

# Inside NT Networking

Mark Russinovich

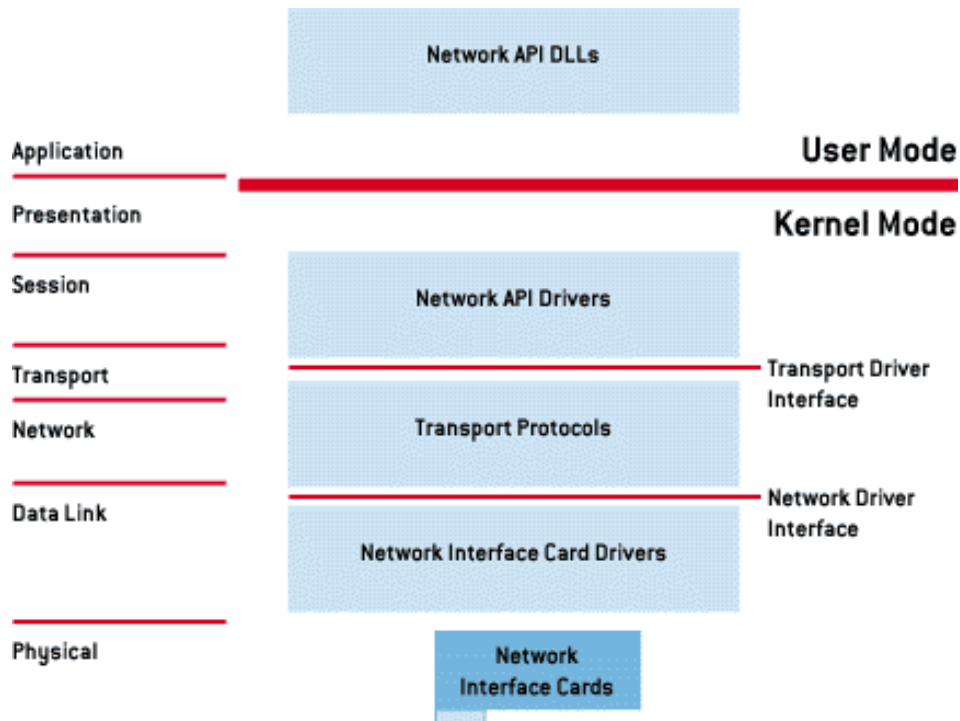
(Reprinted from WindowsItPro Magazine)

Microsoft learned a lesson when the lack of native networking support in DOS and Windows 3.0 limited those OSs' acceptance in business environments and left the networking market open for Novell and UNIX vendors. Microsoft strove to ensure that the first release of Windows NT contained an array of networking options out of the box. NT's debut in 1993, with version 3.1, fulfilled NT's promise as an OS that is ready for peer-to-peer and client/server networking configurations. NT 3.1 included a variety of popular network programming APIs, networking protocols, and a network card driver model that facilitated third-party support for Microsoft networking hardware.

This month, I'll peel away the layers of NT's network architecture to show you how this architecture implements network programming APIs. I'll describe how the APIs interface with protocol drivers and implement NIC drivers. Along the way, I'll introduce you to some new networking features that Windows 2000 (Win2K—formerly NT 5.0) will include.

## Network APIs

In 1978, the International Standards Organization (ISO) introduced the Open Systems Interconnection (OSI) model to standardize defining network communications in layers. The OSI model consists of seven network layers, which Figure 1, page 52, illustrates. Higher layers represent higher levels of abstraction. The idea behind network layering is that lower layers transform requests coming from above to a lower level of abstraction, and higher layers transform requests coming from below to a higher level of abstraction. The highest level of abstraction occurs at the application layer, in which network programming APIs such as remote procedure call (RPC) and sockets fall. The lowest level of abstraction occurs at the physical layer, in which bits and bytes on wires or cables transmit between computers. The division of networking into layers promotes a model of standardized interfaces between layers, and flexible and replaceable layer implementation.



In general, OS designers don't divide the network I/O path into seven well-defined layers, as the OSI model promotes. For simplicity and performance, OS designers can designate one layer in an OS's network architecture to correspond to several OSI layers. Figure 1 shows how NT's network

# Inside NT Networking

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

architecture maps to OSI layers. The highest of the three NT layers is the network API layer. A network API gives applications an interface to networking I/O. Application designers can define their own networking APIs, but relying on standard APIs is simpler. NT incorporates the following networking APIs: named pipes, mailslots, NetBIOS, Winsock, RPC, Network Dynamic Data Exchange (NetDDE), Server Message Block (SMB), Distributed Component Object Model (DCOM), and Message Queues.

Named pipes and mailslots are programming APIs that Microsoft originally developed for OS/2 LAN Manager, then ported to NT. Named pipes let programs on different computers establish a reliable 2-way communications link. A named pipe server calls the `CreateNamedPipe` function to define a pipe on the local computer, and a named pipe client opens a connection to the server's pipe by specifying the pipe's name to the `CreateFile` Win32 function. With the established connection, the client and server can use `WriteFile` and `ReadFile`, standard NT file I/O routines, to exchange data through the pipe connection. Named pipes are popular NT programming APIs, but applications that use pipes do not generally port to Windows 98 and Win95, because these OSs do not implement named pipe server APIs.

Mailslots give programs an unreliable, connectionless broadcast facility. Similarly to a named pipe server, a mailslot server defines a mail slot using the `CreateMailSlot` API. Mailslot clients can open the server's mail slot and send the server messages through the slot. The mailslot service is unreliable, because the server might not receive messages clients send. Mailslots also support message broadcast. In message broadcast, when two or more mailslot servers in a domain create a mail slot with the same name, those servers will receive the messages clients send to that mail slot name. What kind of application can make use of an unreliable broadcast mechanism? One example is a time-synchronization service, which might broadcast a source time across the domain every few seconds. Receiving the source-time message is not crucial for every computer on the network.

Until the 1990s, the NetBIOS programming API has been the most widely used programming API on personal computers. NetBIOS allows for both reliable-connection-oriented and unreliable, connectionless communication. NT supports NetBIOS for its legacy applications. Microsoft discourages application developers from using NetBIOS, because other APIs, such as named pipes and RPC, are much more flexible and portable.

Winsock is Microsoft's implementation of BSD Sockets, a programming API that became the standard by which UNIX systems communicated over the Internet during the 1980s and 1990s. Support for sockets on NT makes the task of porting UNIX networking applications to NT relatively straightforward. Winsock includes most of the functionality of BSD Sockets but also includes Microsoft-specific enhancements, which continue to evolve. Winsock has a mode it calls streams for reliable-connection-oriented communication, and a mode it calls datagrams for unreliable, connectionless communication.

RPC is a network programming standard that Sun Microsystems originally developed. The Open Software Foundation (now The Open Group) made RPC part of the distributed computing environment (DCE) distributed computing standard. An interesting aspect of NT's implementation of RPC is that RPC in NT uses other network programming APIs. Thus, RPC in NT is really a programming API layer that rides on top of named pipes, Winsock, NetBIOS, or Message Queues.

Programmers developing an RPC client/server application define functions that the server implements. The developer writes the client program as if the program can invoke the server functions directly. What happens under the hood, however, is that a client-side library takes the parameters the program

# Inside NT Networking

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

passes to the server function and shapes the parameters into a message. This message transmits to the server, which has a library that unpacks the parameters from the message and passes them to the server function. When the server function finishes, any return parameters go through the same process as the server sends them back to the client. The fact that the parameter packing and unpacking (called marshalling and unmarshalling) and the message transmission are hidden from the application developer makes RPC an attractive API. The Microsoft Interface Definition Language (MIDL) Compiler automatically builds client and server libraries for a developer, based on the function definitions the developer provides.

The Dynamic Data Exchange (DDE) API lets Windows applications communicate with one another via special types of Windows messages. Windows has used DDE since Windows 2.x. Network DDE (NetDDE) lets Windows applications that communicate via DDE reside on different computers. DDE is not a general communications API because graphical Windows are the basis for DDE connections, but DDE is a convenient API whenever it is an option.

SMB communication is the basis for NT's file-sharing client and server, which are known as the Workstation and Server services, respectively. Recently, Microsoft enhanced the SMB API as the Common Internet File System (CIFS) API, in an effort to promote SMB as a network file-sharing standard that can compete directly with Sun Microsystems' Web Network File System (NFS). Use of the SMB API is usually transparent to applications, which access network files by specifying filenames relative to local drive letters that map to a server's share point on a remote file system. Thus, standard file-opening, -reading, and -writing APIs give the application access to the remote files. The Uniform Naming Convention (UNC) method lets applications directly identify the names of files relative to a remote computer's share points.

NT's next built-in network API, DCOM, is the API Microsoft has focused the most attention on in recent years. Microsoft's COM API lets applications consist of different components that are replaceable self-contained modules. A COM object exports an object-oriented interface to methods for manipulating the data within the object. Because COM objects present well-defined interfaces, developers can implement new objects to extend existing interfaces and dynamically update applications with the new support. DCOM lets an application's components reside on different computers, which means applications need not be concerned that one COM object might be on the local computer and another might be across the LAN. DCOM thus provides location transparency, which simplifies developing distributed applications. Similarly to RPC, DCOM is not a self-contained API but relies on RPC to carry out its work. COM+, the COM version that Win2K includes, makes it easier for developers to write COM-based applications by adding language-specific compiler and runtime support for COM objects.

Microsoft introduced its newest network API, Message Queues, in the NT 4.0 Option Pack. The Message Queues API is similar to RPC in that its basis is client/server message exchange. However, whereas RPC is a synchronous interface (i.e., a client must wait until a server finishes processing the client's request before performing other work), Message Queues are asynchronous. For example, a client can send many requests to a server, and the requests fall into a queue. The server processes the client's requests in order, as quickly as possible, and the client can perform other work in the meantime. Message Queues can also perform reliable transactions when used with Microsoft Transaction Server (MTS).

## API Implementation

Network APIs are interfaces that applications use to speak to one another through the network. A network API must rely on a transport protocol, which defines an application-independent (interface-

# Inside NT Networking

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

independent) way for computers to identify one another and for data to transfer between computers. Transport protocols include TCP/IP, NetBEUI, and IPX/SPX.

Before an application can use a network API, the application must link with libraries that give the API to the application. For example, a Winsock-based program would link with many libraries, including `ws_32.dll`, `wsock32.dll`, `mswsock32.dll`, `msafd.dll`, and `wsnapi.dll`. The application calls functions that the libraries define to create sockets, assign a network address to the sockets, and connect to sockets on other computers. The Winsock libraries don't perform all these functions alone, however, but rely on a device driver in kernel mode for the bulk of socket management. The Winsock kernel-mode driver is `afd.sys` (i.e., ancillary function driver—AFD).

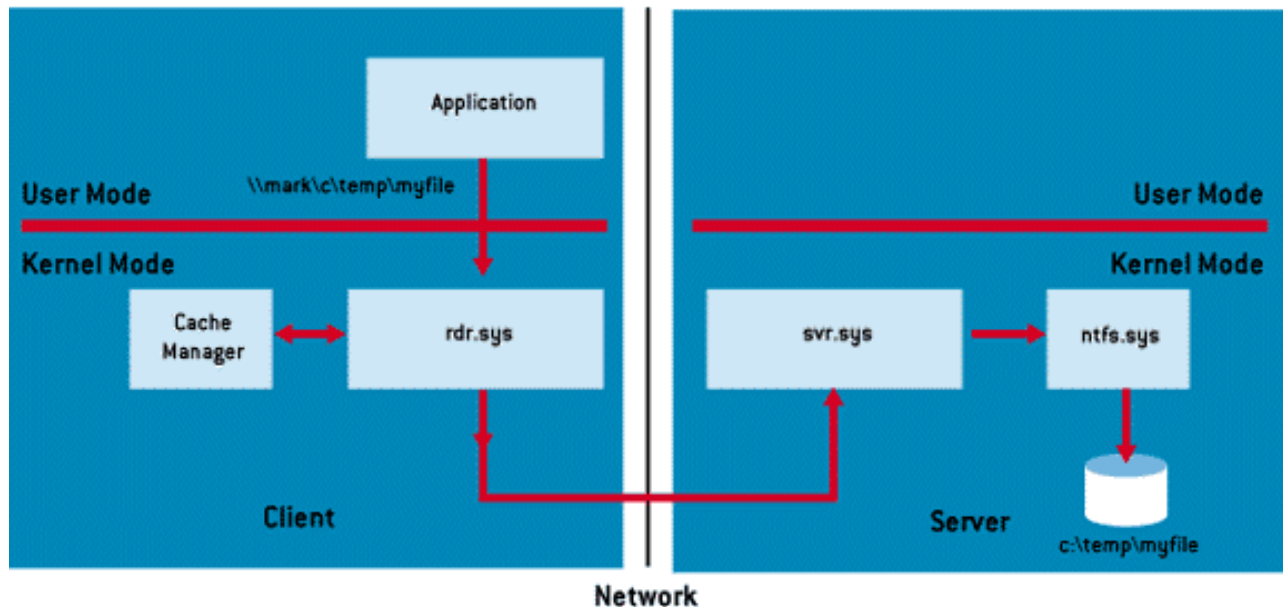
Why does Winsock divide things up this way? Because as a kernel-mode device driver, `afd.sys` can call upon the services of the NT I/O manager to base sockets on NT's file objects. By using file objects as the core of a socket, Winsock can rely on many aspects of NT's I/O infrastructure, including interfaces and security mechanisms, to tightly integrate an API with NT. For example, applications can use NT's native `ReadFile` and `WriteFile` to read and write data through a socket. These APIs are the same ones NT uses for disk-based file I/O, and all the features of these APIs, including asynchronous operation, are automatically at the disposal of a Winsock-based application. To integrate itself with the I/O Manager, `afd.sys` implements as a form of NT file system driver.

The named pipe and mailslot APIs demonstrate other examples of the user-mode/kernel-mode division. Named pipe and mailslot applications link with the `kernel32.dll` library to access the APIs. `kernel32.dll` implements the APIs with the aid of the `npfs.sys` and `msfs.sys` kernel-mode device drivers, which are file system drivers. You can see this connection in the names of the drivers: Named Pipe File System (NPFS) and Mailslot File System (MSFS). Thus, named pipe and mailslot endpoints are based on NT file objects and are accessible through standard file-related APIs. In fact, `npfs.sys` implements a namespace of pipes so a program can obtain a directory listing (using standard file-oriented directory operations) of the pipes a system defines.

The Workstation and Server services are obvious candidates for having kernel-mode file system drivers, and both services have such components. On the client side of a file share, an application accesses a network-located file via NT's file system APIs. The file objects that represent the network files implement with the aid of the `rdr.sys` (redirector—RDR) device driver, which is one step closer to a file system driver than are the other network API drivers I've discussed. `Rdr.sys` uses the NT cache manager to cache data associated with the files that the file-sharing client accesses. When an application accesses data that is not cached on the client, `rdr.sys` sends SMB commands to its counterpart (`srv.sys`) on the file server. `Srv.sys` looks less like a file system driver than a file system client. As do user-mode programs, `srv.sys` accesses files on the server via physical file system drivers such as FAT and NTFS. However, `srv.sys` uses special interfaces that the physical file system drivers provide exclusively for it, including APIs that let `srv.sys` maintain file data consistency across clients accessing the same file. Figure 2 illustrates the flow of a request from a Workstation service client to a Server service server.

# Inside NT Networking

Mark Russinovich  
(Reprinted from WindowsItPro Magazine)



## Resource Name Resolution

One aspect of network APIs is the way in which applications identify network resources such as named pipes or files on other computers. When a named pipe client wants to open a named pipe on a remote server, the client must be able to uniquely identify the pipe. NT has adopted the UNC for network resource naming. UNC specifies names in the form \\servername\resourcesubresource. For example, a named pipe client might open a named pipe with the name \\mark\pipe\mypipe on a server. For specifying network resources that reside on the local system, a period can replace the server name (e.g., \\.\pipe\mypipe).

When an application specifies a UNC name (rather than a standard file name, such as C:\temp\myfile) to one of NT's file system APIs, such as CreateFile, the I/O Manager passes the UNC name to a device driver named mup.sys (Multiple UNC Provider—MUP). A resource redirector, such as rdr.sys, registers with the I/O Manager as being able to locate resources on remote systems. Mup.sys takes UNC names from the I/O Manager and hands the names to registered redirectors for examination. A redirector looks at the names and determines whether it knows how to access the specified resource. If it does know how, the redirector notifies mup.sys and the I/O Manager so that the redirector will receive further I/O requests aimed at the resource. If no redirector claims a name, the original application receives an error code.

I've already mentioned the standard NT LAN Manager redirector, rdr.sys, which handles communications between clients and servers for file sharing. Rdr.sys also handles resource name lookup between computers on a Microsoft LAN Manager network. Thus, a named pipe client specifying \\mark\pipe\mypipe would indirectly use mup.sys and rdr.sys to connect to the pipe on the Mark server.

When you open Network Neighborhood on the desktop or use NT Explorer to connect to a remote printer or file share, you rely on another kind of name resolution. Network providers registered under the HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\NetworkProvider\Order Registry key implement network namespace browsing. Each network provider has a library that is dynamically linked with a standard Microsoft library called mpr.dll (Multiple Provider Router—MPR). Mpr.dll loads each network provider library and queries each library to fill in Network Neighborhood with computers

# Inside NT Networking

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

and the shared resources associated with the computers. Ntlanman.dll is the RDR provider; you'll find it specified under the RDR network provider Registry key:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\LanmanWorkstation\networkprovider
```

The RDR network provider DLL relies on the rdr.sys driver to carry out requests on behalf of mpr.dll. Therefore, the RDR network provider DLL has a private interface with the rdr.sys driver. Another network provider library that comes with NT is the Novell NetWare provider, which permits applications to access resources on NetWare computers.

## Protocol Drivers

Network API drivers must take API requests and translate them into low-level network protocol requests for transmission across the network. The API drivers rely on transport protocol drivers in kernel mode to do this work. Separating APIs from underlying protocols gives the networking architecture the flexibility of letting APIs use different protocols. The protocol drivers that NT includes with its baseline distribution are Data Link Control (DLC), NetBEUI, TCP/IP, and NWLink, although other protocols might be present as options, such as the AppleTalk protocol installed with Services For Macintosh on NT servers.

DLC is a relatively primitive protocol that some IBM mainframes, as well as some HP network printers, use. It is a "raw" protocol, in the sense that no network APIs can use the DLC protocol. Applications that want to use DLC must interface directly to the DLC transport protocol device driver.

IBM introduced NetBEUI in 1985, and Microsoft adopted NetBEUI as the default protocol for LAN Manager and the NetBIOS API. Microsoft has since enhanced NetBEUI, but the protocol is limited because it's not routable and performs poorly on WANs. NetBEUI (NetBIOS Extended User Interface) was so named because it was tightly integrated with the NetBIOS API, but the protocol Microsoft's NetBEUI protocol driver implements is NetBIOS Frame Format (NBF). The Internet's explosive growth and reliance on the TCP/IP protocol has made TCP/IP NT's preeminent protocol. The Defense Advanced Research Projects Agency (DARPA) developed TCP/IP in 1969 specifically as the foundation for the Internet; therefore, TCP/IP has WAN-friendly characteristics such as routability and good WAN performance.

NWLink consists of Novell's IPX and SPX protocols. NT includes NWLink for interoperability between NT and Novell NetWare servers.

So that network API drivers need not employ various interfaces for each transport protocol they might want to use, Microsoft established a standard interface called Transport Driver Interface (TDI). Transport protocols that adhere to the TDI standard export the TDI to their clients, which include network API drivers such as afd.sys and npfs.sys. A transport protocol implemented as an NT device driver is known as a TDI Server. Because TDI Servers are device drivers, they format requests they receive from clients as I/O request packets (IRPs).

Support functions in the tdi.sys library, along with definitions developers include in their drivers, make up TDI. This interface essentially provides a convention for the way network requests format into IRPs. In addition, TDI provides a means whereby a TDI client can register callbacks (i.e., functions that are directly invoked) with TDI Servers. For example, when a TDI Server receives data from across the network, it can invoke a registered-client-receive callback. This event-based callback feature of TDI

# Inside NT Networking

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

eliminates some of the overhead associated with allocating and distributing IRPs, and clients can take advantage of the feature to achieve high performance.

As I hinted earlier, a network API might be able to use only certain protocols. For example, the Workstation and Server services do not know how to use TCP/IP. However, these services understand the NetBIOS interface, and NT provides a driver named netbt.sys that implements a TDI-compliant NetBIOS programming API on top of the TCP/IP driver. Thus, the Workstation and Server service can use either NetBEUI or NetBIOS over TCP/IP. Fortunately, all of NT's programming APIs can use either NetBEUI or TCP/IP directly or indirectly, achieving maximum flexibility on Windows networks.

## NDIS Drivers

When a protocol driver wants to read or write messages formatted in its protocols' format from or to the network, the driver must do so using a NIC, which Microsoft calls a network adapter card (NAC). Because expecting protocol drivers to understand the nuances of every NIC on the market (proprietary NICs number in the thousands) is not feasible, NIC vendors provide device drivers that can take network messages and transmit them via the vendors' proprietary hardware. In 1989, Microsoft and 3Com jointly developed the network driver interface specification (NDIS), which lets protocol drivers communicate with NIC drivers in a device-independent manner. NIC drivers that conform to NDIS are called NDIS drivers or NDIS Miniport drivers.

On NT, the ndis.sys library implements the NDIS boundary that exists between TDI Servers (typically) and NDIS drivers. As is tdi.sys, ndis.sys is a helper library that NDIS driver clients use to format IRPs they send to NDIS drivers. NDIS drivers interface with the library to receive requests and send back responses.

One of Microsoft's goals for its network architecture was to let NIC vendors easily develop NDIS drivers and take a driver code and move it between Windows 9x and NT. Thus, instead of merely providing the NDIS boundary helper routines, ndis.sys provides NDIS drivers an entire execution environment. NDIS drivers aren't genuine NT drivers because they can't function without the encapsulation ndis.sys gives them. This insulation layer wraps NDIS drivers so thoroughly that NDIS drivers do not accept and process IRPs. Rather, ndis.sys receives IRPs from TDI Servers and translates the IRPs into calls into the NDIS driver. NDIS drivers also do not have to worry about reentrancy, in which ndis.sys invokes an NDIS driver with a new request before the driver has finished servicing a previous request. Exemption from reentrancy means NDIS driver writers do not need to worry about complex synchronization, which is made even more tricky because of the parallel execution possible on a multiprocessor.

Although ndis.sys's serialization of NDIS drivers simplifies development, serialization can hamper multiprocessor scalability. Standard NDIS 4 drivers (the NT 4.0 version of ndis.sys) do not scale well for certain operations on multiprocessors. Microsoft gave developers a deserialized operation option in NDIS 5, which will ship with Win2K. NDIS 5 drivers can indicate to ndis.sys that they do not want to be serialized; ndis.sys will then forward requests to the driver as fast as it receives the IRPs that describe the requests. Responsibility for queuing and managing multiple simultaneous requests falls upon the NDIS driver, but deserialization confers the benefit of higher multiprocessor performance. Microsoft debuted deserialization in the version of ndis.sys that ships with NT 4.0 Service Pack 4 (SP4), although the company has not publicly revealed this fact.

The NDIS model also supports hybrid TDI-NDIS drivers, called NDIS Intermediate drivers. These drivers lie between TDI Servers and NDIS drivers. To an NDIS driver, an NDIS Intermediate driver

# Inside NT Networking

Mark Russinovich

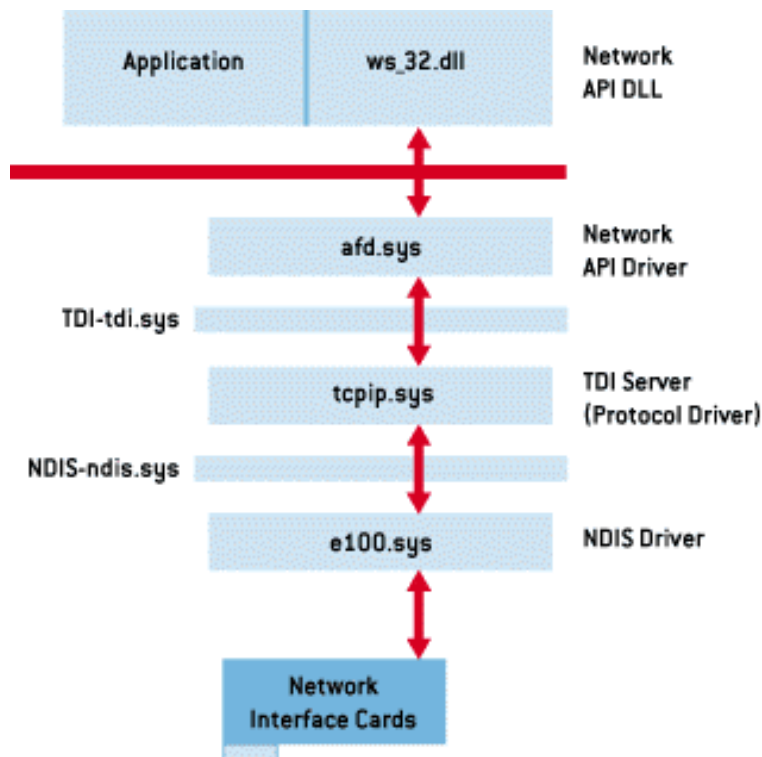
(Reprinted from WindowsItPro Magazine)

looks like a TDI Server; to a TDI Server, an NDIS Intermediate driver looks like an NDIS driver. Microsoft publicly introduced NDIS Intermediate drivers in NT 4.0 as a way for developers to write packet capture drivers. NDIS Intermediate drivers can see all network traffic taking place on a system because the drivers lie between protocol drivers and network drivers. Developers base software that provides fault-tolerant and load-balancing options for NICs on NDIS Intermediate drivers.

NDIS 5.0 introduces a new type of NDIS driver—a connection-oriented NDIS driver. Support for connection-oriented network hardware (e.g., ATM switches) is therefore native in Win2K, which makes connection management an establishment standard in NT's network architecture. Connection-oriented NDIS drivers use many of the same APIs that standard NDIS drivers use; however, connection-oriented NDIS drivers send packets through hardware network connections, rather than placing them on the network medium.

The interfaces that `ndis.sys` provides for NDIS drivers to interface with NIC hardware are available via functions that translate directly to corresponding functions in NT's Hardware Abstraction Layer (HAL). The HAL is a hardware-dependent library that computer vendors can supply to interface NT with proprietary motherboards with a standard API set and exposed hardware model.

Figure 3 illustrates a complete network driver stack. In Figure 3, you can see how Winsock components fit with the TCP/IP protocol and an NDIS driver. (The figure does not show the I/O Manager and HAL).



## Binding

The final piece in the NT networking architecture puzzle is the way in which the components at the various layers—network API layer, TDI Server layer, NDIS driver layer—locate one another. The name of the process that connects the layers is binding. You've witnessed binding taking place if you've changed your network configuration by adding or removing a component with the Network applet.

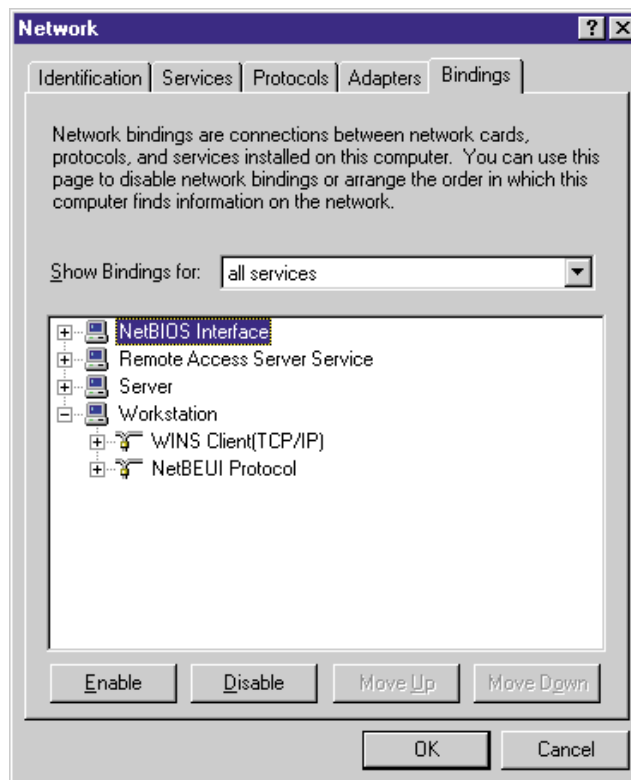
# Inside NT Networking

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

When you install a networking component, you must supply an information (.inf) file for the component. This file includes directions NT must follow to install and configure the component, including binding dependencies or binding relationships. A developer can specify binding dependencies for a proprietary component so that the NT Service Control Manager will not only load the component in the correct order but will load the component only if other components the proprietary component depends on are present on the system. Binding relationships, which the bind engine determines with the aid of additional information in a component's .inf file, establish connections between components at the various layers. The connections specify which components a network component on one layer can use on the layer beneath it.

For example, the Workstation service (RDR) will automatically bind to the NBF (NetBEUI) and NetBIOS over TCP/IP (NetBT) protocols if both protocols are present on the system. The order of the binding, which you can examine on the Bindings tab on the Network properties page (as Screen 1 shows), determines the priority of the binding. When RDR receives a request to access a remote file, RDR submits the request to both protocol drivers simultaneously. When the response comes, RDR waits until it has also received responses from any higher-priority protocol drivers. Only then will RDR return the result to the caller. Thus, reordering bindings so that bindings of high priority are also the most performance-efficient (or applicable to most of the computers in your network) can be advantageous. You can also manually remove bindings with the binding editor.



The Bind value, in the Linkage subkey of a network component's Registry configuration key, stores binding information for that component. For example, if you examine HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\LanmanWorkstation\Linkage\Bind, you'll see the binding information for the Workstation service.

# Inside NT Networking

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

## The Evolving Network

NT's network architecture provides a flexible infrastructure for network APIs, network protocol drivers, and NIC drivers. NT architecture takes advantage of I/O layering to give NT networking support the extensibility to evolve as computer networking evolves. When new protocols appear, developers can write a TDI Server to implement the protocol on NT. Similarly, new APIs can interface to NT's existing protocol drivers. Even less-major enhancements, such as the addition of support for connection-oriented hardware in Win2K, demonstrate that the NT networking model is flexible enough to keep pace with changing networking demands. You can learn more about NT's networking internals in the Microsoft Windows NT Server 4.0 Resource Kit's Networking Guide and the Windows NT Device Driver Kit.