Networking in Microsoft Windows NT

Glen Clark
Microsoft Corporate Technology Team

If there are any comments on this article, or suggestions for future articles, please contact Glen via CompuServe (73750,1607) or FAX: (206) 883-8101.

Created: May/June 1993

A significant difference between the Microsoft® Windows NT™ operating system and OS/2® 1.*x* and even 2.*x* is that networking capabilities were built into Windows NT at the ground level. With MS-DOS®, Windows™ (except for Windows for Workgroups), and OS/2, networking was added on top of the operating system. This meant that the NOS (Network Operating System) designers for Windows NT had the opportunity to design their components within the context of an operating system platform that was still being defined. It also meant that the network team did not have to duplicate the efforts (or code) of the kernel team, and vice versa.

The original designers had three things in mind: First, Windows NT should provide integral, application-transparent networking services. Basic file and print sharing and using services should be part of every Windows NT machine. Second, Windows NT should provide a platform for distributed applications. Application-level interprocess communication (IPC) should be provided for the development of client/server-type applications. Third, the designers recognized that the network market was enormous and growing larger. Windows NT should provide an expandable platform for other network components. All of these goals were to be met within the context of the other major goals of Windows NT, such as portability and security.

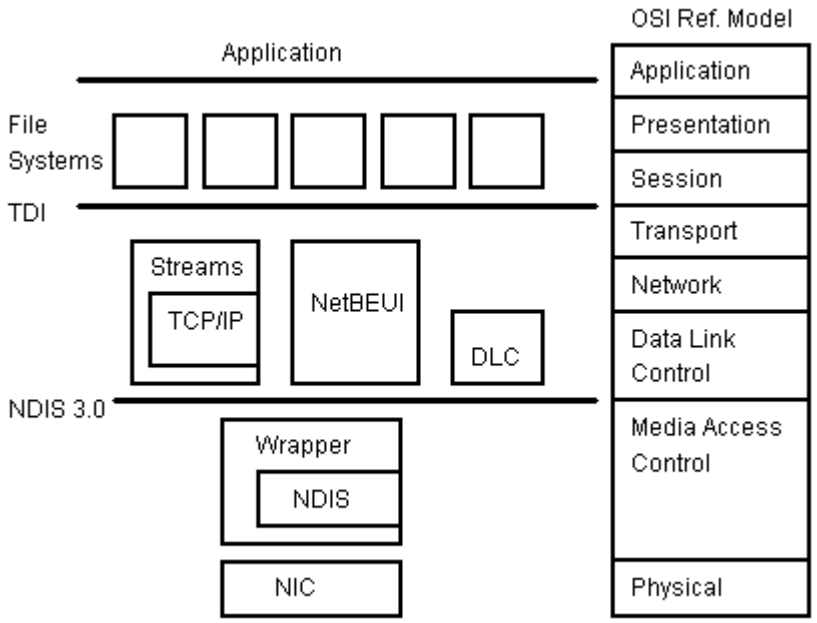

Figure 1.

To understand networking on Windows NT, we need to understand the architecture. As with other architecture components of Windows NT, the networking architecture is built of layers. This helps provide expandability—for others to add functions and services. We are going to look at the model from the bottom up. The layered architecture used by Windows NT mirrors the OSI reference model quite well. The presentation layer is thin to nonexistent, depending on the protocol and system used, however.

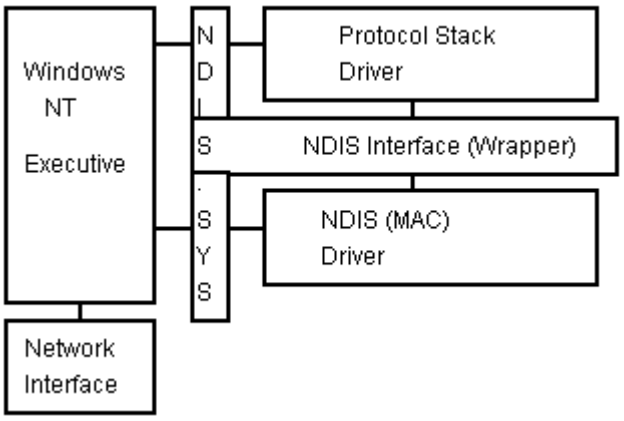

Figure 2.

At the bottom of the networking architecture is the network adapter card device driver. Windows NT currently supports device drivers written to Network Device Interface Specification (NDIS) 3.0. NDIS 3.0 is based on NDIS 2.0, which was the standard used by OS/2 NDIS device drivers. NDIS 3.0 conforms to the device driver standards established for Windows NT. There is a C-call interface; drivers have access to the helper routines; and drivers are 32-bit, portable, and multiprocessor-safe. By providing a standard interface, NDIS permits the high level protocol components to be independent of the network interface card.

Unlike previous NDIS implementations, Windows NT does not need a PROTMAN (Protocol Manager) module to link the various components at each layer. This is accomplished through the information in the Registry and a small piece of code, or wrapper, around all of the NDIS device drivers. The NDIS wrapper provides a uniform interface between protocol stack drivers and NDIS device drivers. It also contains supporting routines, which makes the development of an NDIS driver easier.

Above the NDIS wrapper are the transport protocol device drivers (see Figure 3). Windows NT ships with three transports: NetBEUI provides compatibility with existing LAN Manager, LAN Server, and MS-Net installations; Transmission Control Protocol/Internet Protocol (TCP/IP) provides a popular routable protocol for wide area networks; and Data Link Control (DLC) provides an interface for access to mainframes and printers attached to networks.

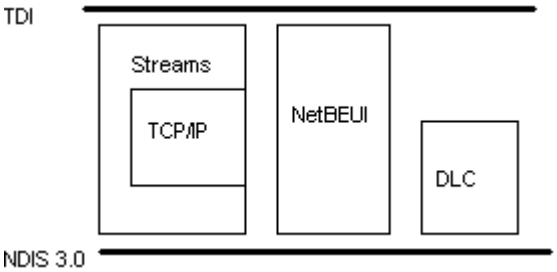

Figure 3.

The DLC protocol is not a full transport protocol by OSI definitions. Its top interface is at the data link control (DLC or, in IEEE terms, the Link Layer Control [LLC]) layer. DLC is used for fast, simple connection and connectionless conversation. The DLC protocol is used to communicate with network attached printers such as the HPIIISi and to communicate with some mainframe computers. It is not possible to establish a client-server or peer-to-peer type session—for file sharing and using, for example—over the DLC protocol alone.

The NetBEUI protocol is provided with Windows NT to maintain connectivity to existing LAN Manager and MS-Net–based networks. The NetBEUI protocol is fast, with low overhead

(number of extra bytes) per frame of data transmitted. The protocol cannot be routed, however. Thus, NetBEUI is most appropriate in single sub-net (continuous network) networks.

When we talk about NetBEUI, it is important to understand that we are talking about the transport layer protocol, not the programming interface NetBIOS (see Figure 4). Earlier implementations on MS-DOS and OS/2 provided the programming interface as part of the transport's device driver. There is nothing wrong with that, but in the Windows NT implementation we have separated the programming interface (NetBIOS) from the transport protocol (NetBEUI) to increase flexibility in the layered architecture. Separating the two allows us to use the same NetBIOS driver code for multiple transports such as NetBEUI and TCP/IP.

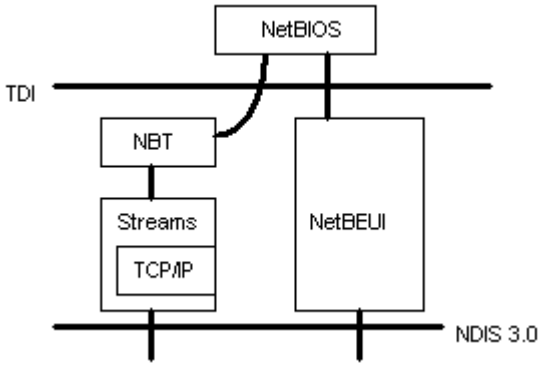

Figure 4.

Finally, we want to look at TCP/IP (Figure 5). TCP/IP is implemented slightly differently from what we have seen with NetBEUI or DLC. Instead of being a single device driver bound directly to the NDIS device driver, TCP/IP resides "inside a wrapper." This wrapper is called Streams (or the Streams driver). Calls to the TCP/IP transport protocol driver must first go through the upper layer of the Streams device driver, and then to the NDIS device driver via the lower end of the Streams device driver.

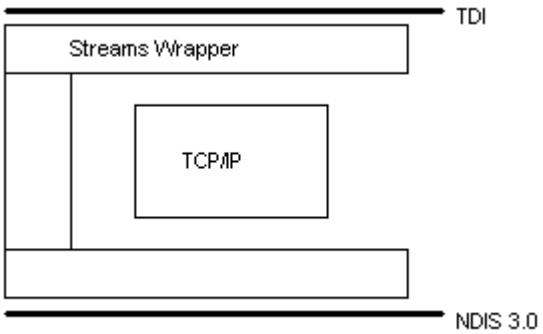

Figure 5.

Streams is a significant departure from the way protocol stacks were developed for MS-DOS and OS/2. There are several reasons for the use of the Streams mechanism. Streams makes it easier to port existing protocol stacks to Windows NT. Streams also encourages protocol stacks to be organized in a modular, stackable style, thus moving closer to the original vision of the Open Systems Interconnection (OSI) model.

The Transport Driver Interface (TDI) provides a common interface for file system and I/O manager processes to communicate with the various network transports. It is a very "thin" layer. There is little code actually involved with the TDI. The TDI interface is based on 32-bit-wide handles. This increases the connection capacity between upper layers and protocols such as NetBEUI, which traditionally only allows an 8-bit-wide handle (LSN—Local Session Number).

The first design goal of the networking system is to support file and print sharing and using. This is accomplished by two modules, LANMANWorkstation and LANMANServer. These two components, with the help of several more we will identify, provide most of the functionality of the OS/2 version of LAN Manager available today. Both of these modules execute as 32-bit services.

The LANMANWorkstation module is really in two pieces (see Figure 6). The LANMANWorkstation component provides the user-mode interface. The other component is the RDR, or Redirector. This component is a File System Driver (FSD) that actually does the interaction with the lower layers of the protocol stack.

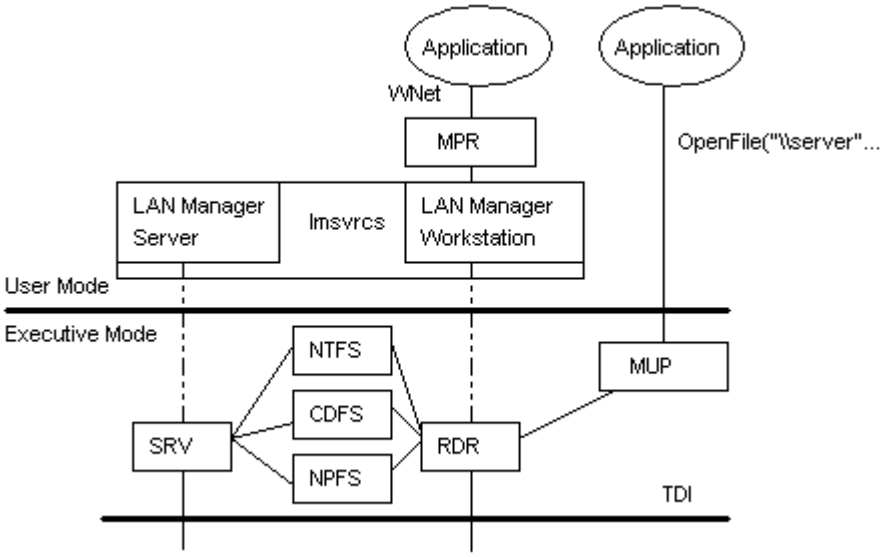

Figure 6.

Multiple UNC (Universal Naming Convention) Provider (MUP) is an interesting entity that runs in kernel-mode memory. The most productive way of thinking of MUP is as a resource locator. The types of resources it locates are UNC names. A UNC name is a naming convention for describing servers, and sharepoints on those servers, on a network. UNC names start with two backslashes (\\) followed by the server name. All other fields in the name are separated by a single backslash (\). A typical UNC name would appear as:

\\server\share\subdirectory\filename

Not all of the components of the UNC name need to be present with each command. "\\server" is sufficient to find a server to get a list of its sharepoints.

Unlike the NDIS and TDI boundary layers, MUP is actually a program. NDIS and TDI simply define ways for a component on one layer to communicate with another over specifically defined paths called binds. MUP, too, has defined paths to redirectors or, as the name implies, UNC providers. The problem is that for any UNC name, MUP is not sure to which of potentially many different UNC providers the command should go.

MUP receives commands from applications that contain UNC names. If this is a UNC name that MUP has not seen in the last 15 minutes, it will send the UNC name to each of the UNC providers that are registered with it. This is why MUP is a prerequisite of LANMANWorkstation. One of the first tasks the LANMANWorkstation did when initializing was register with MUP. The command and the security context of the application generating the request will be passed to the redirector with the highest registered priority response that claims it can establish a connection to the UNC.

One might ask, why a MUP? LANMANWorkstation is the only UNC provider. This is true today. Recall, however, that one of the major design goals for networking in the Windows NT environment was to build a platform upon which others can build. MUP is a vital part of allowing multiple redirectors to coexist in the machine at the same time.

LANMANServer is much like the LANMANWorkstation module. It is a service that runs in the lmsvrcs process. Unlike the workstation component, it is not dependent on the MUP service, since the server is not a UNC provider. It doesn't attempt to connect to other machines, but it is connected to by other machines. Like LANMANWorkstation, it is composed of two parts: the LANMANServer component and the SRV component. The SRV component handles the interaction with the lower levels and also directly interacts with the other file system devices to satisfy command requests such as file read and write.

In addition to the workstation and server services from LAN Manager, a number of other services were ported over: the Alerter, the Messenger, the Browser, and the Replicator. The Alerter is used to forward alerts generated on the local machine to remote computers or user names. The Messenger receives messages and alerts, and displays them on the screen in the form of a message box. The Browser is used to collect information about the machines in this domain or workgroup, and to inform users of these facilities when asked. The information collected by this facility is most obvious in the File Manager when attempting to connect to a new drive. Finally, the Replicator service permits the automatic copying of a directory from one machine to another. The source of the data is said to be on an export machine while the target is an import machine. A Windows NT Advanced Server can be either an export machine or an import machine, or both simultaneously. A Windows NT Server can only be an import machine.

So far we have built up to the redirector and server levels, and accomplished almost all of the design goals. One piece remains, however. Above the redirector and server components live the applications. As with our other layers, we want to provide them with a single unified interface to develop to, independent of the lower-layer services. This is done through two mechanisms. We have already looked at the first—MUP. The other is the MPR, or Multi-Provider Router. Applications that make I/O (Input/Output) calls that contain UNC names are directed to the MUP, where the appropriate UNC provider or redirector is located.

The MPR is much like the MUP. This layer takes in WNet commands, finds the appropriate redirector based on the handle, and passes the command to that redirector for communication onto the network. In addition to I/O calls such as Open and Close, Win32™ contains a set of APIs called the WNet API. These are APIs that were ported over from Windows 3.1 network calls. Most of these calls deal with establishing remote connections. With these commands and the standard I/O commands, an application can do almost all of the networking functions needed.

The goal of distributed computing is to divide the computing task into two sections: One section runs on the client's workstation, something that may not take a great deal of computing power but would require a lot of network bandwidth. This section includes operations such as screen graphics, mouse movements, and keyboard functions. The other section of the process requires large amounts of data, number crunching, or specialized hardware. This section includes operations such as database lookups and updates, or mainframe data access. Central to the theme is that there is a connection between the client and the server at a process-to-process level that allows data to flow in both directions. There are a number of different ways to establish this conduit; this article will discuss the six different mechanisms that Windows NT provides.

The six IPC mechanisms provided by Windows NT are: named pipes, mailslots, NetBIOS, Windows Sockets, Remote Procedure Calls (RPC), and Network Dynamic Data Exchange (NetDDE). Named pipes and mailslots provide backward compatibility with existing LAN Manager installations and applications. This is also true of the NetBIOS interface. Windows Sockets is a Windows-based implementation of the widely used sockets programming interface

created by the University of California at Berkeley. RPC is compatible with the Open Software Foundation/Distributed Computing Environment (OSF/DCE) specification for remote procedure calls. NetDDE allows standard DDE connections to be redirected across the network as was possible with Windows for Workgroups.

Named pipes and mailslots are implemented slightly differently than the other IPC mechanisms. They are actually written as file systems (FS). Thus, in the Registry you find an MSFS and an NPFS. As file systems they share common functionality with the other file systems, such as security. In addition, local processes can use named pipes and mailslots with other processes on the local machine without going through the networking components. Remote access to named pipes and mailslots, as with all of the file systems, is accomplished via the redirector.

Named pipes are based on the OS/2 API set. Most of the calls, however, have been ported to the Win32-based API set. Additional asynchronous support has been added to the named pipes to make support of client/server applications easier. Because named pipes is a standard file system, it can take advantage of the cache manager. This can be used to improve the performance of certain types of named pipe applications. Specifically, the cache can be used with character mode pipes to buffer "outbound" traffic for a number of characters or a number of seconds. This can improve performance by reducing the number of frames (and network overhead) generated.

A new feature added to named pipes is "impersonation." In impersonation the server can change its security identity to that of the client on the other end. This is done with the **ImpersonateNamedPipeClient()** API. Assume you have a database server system that uses named pipes to receive read and write requests from clients. When a request comes in, the database server program can impersonate the client before attempting to perform the request. So even if the server program does have authority to perform the function, the client may not, and the request would be denied.

The mailslot implementation in Windows NT is not the full OS/2 LAN Manager implementation. Where in LAN Manager there are first- and second-class mailslots, in Windows NT only second-class mailslots exist. Mailslots provide connectionless messaging, basically broadcast messaging. Delivery of the message is not guaranteed, although the delivery rate on most networks is quite high. It is most useful for discovering other machines or services on a network, or for wide-scale notification of a service. The use of mailslots should be contained as much as possible, however. Each mailslot transmitted is received by each machine on the local area network and processed at least to the degree that determines if the message is to be received or not. This can cause workstations to slow down. In addition, applications designed using mailslots are probably limited to local-area network implementations only, since most wide-area networks do not propagate broadcast messages across bridges or routers.

The use of NetBIOS as an IPC mechanism has been around since the introduction of the interface in the early 1980s. From a programming aspect, however, higher-level interfaces such as named pipes and RPC are superior in their flexibility and portability. The NetBIOS entry in the Registry defines a common interface point for multiple possible transport protocol providers.

The sockets interface for TCP/IP was created by the University of California at Berkeley in the early 1980s. Since then it has become a very popular interface for developing distributed applications in the TCP/IP and UNIX environments. Microsoft, in cooperation with several other software vendors, developed the Windows Socket API set to accomplish two things: (1) to migrate the sockets interface into the Windows and Windows NT environments, and (2) to help standardize the API set for all platforms. The Windows Socket interface for Windows NT is a layer above the TCP/IP.

The RPC mechanism is unique in that it uses the other IPC mechanisms to establish communications between the client and the server. RPC can use named pipes, NetBIOS, or TCP/IP Sockets to communicate with remote systems. If the client and server are on the same machine, it can use the LPC (Local Procedure Call) system to transfer information

between processes and subsystems. This makes RPC the most flexible and portable of the available IPC choices.

Much of the original RPC work was started by the SUN® computer company and carried forward by the Open Software Foundation (OSF) as part of their Distributed Computing Environment (DCE). The Microsoft RPC implementation is compatible with the OSF/DCE standard RPC. It is important to note that it is *compatible* but not *compliant*. "Compliance" in this situation implies that one started with the OSF source code and worked forward. For a number of reasons, Microsoft-developed RPC started with the OSF specification but not the source code. The RPC mechanism is completely interoperable with other DCE-based RPC systems such as the ones for HP® and IBM®/AIX® systems.

Looking at how RPC works, we first need to understand what RPC is attempting to accomplish. A program can be viewed as having a "backbone" and a series of "ribs" spanning off the backbone. The backbone is the mainstream logic of the program, which should rarely change. The ribs are the procedures the backbone calls on to do work or functions. This is simply another way of looking at "structured" programming. In traditional programs, the ribs were "statically" linked to the backbone—that is, they were linked and stored in the same executable module.

With OS/2 and Windows, the concept of dynamic-link libraries (DLLs) is used. With DLLs the procedure code and the backbone code are in different modules. This allows the DLL to be modified or updated without changes to or redistribution of the backbone modules.

RPC takes the concept one step further and places the backbone and the ribs on different machines. Doing this raises many issues, such as data formatting, integer byte ordering, locating the server that contains the function, and establishing the communications mechanism being used (see Figure 7).

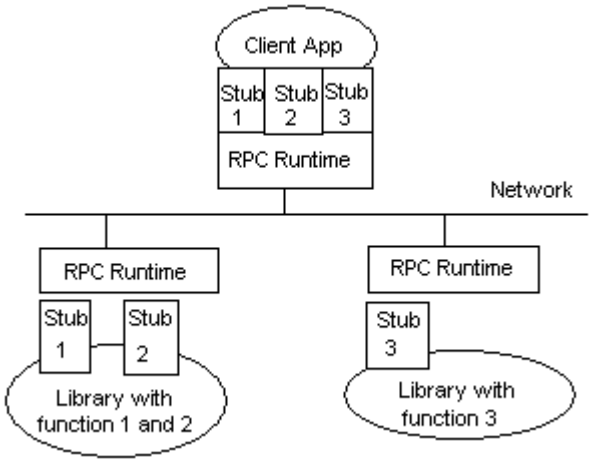

Figure 7.

When these areas are put together, we have RPC. The client application was developed with a specially compiled "stub" library. The client application thinks it is calling its own subroutines; in reality these stubs are transferring the data and the function down to a module called the RPC Runtime. This module is responsible for finding the server that can satisfy the RPC command. Once found, the function and data are sent to the server, where they are picked up by the RPC Runtime module on the server. The server piece then loads the needed library for the function, builds the appropriate data structure, and calls the function. The function thinks it is being called by the client application. When the function is completed, any return values are collected, formatted, and sent back to the client via the RPC Runtime modules. When the function returns to the client application, it either has the appropriate returned data or it has an indication that the function failed in stream.

Finally at the top of the mountain! From the application viewpoint there are two sets of commands that can cause network traffic: any I/O command, such as Open, which contains a UNC name, and WNet commands. UNC commands are sent to the MUP in the Windows NT kernel, where it finds a UNC provider or redirector that can make a connection to the specified UNC name. WNet commands are passed to the MPR, which passes the request to each redirector in turn until one is found that can satisfy the request. One machine gains access to another machine via a redirector. Windows NT ships with a redirector that allows connection to LAN Manager, LAN Server, and MS-Net servers. This redirector communicates to the protocol stacks to which it is bound via the TDI layer. (The TDI layer is a boundary layer between the file system modules and the network protocol stacks. Boundary layers are used to provide a unified platform for others to develop "plug-and-play" components.) Windows NT ships with three protocol stacks: TCP/IP, NetBEUI, and DLC. TCP/IP is wrapped in the Streams driver. Streams will make porting other protocol stacks to Windows NT easier. Protocol stacks communicate with the network interface card (NIC) via an NDIS device driver. NDIS 3.0 provides another boundary layer that makes interoperability between components at different layers easier. In addition to providing file and print sharing capabilities, Windows NT provides five mechanisms for building distributed applications. Named pipes, mailslots, NetBIOS, Sockets, and RPC can all be used. The most portable is the RPC mechanism. RPC uses other IPC mechanisms to transfer functions and data between client and server machines.

Through this process we have seen how the three main goals of the design for networking within Windows NT have been accomplished. First, file and print services are provided through the default server and workstation components. These default server and workstation modules are the progression of the LAN Manager technology into the Windows NT environment and provide backward compatibility for your existing LAN Manager-based networks. Second, we have seen support for client-server applications in the rich availability of IPC mechanisms such as named pipes, Windows sockets, and of course RPCs. And finally, we have seen the design of an expandable architecture based on standard boundary layers upon which others can (and do) build.
