据时，该读操作的数据源不是直接从网络得到，而是经过有效管理的存放着数据的缓冲区。这样，丢包的问题便得以解决。

- 如果上层应用程序还是一直没有读取数据的请求，或者上层应用程序处理数据的速度低于NIC从网络接收数据的速度，再或者网络出现峰值流量时，缓冲区自然会逐渐被填满，最终还是会出现丢包的情况。但是在正常情况下，这种增设缓冲区的方法已经足以避免丢包情况的发生了。
- 实际编程中，正是通过增设缓冲区的方法解决了丢包问题，把数据包先存起来，当发现读请求时上交所有存在缓冲区中的数据包，然后清空。

NDIS协议驱动重要功能的实现

- 1．绑定

PacketBindAdapter函数是一个驱动程序中必不可少的函数，它是操作系统自动调用的函数。

在驱动程序安装的时候，操作系统自动查找当前机器上安装的网卡驱动程序(NIC Driver)，并对每一个网卡自动运行这个函数，其目的是在驱动程序中对每个网卡进行记录，以备以后使用。

这个函数中的主要内容就是实现对网卡的登记注册。

- 2．I／O控制

这部分的功能是由PacketIoControl函数实现的。

程序开始，首先确定控制类型。由于用户态和核心态程序的信息交换都是通过IRP来实现的，

所以这个控制类型是存放在IRP的堆栈段中。因为我们要实现无丢包的读取，所以这个函数必须实现以下5个方面的I／O控制：

- 枚举网络适配器。
- 记录上层用户是否发出读请求。
- 设置缓冲区大小。
- 重置适配器。
- 实现NDIS请求

- 3．数据接收

以协议驱动程序将NIC从网络上接收到的数据包递交给上层请求读取数据的应用程序，分析一下协议驱动程序在整个过程中所起的作用，以及各相关函数的工作流程。

注意的是以下提到的函数名称均为实际程序中的函数，是PacketXxx的形式，而不是标准的ProtocolXxx的形式。

- NIC从网络上接收到数据包并通知NDIS时，NDIS将调用PacketReceiveIndicate或PacketReceivePacket作为接收处理函数将NIC从网络接收到的数据缓存起来。
- 下面的工作是构造一个数据包，用来接收NIC从网络上接收到的数据包。

- 第一步先调用NdisAllocatePacket从PacketPool里请求分配一个数据包描述符。将接收到的数据包组织成队列形式的缓冲区。
- 第二步由HeaderBufferSize和patcketSize相加得到整个数据包的大小，并通过NdisAllocateMemory为其分配存储空间。

- 第三步调用NdisAllocateBuffer请求分配一个缓冲区描述符来描述刚刚通过调用NdisAllocateMemory而得到存储空间。
- 最后一步是将该缓冲区描述符链接到先前得到的数据包描述符上。

至此，一个完整数据包就组装好了，下面的工作就是数据传输了。

- 从以太网上接收到的数据包被分成包头和数据两部分分别传输
- NdisMoveMappedMemory函数负责把包头部分拷贝到前面组装好的数据包之中，之后再调用NdisTransferData将数据部分拷贝过来。
- PacketTransferDataComplete函数的实现非常简单，如果先前调用NdisTransferData传输数据成功完成的话，则将数据包插入缓冲队列中，等待上层应用程序读取。

- 当NIC从网络上接收到数据包后，NDIS通过调用PacketReceiveIndicate或者PacketReceivePacket已经将数据包缓存起来。下面就要等待上层应用程序从缓冲区中读取数据了。
- 上层发出一个读的请求，I／O管理器在接收了该I／O请求之后，分配并初始化一个IRP。构造IRP的重要依据就是上层应用程序在调用ReadFile时所传入的参数。

- 因此当上层应用程序调用ReadFile时，该I／O请求将交由Packetread函数处理。PacketRead函数负责将缓冲队列中的数据包全部取出，捆绑后一并上交。之所以将缓存的数据包

  一并上交，而不是上层应用程序每调用一次ReadFile上交一个数据包，主要是出于以下考虑：一方面，前面已经提到将所有数据包一并上交可以立即清空缓冲区，供NIC从网络上新接收的数据包使用，减小丢包的可能；另一方面，I／O操作的速度相对较慢，因此在上层应用程序中应尽量减少I／O操作的次数。

上层应用程序

- 前面介绍了下层的驱动程序的设计思想具体实现，但是光有下层的程序并不能完全实现我们需要的功能。为了让一个软件可用，我们必须有在用户态下的应用程序为用户提供方便的服务。上下层之间的接口这时就显得尤为重要

协议驱动的接口

- 动程序上下层之间的接口实际上就是主功能代码。在这个程序里主要用到了五个主功能代码，分别是IRP_MJ_CREATE、IRP_MJ_CLOSE、IRP_MJ_READ、IRP_MJ_WRITE、IRP_MJ_DEVICE_CONTROL，分别对应于Win32 API的CreateFile、CloseHandle、ReadFile、WriteFile和DeviceIoControl。上层通过调用这5个Win32API可以告诉下层该干什么，干些什么。

- 在Windows中，设备是被当成一个文件进行操作的，所以我们可以调用这些API实现对设备

的打开、关闭、读写和其他通信。CreateFile和CloseHandle的原型比较简单，实现的就是打开或者关闭的功能，主要工作是得到设备的句柄或者释放该句柄；而ReadFile和WriteFile则实现了读写的功能

DeviceIoControl是一个需要特别说明的函数，该函数的人口参数中有三个比较重要，分别是DWORD dwIoControlCode、LPVOID lpInBuffer和LPVOID lpOutBuffer。第一个是控制码，它是一个直接传递给下层设备的常量。驱动程序通过不同的dwIoControlCode进行不同的处理；后两个是两个缓冲区，一个是下层读取的输入缓冲区，一个是下层处理以后返还给上层的输出缓冲区，通过这两个缓冲区和一个控制码就可以通知下层该干什么，从哪读，往哪写。

- 数据包的解析

从下层截获的数据包，如果不经过任何处理，将是原始的比特流，这对用户是没有任何意义实际上我们要对这些二进制数据进行解析，才能知道网络上的数据包的具体情况。

- 1．解析依据

我们对包进行解析的依据是TCP／IP协议，当然，我们不关心协议中有关差错控制等方面的描述，我们关心报文结构。

- 1 数据链路层：负责收集物理层的原始比特流形成更大的集合称为帧。

图3-3　数据链路层帧头结构

- 我们将要解析出目的地址和源地址，然后根据类型的不同做不同的处理。类型域主要有三类：IP包(0800)，ARP包(0806)，RARP包(8035)。其他类型的包出于程序简单的需要不再进行解析。ARP和RARP是定长的，数据域长28字节，然后是18字节的填充。需要注意的是，因为我们实际不是从最下层直接截获的比特流，而是经过了NDIS，所以实际的包中没有了CRC(循环冗余校验码)，这部分已经被NDIS滤掉了。

- (2)网络层。在这里我们将根据IP报头进行解析

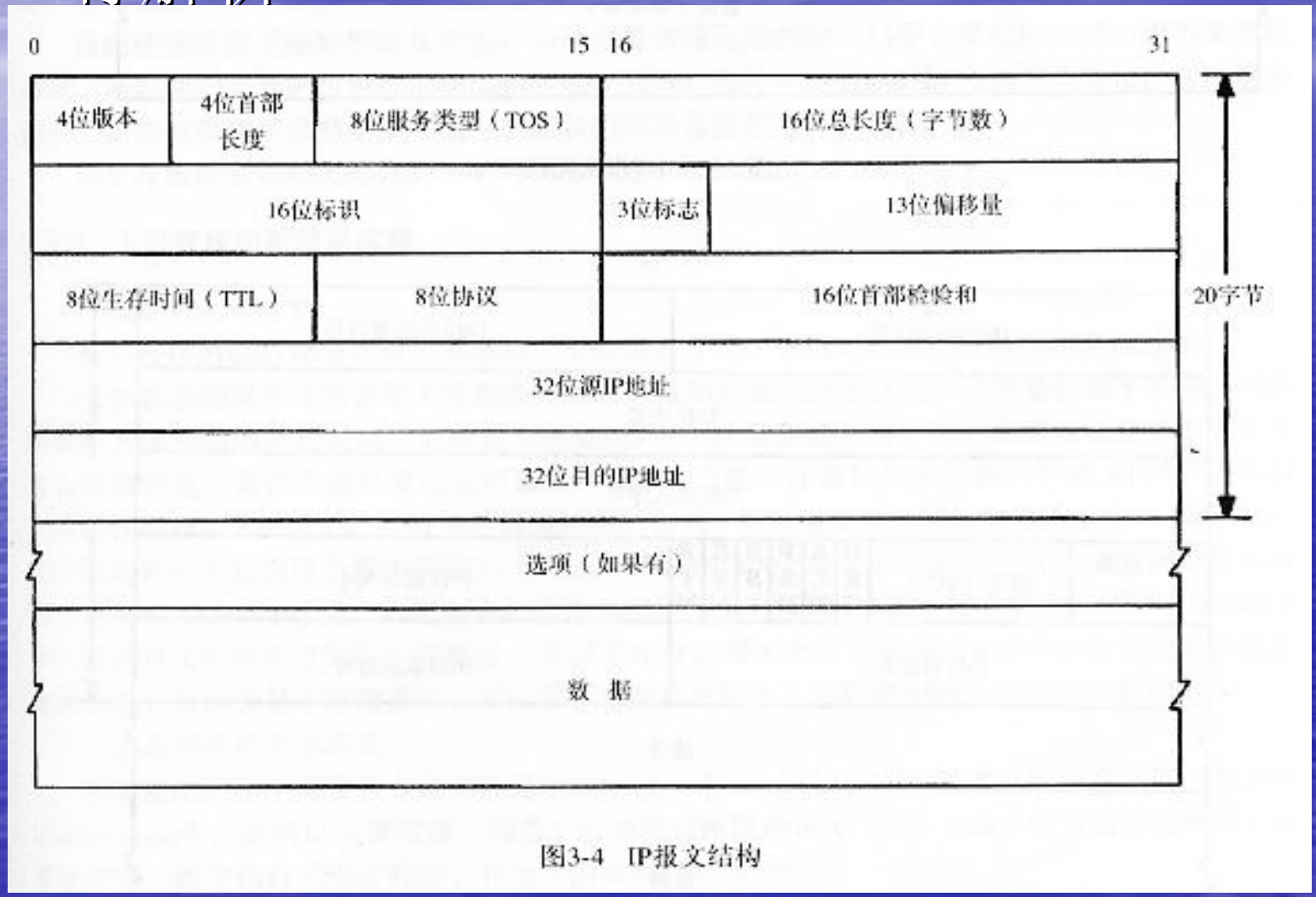


图3-4 IP报文结构

- 这里我们已经不再针对ARP和RARP进行分析(这两者的解析仅限于数据链路层)，我们的对

象限于IP报文。在这个层次上我们将要解析出的内容主要包括目的IP地址、源IP地址和包的长度。需要注意的是ICMP(Internet控制消息协议)和IGMP(Internet组管理协议)是封装在IP数据报中的，所以这两种类型的报文需要根据从IP报头开始计算的第10个字节来区分。至于具体的ICMP和IGMP的报文形式，这里不再详述。

-

- (3)传输层。在这里我们只需要注意TCP(传输控制协议)和UDP(用户数据报协议)的区分，而需要解析的内容只有端口，应该说是很简单

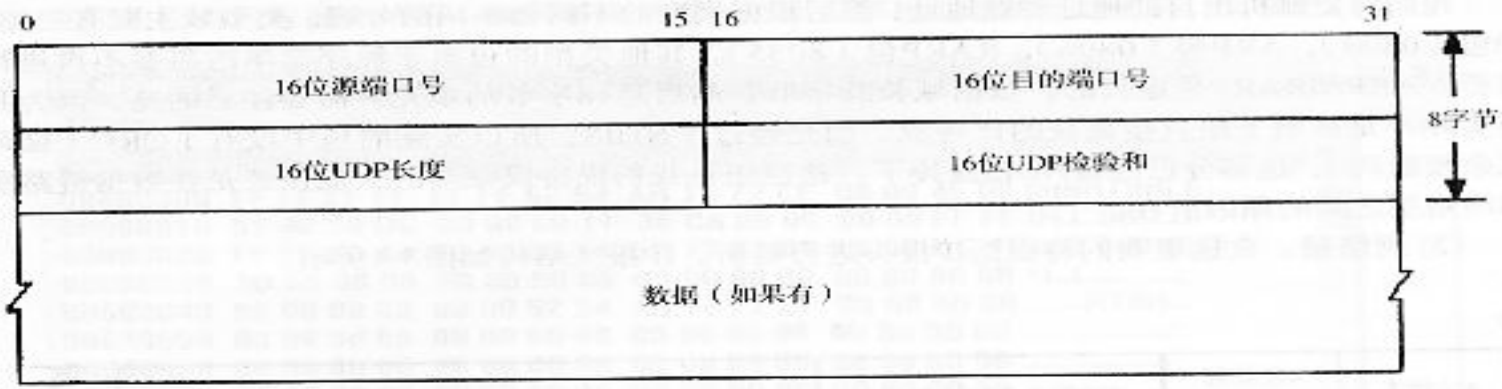


图3-5 UDP报文结构

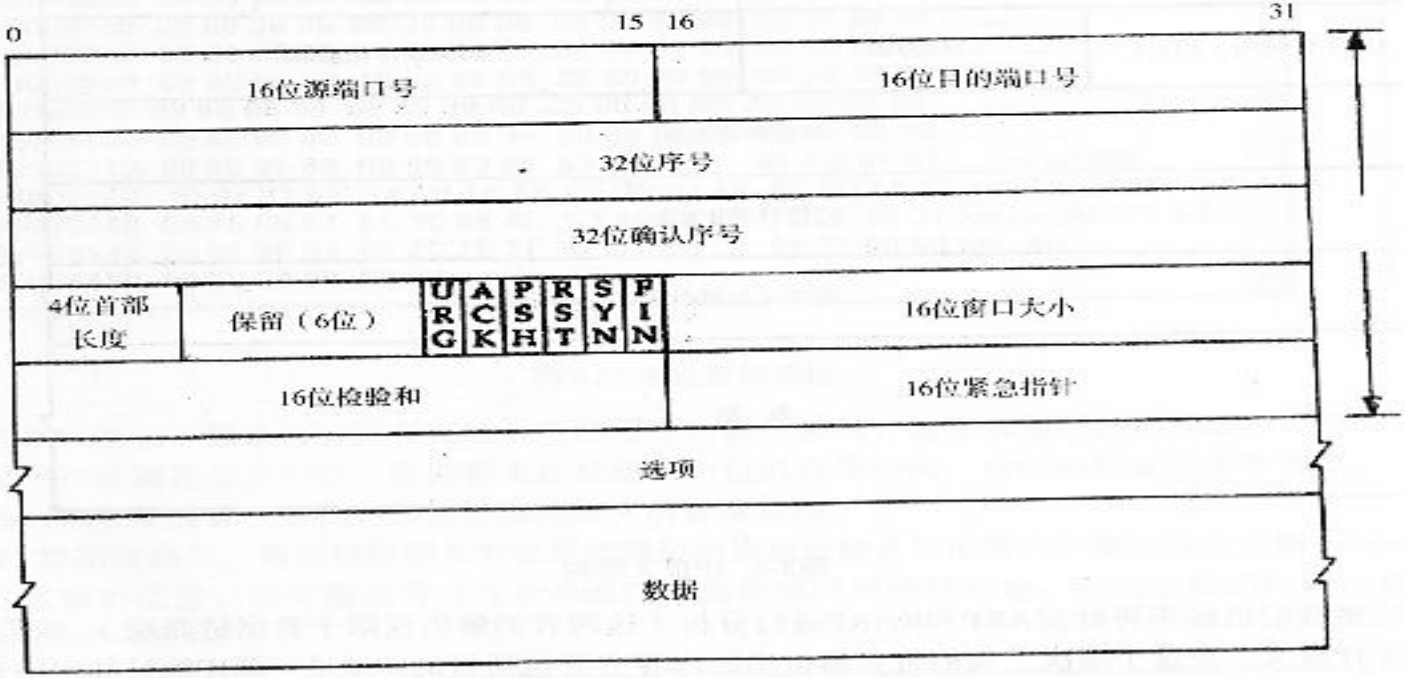


图3-6 TCP报文结构

解析实现

- 把数据包的13，14字节存起来，并对其进行判断，实际就是看包是否是我们需要解析的ARP或者IP报文。如果不是，我们仍然要把这个数值存起来，因为这时这两个字节将是包的长度(显然，这种包不是以太网的标准结构)。下面我们分情况讨论。
- 　如果是ARP包，我们将从后面的字节中提取出源端和目的端的以太网地址和IP地址，并且指针自加指向下一个数据包的头(ARP包定长为60字节)。

如果是IP包，我们又需要进一步分情况讨论。有可能是ICMP、IGMP、UDP或者TCP，这些

- 都将通过从IP报头开始计算的第10字节体现出来。ICMP是01，IGMP是02，TCP是06，而UDP是17，所以我们用一个数组IPType来存这个数值。我们注意到，TCP和UDP中，端口是一个比较重要的参考数据，所以我们用两个数组SouPort和DesPort存储端口号。
-

- 我们还要注意不能解析的几个包：一种是前面提到的猫13、14字节是包的长度，而不是协议信息，所以我们需要把指针往后移动相应的字节数；另外一种是以886F作为开头的包，这种包在截获上来的时候将填满整个缓冲区，所以我们应该直接把这种包扔掉

上层程序的原理与实现

上层对包的读取

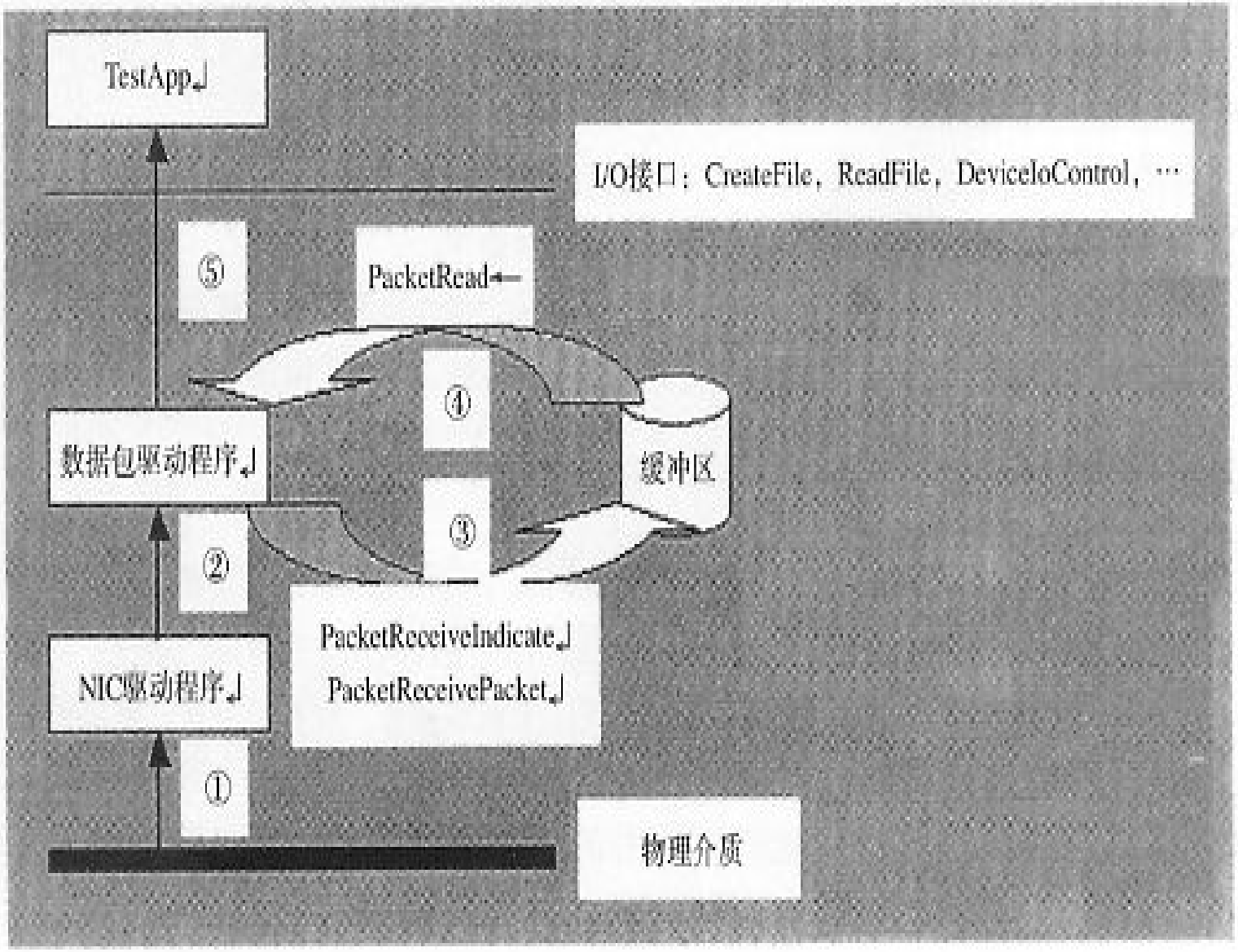
当加载驱动程序这些必要工作都做完以后，按用户请求开始读包。因为要保证不丢包，所以采取的方法是每隔一段时间(目前设定的是2秒钟)从设备读一次，这一次的读取要求下层上交所有存储的包。具体实现是采用定时器SetTimer的方法，让窗口每次接收到WM_TIMER消息时就执行读操作。

- 本章深入分析了NDIS驱动程序的原理和数据包传输的机制，设计了一个比较健壮的NDIS协议驱动程序和用户态应用程序，为学习Windows操作系统原理的有关人员提供了一个完整的实习例子。在本章提供的例子的基础上，还可以进一步实现更加复杂和更有实用价值的应用，例如，网络安全、网络流量监控等。

下层由于已经改为开辟缓冲区的方式，所以递交到上层的包实质上是一个大数组。

程序调用ReadFile把这个数组读进一个预先开辟好的缓冲区A，之后对这个大数组进行定界和解析(只解析ARP和IP报文，而把剩余的数据包都扔掉)，把解析结果字符串放入另外一个缓冲区B中。缓冲区A在每次读完后立即释放，所以不用开辟得太大，而缓冲区B则存放所有已经读到的解析信息，其内容是不断增多的，所以需要开辟得比较大。然后把B中的内容显示给用户。

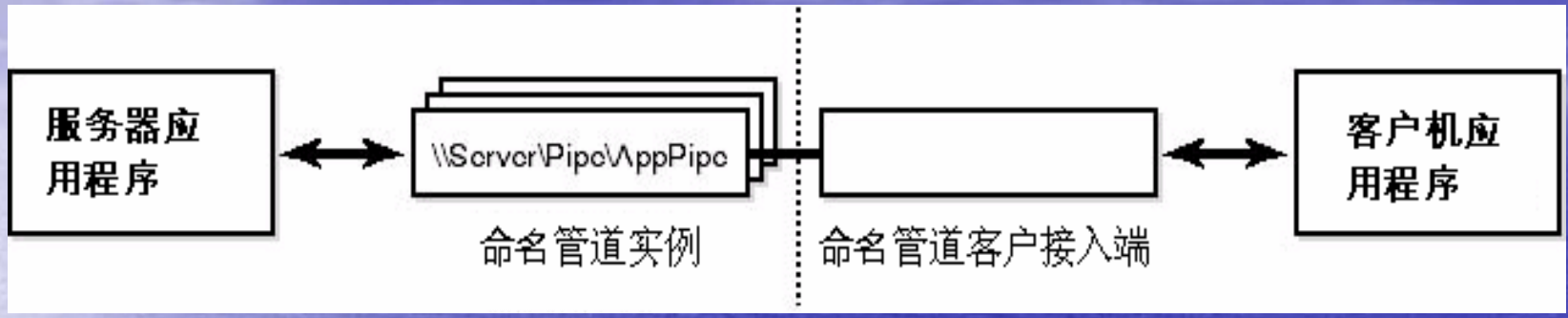
本章深入分析了NDIS驱动程序的原理和数据包传输的机制，设计了一个比较健壮的NDIS协议驱动程序和用户态应用程序，为学习Windows操作系统原理的有关人员提供了一个完整的实习例子。在本章提供的例子的基础上，还可以进一步实现更加复杂和更有实用价值的应用，例如，网络安全、网络流量监控等。


TestApp
I/O接口：CreateFile，ReadFile，DeviceIoControl，…
⑤
PacketRead
④
数据包驱动程序
缓冲区
③
②
PacketReceiveIndicate
PacketReceivePacket
NIC驱动程序
①
物理介质

2. Windows 2000网络体系结构

- 网络API
  - 命名管道（Named Pipe）和邮件槽（Mailslot）
  - Windows套接字（ WinSock ）
  - 远程过程调用（RPC）
  - 通用互连网络文件系统（CIFS）
- 网络资源的名字解析
- 协议驱动程序
- NDIS驱动程序

命名管道（Named Pipe）



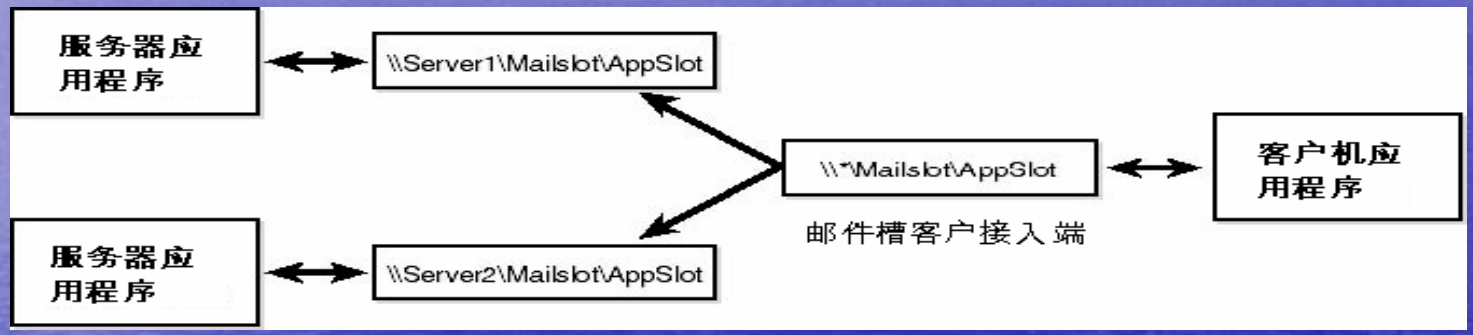
服务器：

- 创建命名管道
  CreateNamedPipe
- 建立连接
  ConnectNamedPipe
- 使用命名管道
  ReadFile
  WriteFile

客户：

- 连接服务
  CreateFile
  CallNamedPipe
- 使用命名管道
  ReadFile
  WriteFile

邮件槽（Mailslot）



服务器：

- 创建邮件槽

  CreateMailslot
- 使用邮件槽

  ReadFile

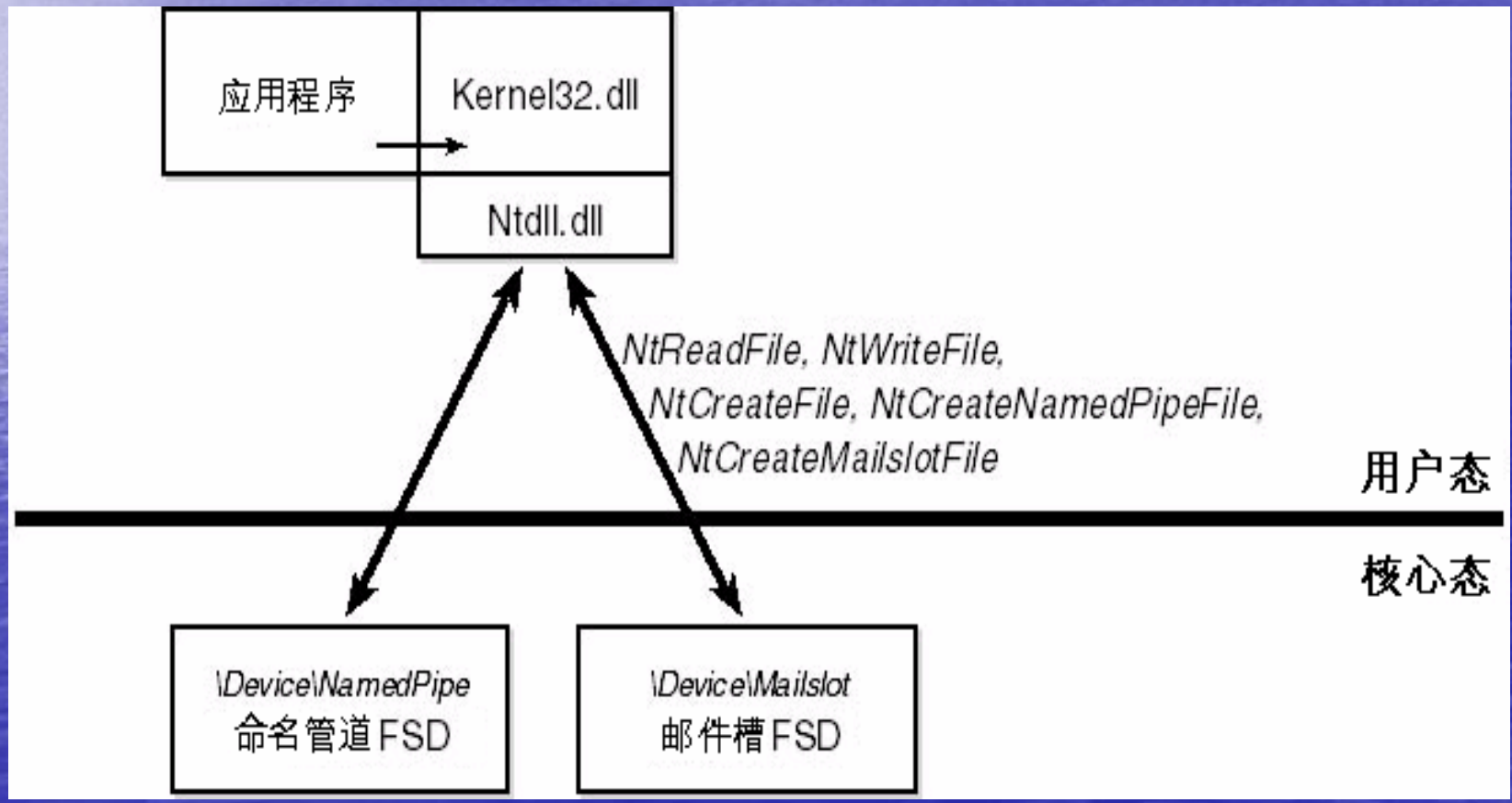
客户：

- 连接服务

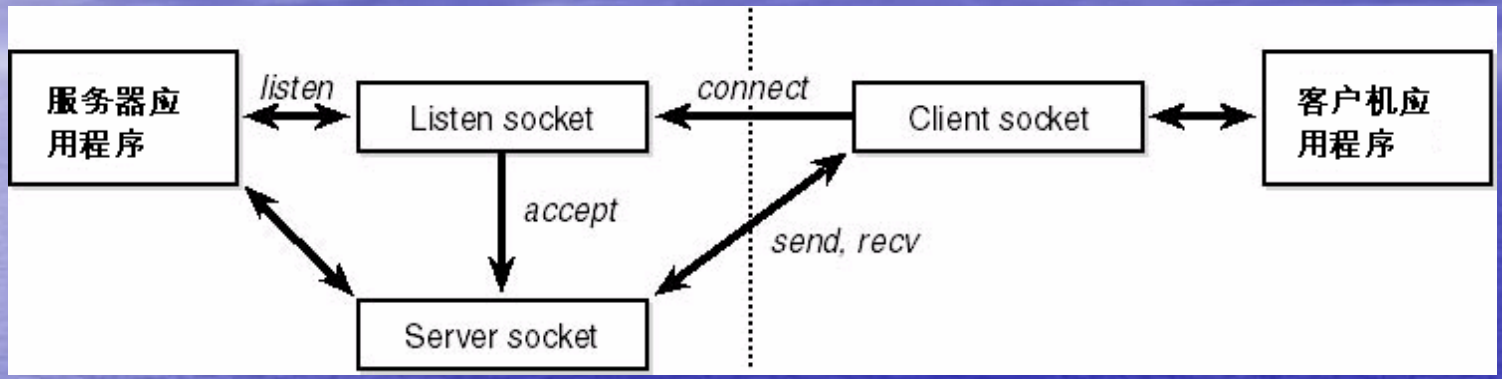
  CreateFile
- 使用邮件槽

  WriteFile

命名管道和邮件槽的实现



Windows套接字（Winsock）



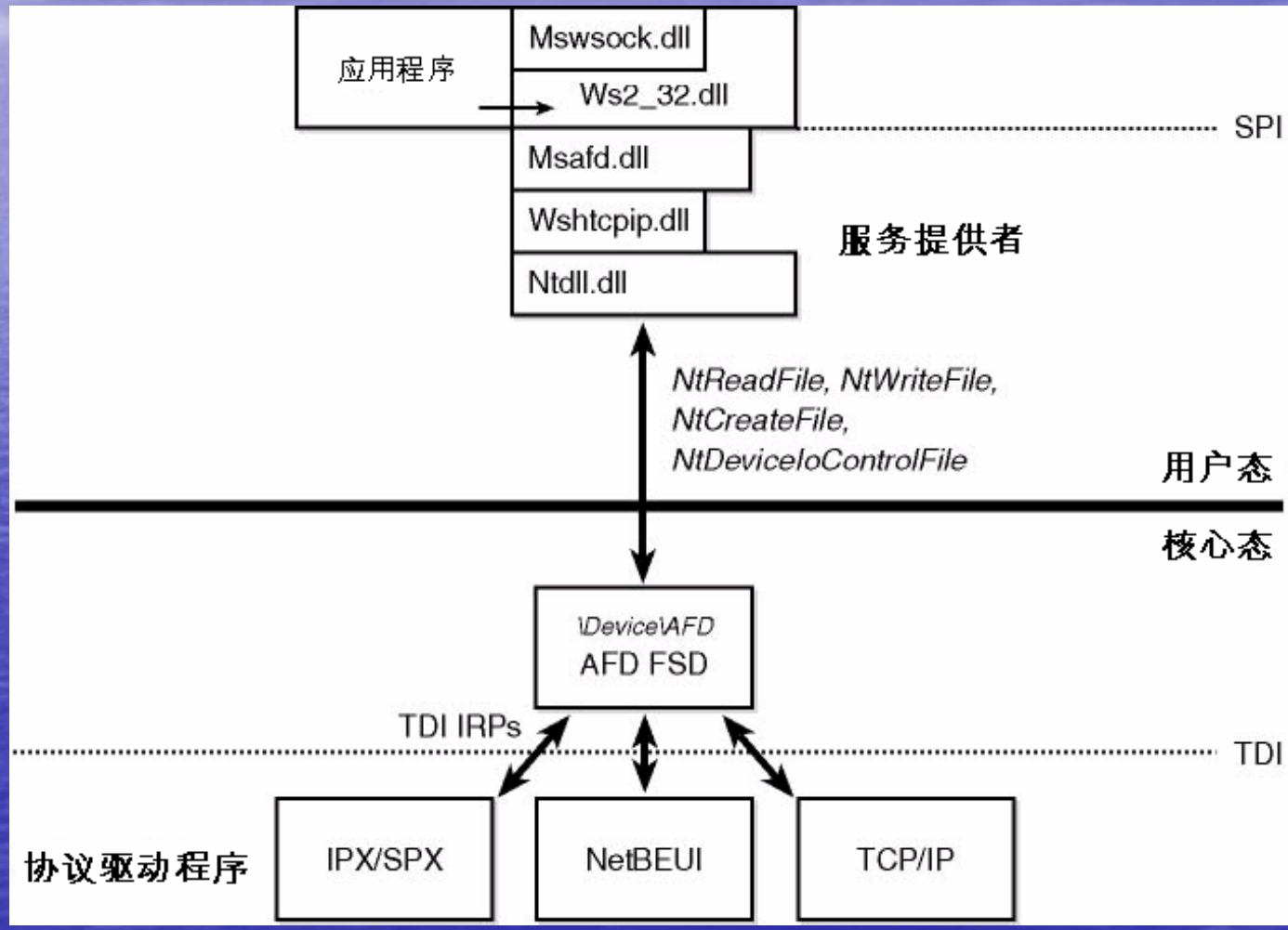
服务器：

socket
bind
listen
accept
read,recv
write,send
closesocket

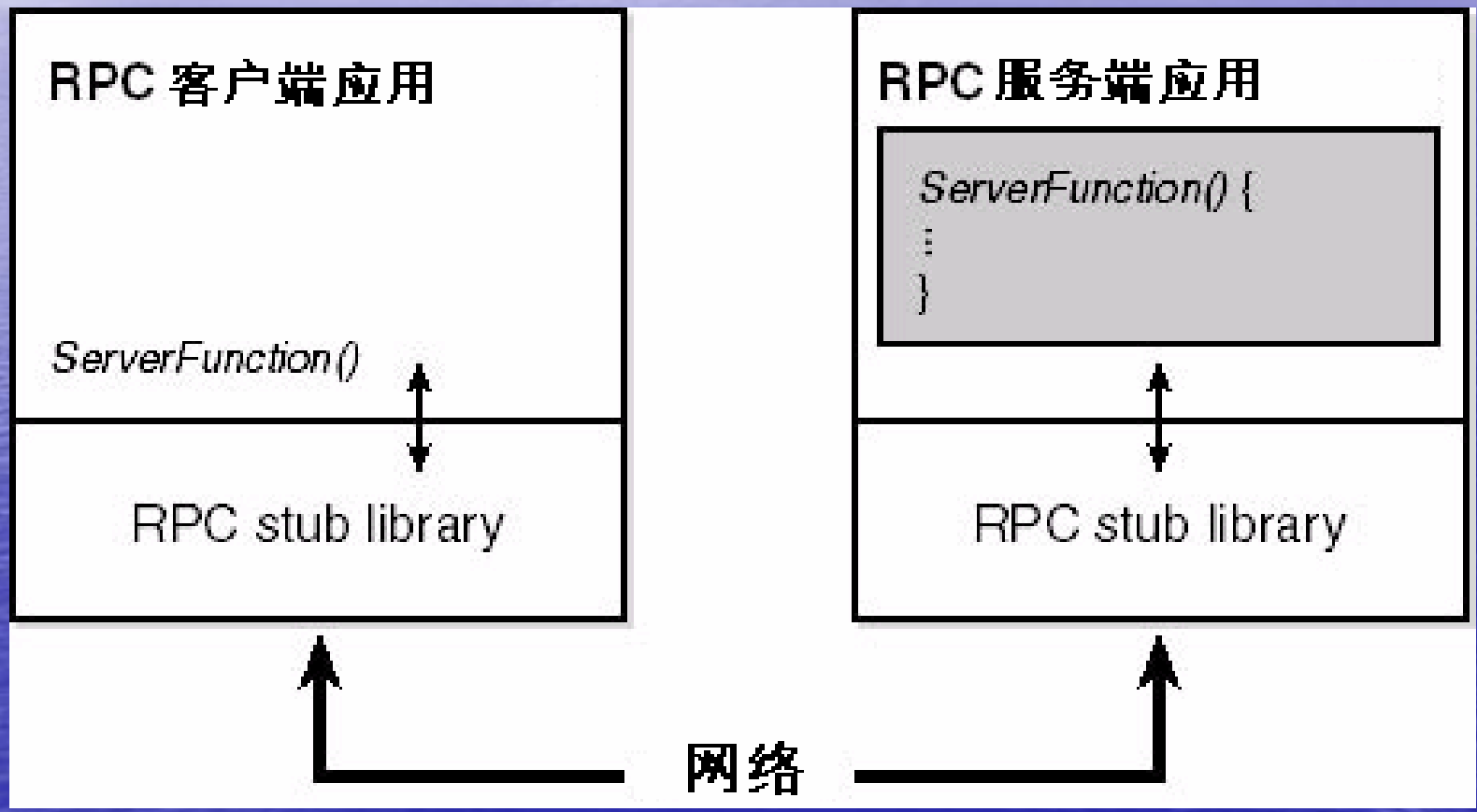
客户：

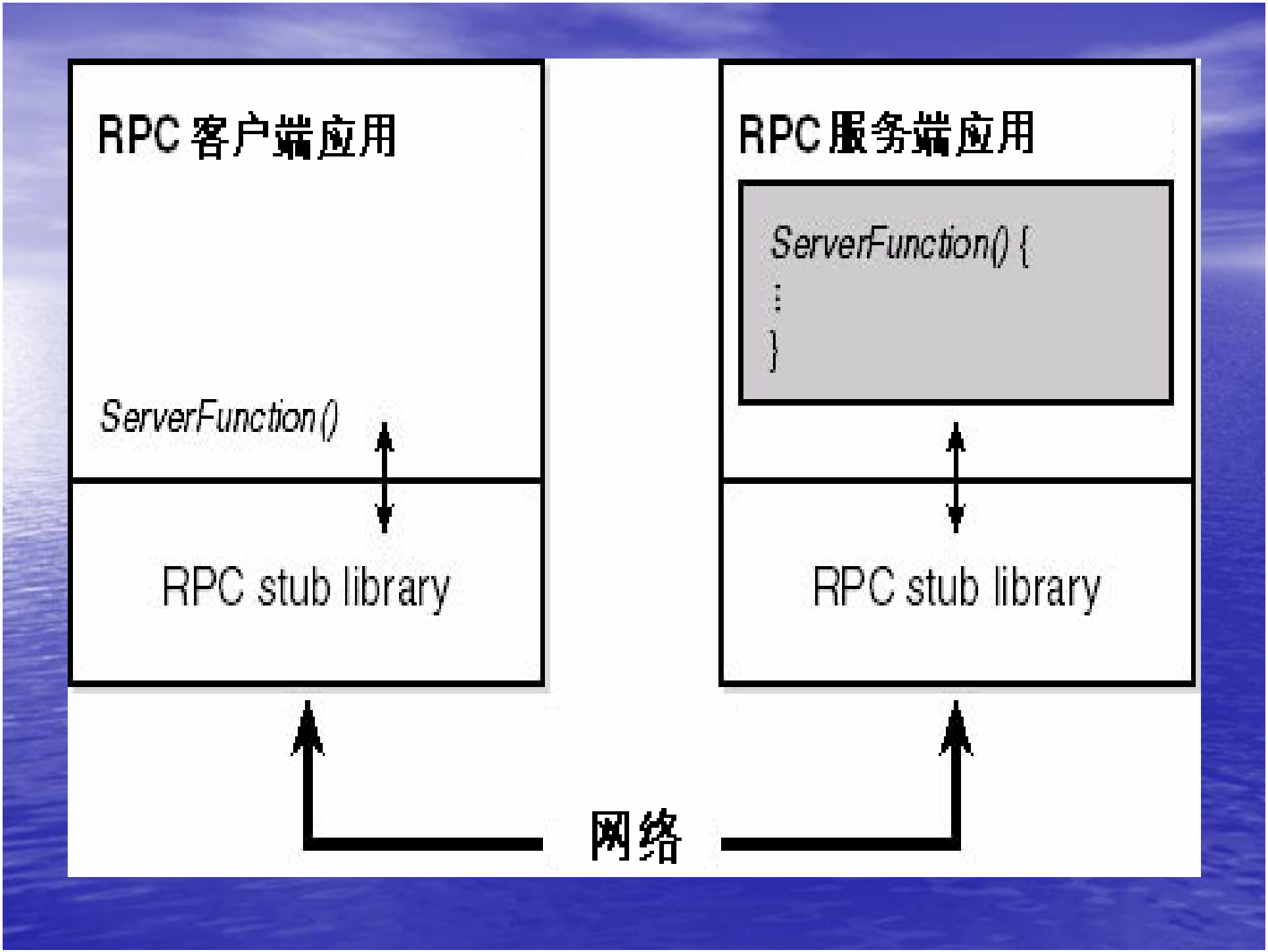
socket
connect
write,send
read,recv
closesocket

Winsock 的实现

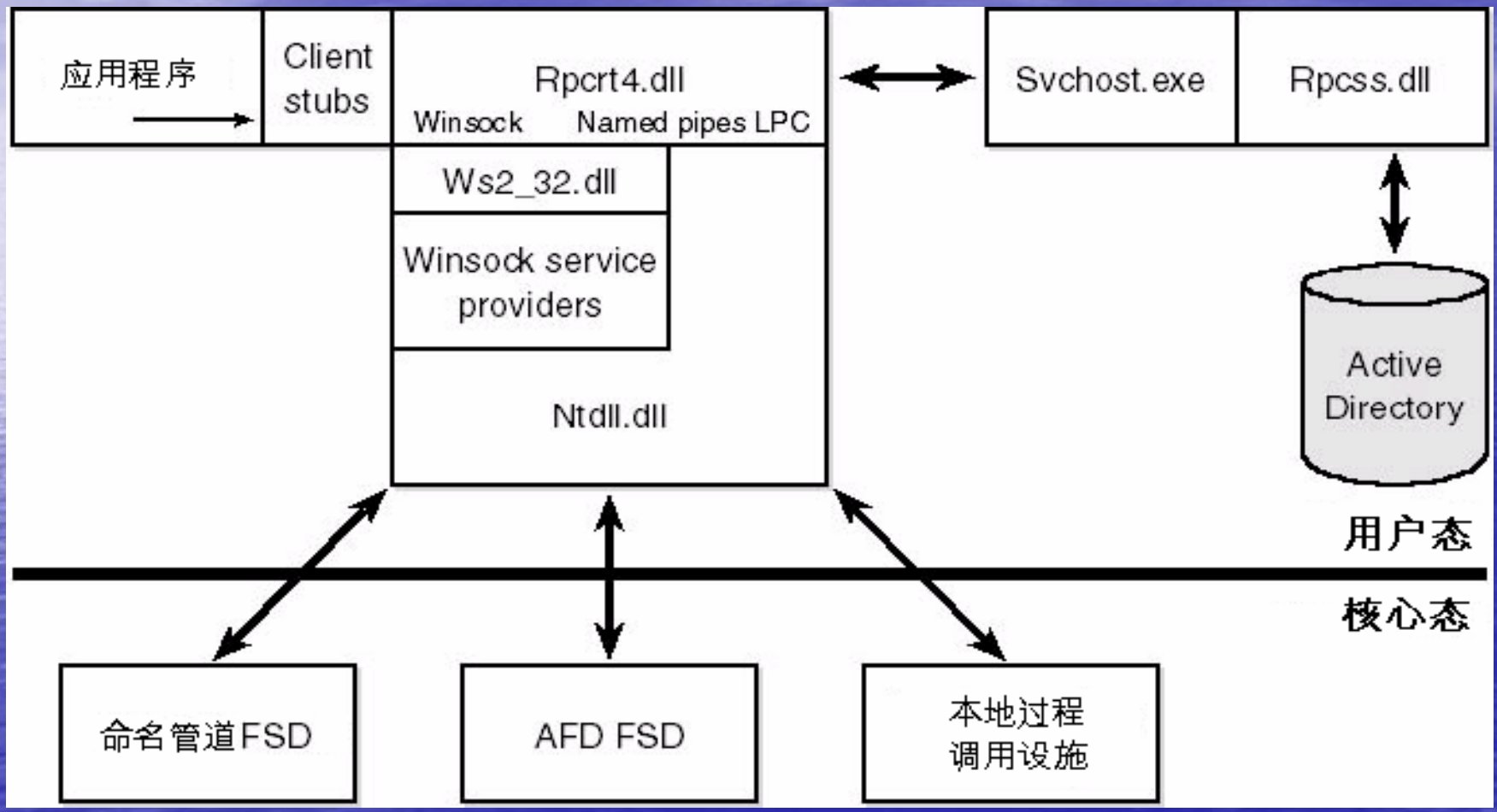

应用程序
Mswsock.dll
Ws2_32.dll
Msafd.dll
Wshtcpip.dll
Ntdll.dll
SPI
服务提供者
NtReadFile, NtWriteFile,
NtCreateFile,
NtDeviceIoControlFile
用户态
核心态
\Device\AFD
AFD FSD
TDI IRPs
TDI
协议驱动程序
IPX/SPX
NetBEUI
TCP/IP

远程过程调用（RPC）


RPC 客户端应用
ServerFunction()
RPC stub library
RPC 服务端应用
ServerFunction() {
:
}
RPC stub library
网络


RPC 客户端应用
ServerFunction()
RPC stub library
RPC 服务端应用
ServerFunction() {
}
RPC stub library
网络

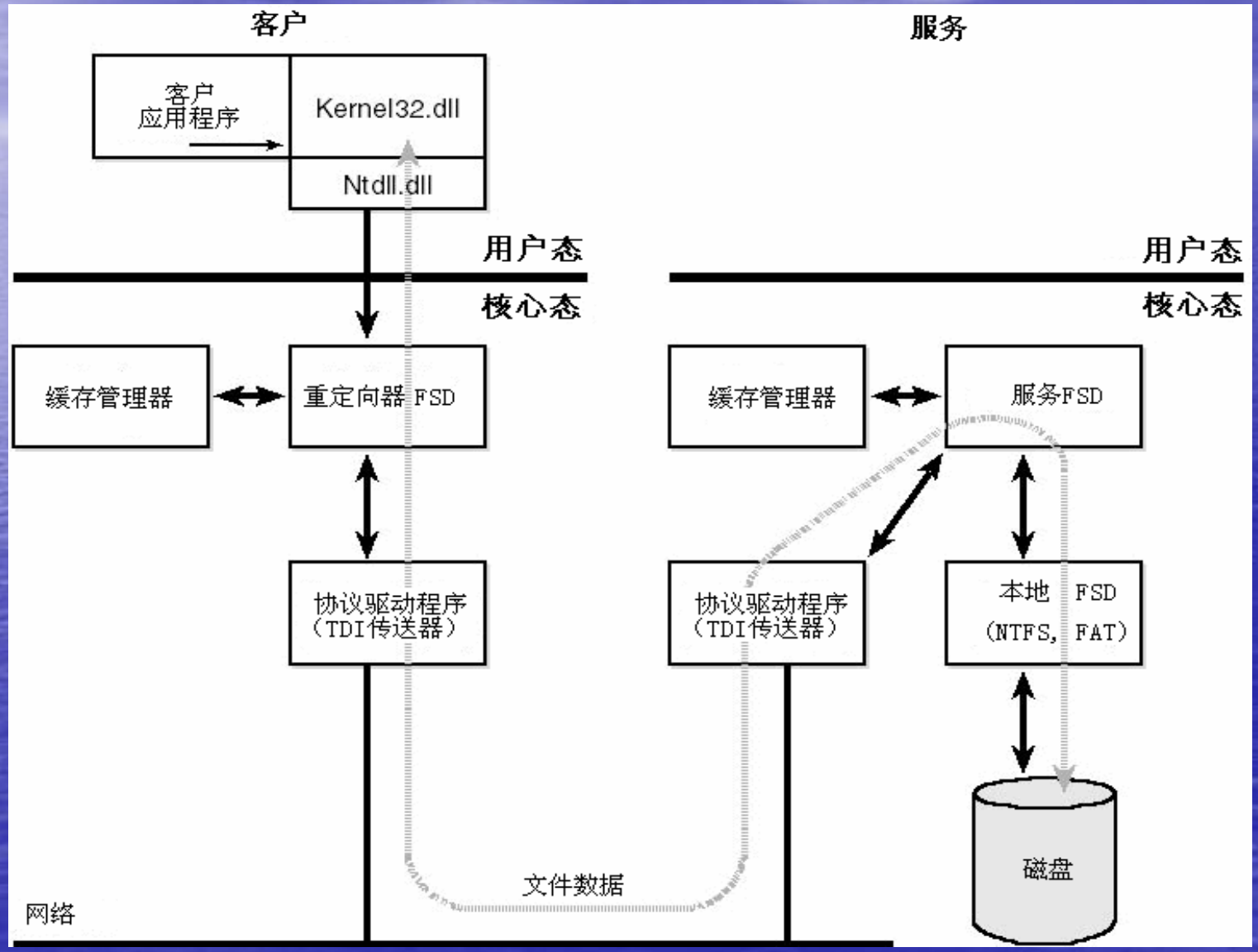
RPC 的实现


应用程序
Client stubs
Rpcrt4.dll
Winsock
Named pipes LPC
Ws2_32.dll
Winsock service providers
Ntdll.dll
Svchost.exe
Rpcss.dll
Active Directory
用户态
核心态
命名管道FSD
AFD FSD
本地过程调用设施

通用互连网络文件系（CIFS）

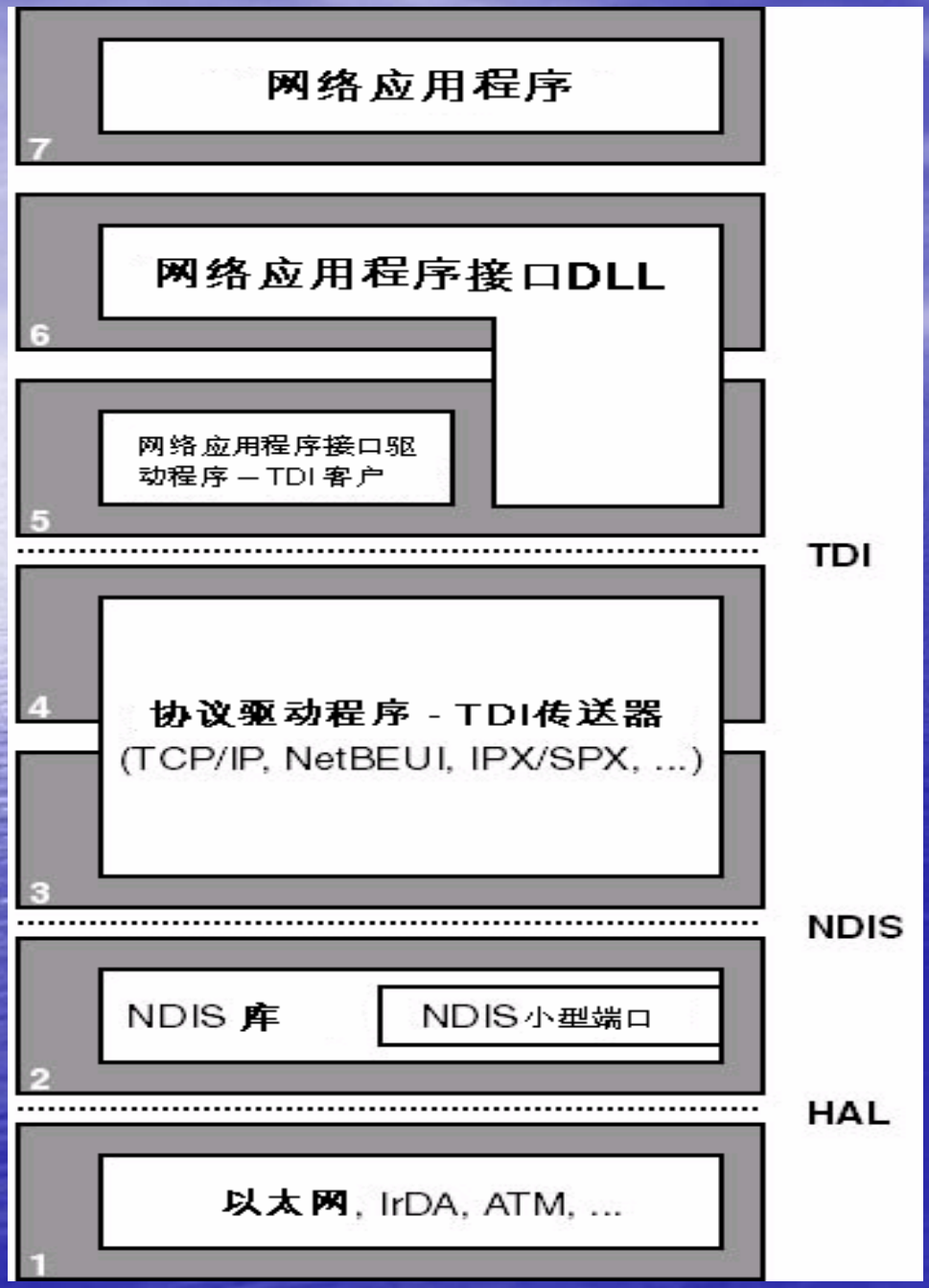
通用互联网文件系统（CIFS）是Windows 2000用于互联网文件共享的协议。应用程序通过标准的Win32文件I/O函数可以访问远程文件。

CIFS 的实现


客户
服务
客户
应用程序
Kernel32.dll
Ntdll.dll
用户态
核心态
用户态
核心态
缓存管理器
重定向器 FSD
缓存管理器
服务FSD
协议驱动程序
（TDI传送器）
协议驱动程序
（TDI传送器）
本地 FSD
（NTFS, FAT）
磁盘
文件数据
网络

网络资源的名字解析

应用程序可以通过两种方法查询和访问远程机器上的资源。一种是使用UNC标准，通过Win32函数直接访问远程资源。另一种方法是使用微软网络（WNet）API枚举所有计算机提供的可共享的计算机和资源。两种方法都使用重定向器访问网络上的资源。



协议驱动程序

- 网络API驱动程序接受API 请求，把它们转换为底层网络协议的传输请求。
- API驱动程序依赖核心态的运输协议驱动程序进行实际的转换。
- API和下层的网络协议是分开的，使得整个网络体系结构十分灵活，它允许每个API使用不同的网络协议。

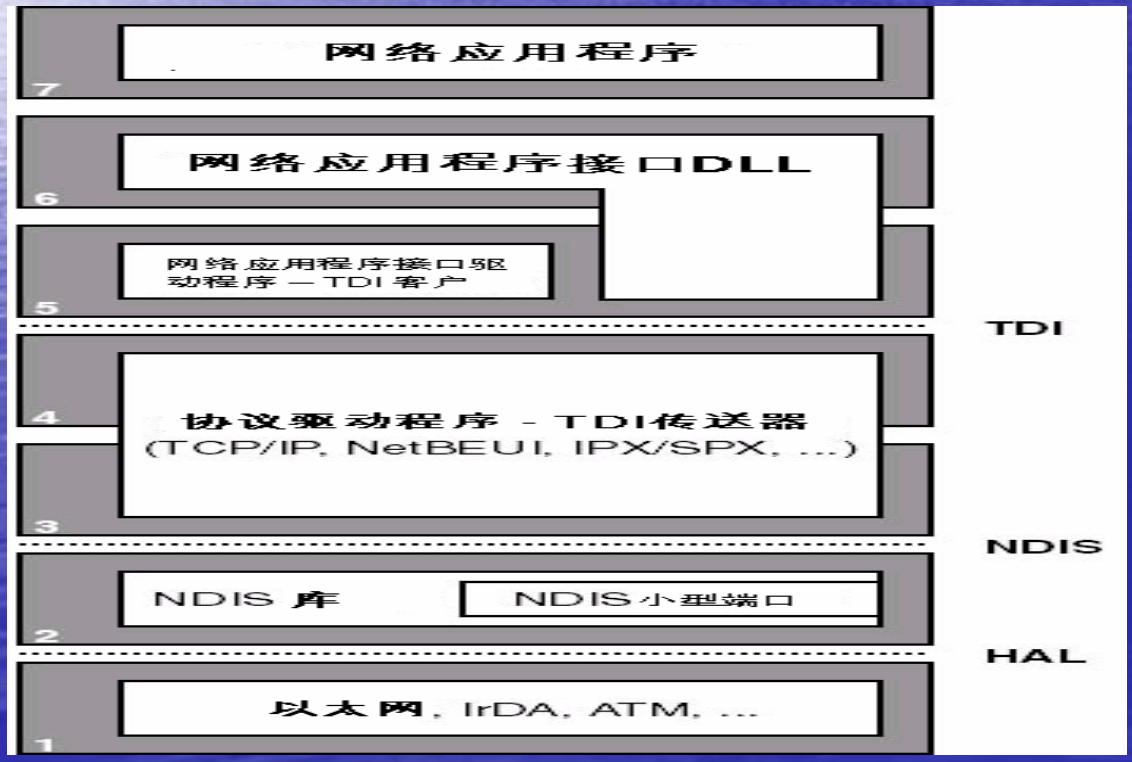
协议驱动程序

- DLC协议，一种相对原始的协议。IBM的一些大型机和HP的一些网络打印机使用这一协议。
- NetBEUI（NetBIOS Extended User Interface）协议，它和NetBIOS紧密集成在一起
- TCP/IP协议，有适于WAN的特征和在WAN上较好的性能。TCP/IP协议是Windows2000优先使用协议。
- NWLink协议，由Novell的IPX协议和SPX协议组成的。

NDIS驱动程序

网络驱动程序接口规范(NDIS）允许协议驱动程序以设备无关的方式和网络适配器驱动程序通信。

遵守NDIS的网络适配器驱动程序被称为NDIS驱动程序或NDIS小型端口驱动程序。



远程访问

Windows 2000允许远程访问的客户连接远程访问服务器并访问网络资源，例如文件，打印机，以及网络服务。这样，客户就好像与远程访问服务器的网络连在了一起。

Windows 2000的层次化网络服务

- 远程访问
- 活动目录
- 网络负载平衡
- 文件复制服务
- 分布式文件系统
- TCP/IP的一些扩展特性

活动目录

活动目录是Windows 2000 Server实现的轻量目录访问协议（LDAP）中的目录服务。活动目录能让客户在活动目录的数据库内访问对象。活动目录提供了一个安全、分布式、可扩充的目录服务。

网络负载平衡

Windows 2000 Advanced Server中的网络负载平衡允许建立一个可以多达32台计算机的集群，该集群维护一个虚拟IP地址，并公开给客户访问，客户的请求能够分布到集群中的所有计算机上去处理，以达到网络负载平衡的目的。

文件复制服务

文件复制服务通过对文件的复制来增强系统的可靠性，提高服务器的运行性能。

Windows 2000 Server文件复制服务(FRS)的主要功能是复制域控制器\SYSVOL目录的内容。当复制的目录或文件一旦发生改变，这些变化会广播到其他域控制器上。

分布式文件系统

分布式文件系统（DFS）服务将文件连入一个单一的名字空间，使文件能够在同一台或者多台不同的计算机上得到共享，而且DFS可以为客户提供位置透明的资源访问。

TCP/IP 的一些扩展特性

网络地址翻译（NAT）

互连网络协议安全性（IPSec）

服务质量（QoS）。