CHAPTER 10

Protocols

The Network+ Certification exam expects you to know how to

- 2.4 Differentiate between the following network protocols in terms of routing, addressing schemes, interoperability, and naming conventions: TCP/IP, IPX/SPX, NetBEUI, AppleTalk
- 2.13 Identify the purpose of network services and protocols, such as SMB (Server Message Block)

To achieve these goals, you must be able to

- Understand the concept of protocols
- Learn about the NetBEUI protocol suite
- Learn about the IPX/SPX protocol suite
- Learn about the TCP/IP protocol suite

Everything you've learned up to now points to a single goal—getting the right packets to the right system. We've concentrated on hardware in the form of cabling, hubs, switches, and NICs that handle the majority of this job. Now it's time to move away from hardware and start looking at the software built into your operating system that handles the primary networking duties: the network protocols.

Using the OSI seven-layer model as a guide, this chapter defines network protocols and details exactly what network protocols do to make the network work on your system. Once you understand exactly what network protocols do, we'll take a deeper look at a number of different network protocols from both the past and today to see how they work and how to install them.

Historical/Conceptual

Network Protocols

The term *network protocol* describes all the software on a PC that enables your applications to share or access resources. Your first reaction might be "Cool! Where is this software?" Don't bother trying to look under My Computer to try to find these programs. Of course, they do exist, but every operating system stores them in different ways. These

programs might be executable files or they might be support files, such as Windows Dynamic Link Libraries (DLLs). Some operating systems build these programs into the core of the operating system itself. Even though it's difficult to show the exact programs that do this work, you can understand what they do and see what you can do to make them work.

Only a handful of network protocols exist. Currently, the famous TCP/IP network protocol is by far the most commonly used in networks. Other network protocols, such as Novell's IPX/SPX and Microsoft's NetBIOS/NetBEUI, are also popular and well supported, although both of these have lost ground to TCP/IP. There's a simple reason for this: the biggest network in the world, the Internet, runs on TCP/IP.

NOTE I'd like to apologize on behalf of the entire networking industry for its horrific use of the term "protocol." You know from the OSI seven-layer model that a network uses many different protocols and procedures to get data from one system to another in the proper format. The networking industry's flexible use of the term can cause a great deal of confusion. When a network tech, book, or FAQ uses the word "protocol," take the time to be sure you understand what type of protocol is involved.

Protocol Stacks

Notice that the network protocols I just named all have a slash in their names. Those slashes are there for a reason. Network protocols are groups of protocols designed to work together, but at different levels of the OSI seven-layer model. The TCP/IP label, for example, stands for the Transmission Control Protocol (TCP) and the Internet Protocol (IP). The TCP part of TCP/IP defines a set of protocols that run at the Session and Transport layers. The IP of TCP/IP defines the protocols that run at the Network layer. We use the terms *protocol stacks*—or less commonly, *protocol suites*—to describe these groups of network protocols.

TIP A protocol stack is a group of protocols that work together from Layers 3 through 7 of the OSI seven-layer model to enable your applications to use the network.

In fact, the details get even a bit more complex. When you say TCP, you'd think that was a single protocol, but it's a number of protocols operating at the same OSI layers with names like TCP, UDP, and ICMP. So, even though a single network protocol might only have two names separated by a slash, that doesn't mean there are only two! Fear not! This chapter breaks these protocols down to show you their different functions. For now, get the idea into your head that a network protocol isn't a single protocol. A network protocol is many protocols, working together at the Application, Presentation, Network, Transport, and Session layers of the OSI.

Protocols by Layer

All network protocols perform basically the same functions at the Session, Transport, and (on some protocols) Network layers of the OSI seven-layer model. Network protocols also function at the Presentation layer and Application layer, although how they work at these layers differs tremendously, depending on the protocol.

Each of the major protocol suites provides a different mix of efficiency, flexibility, and scalability (*scalability* means the capability to support network growth). NetBIOS/NetBEUI works best for small networks without routers; IPX/SPX provides support for integrating with Novell NetWare; and TCP/IP provides a complex, robust, open solution for larger networks. The rest of this chapter inspects these and a few other network protocols to help you understand which one to use, how to install them in Windows client PCs, how to configure them, and how to deal with network protocol suites when they fail. This is a big job, which we will break down by protocol. First, we need to cover a few critical points about network protocol suites in general. Let's do this by clarifying exactly what a network protocol does at the Session, Transport, and Network layers. We'll cover the Application and Presentation layers function in the section called "IPX/SPX," when we discuss individual network protocols.

A Network Protocol's Session-Layer Functions

From a conceptual standpoint, a network protocol's Session-layer functions are far more complex than the Transport and Network functions combined. Let's conquer the Session functions first—Transport and Network will be much shorter!

A system's *Session-layer software* has many jobs. It must create a connection—a session—between two systems. Second, it must determine the type of data being moved in that session and confirm the other system supports that particular type of data. Third, the Session software monitors the session as data moves from one system to another. Fourth, the Session software must have a method of handling multiple sessions at a single time. Last, the Session software must recognize when the data transfer completes, and then shut off that session. (See Figure 10-1.)

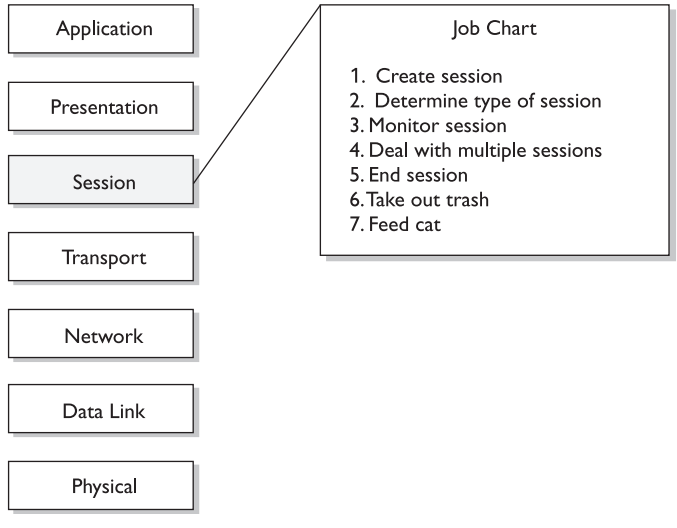


Figure 10-1 Five jobs of the Session layer

Interestingly, an almost perfect analogy to the Session-layer functions is a family reunion. Don't laugh, this works! Imagine you're entering a pub, crowded full of family members you haven't seen in years, as well as their sundry significant others. To make this analogy work best, assume everyone is wearing those "Hi! My Name is ____" name tags. (It's been a few years or decades, after all, since you've been together!)

You enter the pub and begin to scan the room. Initially, your goal is to purchase a beverage. To secure a beverage, you need to create a conversation, a "session," with the bartender. You locate the bartender (she's wearing a name tag that says "bartender"). She's busy with another customer, so you wait, occasionally doing something (raising your hand or yelling "bartender!") to let her know you want to create a session. Eventually, the bartender finishes with a customer and notices you. She asks you what you need. You tell the bartender to pour a pint of Guinness for you. She says "Okay." Congratulations! You now have established a session!

Network session protocols work nearly the same way. Every network protocol uses some form of addressing function (the name tag) that goes beyond MAC addresses. This might be a number or it might be a word—but all network protocols will provide some form of addressing convention above and beyond MAC addresses.

What if you couldn't locate a bartender? In that case, perhaps you might yell out, "Where do I get a drink?" Assuming you didn't offend anyone, the bartender would yell back, "Right here! I'm the bartender!"

All network protocols realize that not every other computer will know the name of every other computer on the network. The network protocol must provide some method—we like to use the term *name resolution*—to enable one computer on the network to locate another to establish a session. All network protocols perform name resolution in one of two ways: *broadcasting*—the equivalent of yelling to the whole network, or by providing some form of *name server*—a computer whose job is to know the name of every other computer on the network. A name server would be analogous to an information booth in the corner of the pub. You could ask a person manning the booth to locate any other person in the room and he could point her out for you.

NOTE Even though the Session-layer software of the network protocol needs the name of the other computer to create a session, that doesn't mean the Session-layer software performs the name resolution! In most network protocols, name resolution is handled by software at higher or lower levels of the OSI seven-layer model.

Most network protocols combine broadcasting and name serving in one way or another. The bottom line is that network protocols must give you a way to get the name of any other computer on the network.

Let's head back to the pub, shall we? In both the bartender's mind and yours, each of you realizes that a session is taking place. You each also assume certain actions to perform based on the type of session. The bartender gives a pint of Guinness to you. You ask the cost. The bartender tells you. You give money. You get change. You and the bartender know what to do for this type of session. If you had asked for directions to the restroom,

you would still have a session, but both you and the bartender would assume a totally different set of actions.

All network protocols must not only establish a session, they must identify to the other system the type of session they want to initiate. All network protocols use some type of function-numbering system that identifies the type of session you want to perform. These numbers are embedded into the packet along with the name of the computer with which you want to create a session. Network protocols initiate a session by stating the name of the remote system, the name of the system making the request, and the function desired. It's as though you said to the bartender, "Hi, bartender, my name is Mike. Would you pour me a Guinness?"

It's important to appreciate that not all systems on a network perform every possible function. Imagine walking up to a huge, burly man sitting at a booth and saying, "Hello, Uncle Bruno. Could I have a Guinness?" In the human world (assuming he is the pleasant sort), he might say, "I'm sorry, I'm not a bartender." In the computer world, if you send a packet with a function number built into the packet for a type of data transfer the other computer doesn't know how to respond to, you usually just get silence (there are exceptions to this). A successful session requires both the name of the other computer and a function number that the other computer knows how to handle.

The bartender is capable of serving you a pint of Guinness. She looks at you and says, "Hi, Mike. I'll get you a Guinness!" and the session begins. In the networking world, the responding system must give you some form of function number in return. This function number may be the same in some network protocols, while in others, it may be a totally different number. Either way works perfectly well, as long as each system keeps track of the corresponding function numbers.

While waiting for your pint, you scan the room, only to discover a lady is looking right at you with a friendly smile on her face. A new session is taking place! You see her nametag says "Laura." Laura is initiating a session! You look back at her, smile, and casually stroll over for a chat. Seems that Laura is your long-lost cousin and you begin to converse. You two humans certainly don't need a function number here, but a network protocol would, of course. The session is successfully established—you and cousin Laura begin to catch up on family stories.

The interesting part (from a network protocol standpoint) is that you now have two sessions running—a session with cousin Laura and a session with the bartender (remember, you still haven't gotten your Guinness). In your mind, you must keep track of both sessions at the same time. This is also true of network protocols: the session software must keep track of multiple "open" sessions. Not only do you need to remember the session with the bartender, you must remember the current status of the active session—keeping up conversation with cousin Laura. At this point, you are simply waiting for your beer—there's nothing you need to do except listen for the bartender to tell you your beer is ready. The same is true in a network. Not all sessions need to be constantly active, just constantly monitored.

While speaking to Laura, you notice another person you want to speak to. Even though Laura and you are conversing, you turn to the other person and start to talk. In the human world, this would irritate cousin Laura no end (and probably result in a

rather abrupt end to your session!), but in the networking world, there's no problem carrying on the same type of session with multiple computers at the same time. A good example is if you had two web browsers open at the same time, each going to a different web site (Figure 10-2). Even though the function numbers are the same for both sessions, the network session software lists them with different computer names and keeps each session separate from the other. So you can initiate a session with another person without fear of cousin Laura just walking away.

While speaking to this other person, you discover that he is poor cousin Martin and he starts asking for a loan. Given your "never loan money to family" policy, you politely break off the conversation with cousin Martin, each of you saying a pleasant goodbye. In the networking world, session software must recognize a session has ended and close that session. Sessions don't always end that cleanly, though, and so session software must also recognize when sessions end abruptly. Imagine if cousin Martin, miffed as you began to say you would not loan him money, simply walked away. You might stand there for a moment, perhaps call out to him, but eventually you would realize that the conversation was over and return to talking with cousin Laura. This is also true in the networking world. The session software waits for response, calls out to the other system to confirm the session has died—"Hey, wait!"—and then closes the session and removes it from memory.

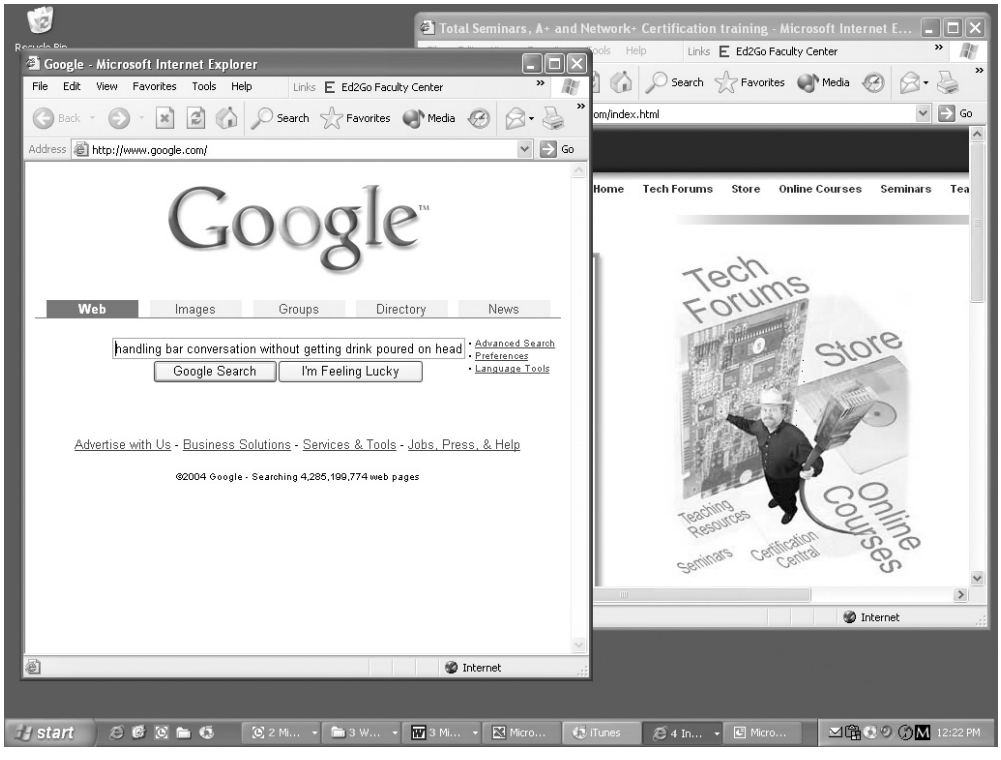


Figure 10-2 Two identical session types open

Returning your attention to cousin Laura, you notice her drink is almost empty. Without ending the current topic of conversation (Can you believe what happened to Uncle Bruno?), you ask her if you might buy her a drink. You now have two separate conversations running: one about Uncle Bruno and one about a drink. This is also a critical aspect of session software. The session software must enable two computers to run two totally separate sessions. In the networking world, this takes place constantly—for example, you might access two different shared folders on a single system. The session software tracks each session separately using the function numbers, even though the computer names are the same.

All the sessions discussed so far are *connection-oriented*: this means that both computers had to recognize by name and accept the function of the other computer before data could transfer. But now imagine that there's a pay telephone in the pub, and it rings. Niece Susan, quietly reading a book next to the telephone, picks up. It's that no-good Uncle Bruno's boss on the line. She tells poor Susan that Bruno had better get to the office quickly! Susan hangs up, and then yells to Bruno—without bothering to see if Bruno is listening, "Uncle Bruno, you better get to work! Your boss is angry!" Susan then returns to her book. Network protocols occasionally do the same thing. They send packets out without first creating a connection-oriented session. We call these *connectionless* sessions. Network protocols use connectionless sessions only for data that won't cause problems if it doesn't make it to the intended recipient; a great example of a connectionless session is the popular PING command. Like Susan in our example, who doesn't care much for Bruno and therefore doesn't care whether or not he hears her, a connectionless session might or might not get its point across.

TIP Make sure you understand the difference between connection-oriented and connectionless sessions.

Let's review the functions of the Session-layer software of every network protocol—I'll leave the end of the analogy to your imagination (have fun at the reunion!). The session software of a network protocol:

- Establishes sessions with another system
- Needs a name or number other than the MAC address to identify a destination system
- Needs a function number to determine what type of data transfer should take place
- Can support multiple sessions, both with multiple machines and multiple sessions on a single machine
- Must monitor open sessions and close completed sessions

The Session layer is easily the most complex, most visible part of the network protocol stack. Given this complexity, almost all operating systems give you tools to inspect

sessions in one way or another. Even though Session-layer tools exist, most of these tools only enable you to inspect, not to make changes to your sessions. So where can you tweak the Session layer functions? Don't bother looking for a "Session Layer Configuration" program on your computer— no such program is on any operating system. Instead, think about what a session needs: a computer name (or number) and a function number—and those you can adjust! Every operating system has ways for you to name or number your computer and ways to define what functions your computer can ask for or accept. Where and how these settings take place, though, varies wildly by operating system and by the types of resources you want to share or access.

A Network Protocol's Transport-Layer Functions

Once the Session layer has done its magic and made the right kind of connection with a destination machine, the Transport-layer software steps in to handle the data. In Chapter 3, "Building a Network with OSI," you learned that the main job of the *Transport layer* is to chop up data into packet-sized chunks and add a sequence number before passing the packet down to the Network layer for further processing. On the receiving side, the Transport layer reassembles the packet passed up from the Network layer, inspecting the sequence numbers to verify proper data reassembly. See Figure 10-3.

Unlike the Session layer, no operating system has a way to open a hatch to show you the Transport layer at work. There is no utility program to run to show the assembly or disassembly of data into packets or packets into data. Perhaps this is a good thing: at least one part of every protocol stack works automatically without the need for us to make adjustment or changes.

Windows systems' Transport layer software, by default, will make packets using Ethernet's maximum frame data limitation of 1,500 bytes. This is called the Maximum Transmission Unit (MTU) and the value is stored in the Windows' Registry. Many systems, especially systems using ADSL or cable modems for Internet access, (See Chapter 16, "Remote Connectivity," for details on ADSL and cable modems) enjoy improved perfor-

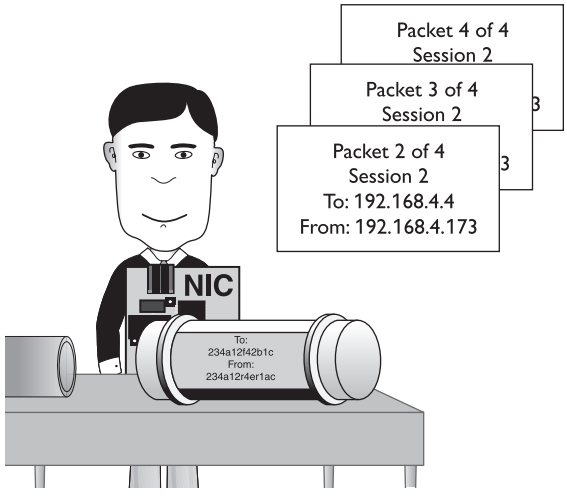


Figure 10-3 The Transport layer at work

mance by adjusting the MTU size to a lower value. Check out the FAQ section of the wonderful web site—www.dslreports.com—to see if adjusting your MTU makes sense for your system. Included with this book's CD is a handy program called Dr. TCP that adjusts your MTU settings without forcing you to dig through your Registry. Try it!

A Network Protocol's Network-Layer Functions

Once the Transport layer has neatly divided and numbered the data into packets, the Network-layer software does its job. In Chapter 3, you saw how the *Network layer* of the OSI seven-layer model deals with the network protocol's capability to provide some form of universal addressing system enabling computers to communicate on any type of hardware technology. Actually, the network layer does more than just enable a system to communicate across platforms. By creating a universal numbering system, large networks can be broken down into smaller subnetworks (the hip term is *subnets*).

As networks get larger and larger, the simple number of machines trying to communicate on the same wire can create both performance problems for the network and difficulties for administration. The first resource for revving up a bogged-down network, as you'll recall from Chapter 4, is to replace the hubs with switches, but even that won't help enough if the network growth spirals out of control. By breaking a single big network into two or more *subnets*—groups of computers that are a subset of a larger network, defined most often numerically—administrators can reduce the overhead for all the machines. Thus, subnets provide a huge benefit for large networks, even large networks that share the same type of network technology.

Subnets are conceptually fairly easy to understand, although in real-world application, things get rather tricky. The secret lies in the universal numbing system. To break a network into subnets (through the process, conveniently called *subnetting*), the numbering system must work in such a way that specific groups of computers can be separated from the rest of the computers on the network. In the TCP/IP world, subnets of computers are identified by the part of their IP addresses that match each other. For example, a single subnet might be all the computers that share the first three octets of 192.168.4. So, computers 192.168.4.3 and 192.168.4.34 are on the same subnet. A computer with the IP address of 192.168.22.3 would not be part of the same subnet. See Figure 10-4.

You know from Chapter 3 that routers read incoming packets to determine which machine on the local network should receive the packet. Routers use the packet information to decide which port to route the packet. Chapter 3 used a single router connected to a cable modem, and then the Internet that enabled multiple PCs on the MHTechEd LAN to connect (remember Dana and Janelle?). A single router might have many connections. In fact, some routers have so many ports on them, they physically look like a switch. This makes for an interesting analogy. Switches decide where frames go based on their MAC addresses. Routers decide where frames go based on their subnet number. See Figure 10-5. This analogy is so strong that it's common to hear a router referred to as a *Layer 3 switch*.

The important point from the concept of a network protocol is to use routers, a network protocol must provide its own universal number convention. Not all network protocols do this! Only two protocols, IPX/SPX and TCP/IP, provide some form of universal

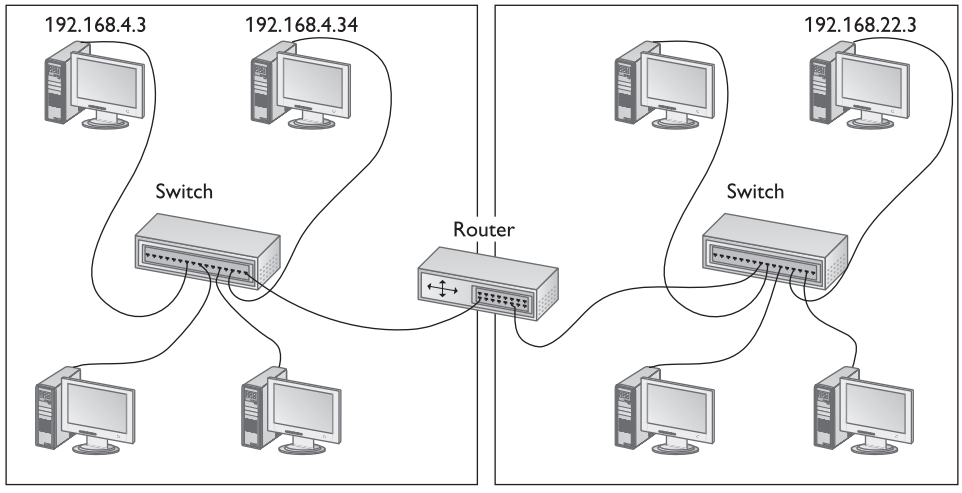


Figure 10-4 Two subnets

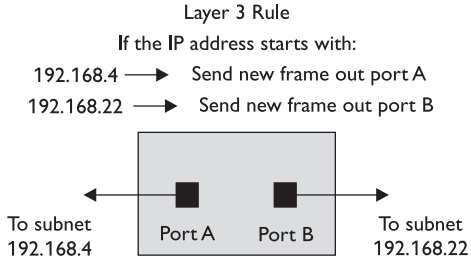


Figure 10-5 Layer 3 switch at work

numbering system. Some protocols, the most famous being NetBIOS/NetBEUI, have no universal numbering convention, making them unusable in networks that use routers.

Test Specific

Implementing Protocols

Now that you have a more detailed idea of what network protocols do for our networks, let's investigate some common issues shared by all network protocols. First, many networks use more than one protocol. Second, this creates a need to determine which protocols go with which NIC, a process called *binding*. Finally, you need to do the actual installation of network protocols.

Multiple Protocols

Most of the time, every system on the network will use the same protocol, but situations do exist where some systems are set up on purpose to run a different protocol from others. Figure 10-6 shows four networked systems. Systems *A* and *B* use the TCP/IP network protocol, while systems *C* and *D* use IPX/SPX. For any two systems to communicate on a network, they must use the same network protocol. Systems *A* and *B* can see each other, but not systems *C* and *D*; systems *C* and *D* can see each other, but not systems *A* and *B*. Why would anyone break their network like this on purpose? Well, they probably wouldn't—but it's important to understand that no computer can see another computer on the network unless they use the same protocols.

You may have more than one network protocol on the same system. Let's add another system, System *E*, to the previous diagram. System *E* has both the IPX/SPX and the TCP/IP network protocols installed. As a result, System *E* can see every system on the network! (See Figure 10-7.)

Hey! If System *E* sees both networks, can we fix System *E* so it acts as a kind of translator between the two sets of systems? Well, yes, you can—but hold onto that concept for just a moment as we discuss one more important concept: binding.

Binding

If a single system can have multiple NICs and multiple protocols, there needs to be a way to decide which NICs use which protocols for which transactions. The solution to this challenge is called *binding*. Every protocol installed on a system must be bound to one or more NICs, and every NIC must be bound to one or more specific protocols.

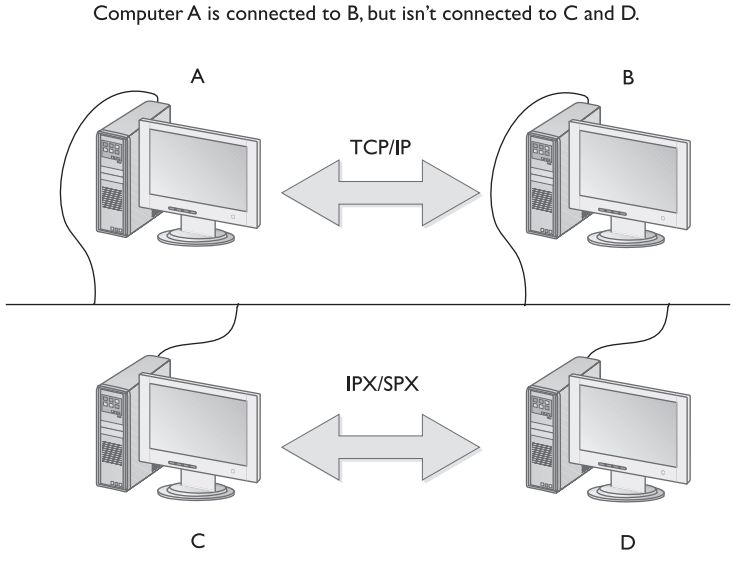


Figure 10-6 Systems *A* and *B* use TCP/IP, while systems *C* and *D* use IPX/SPX.

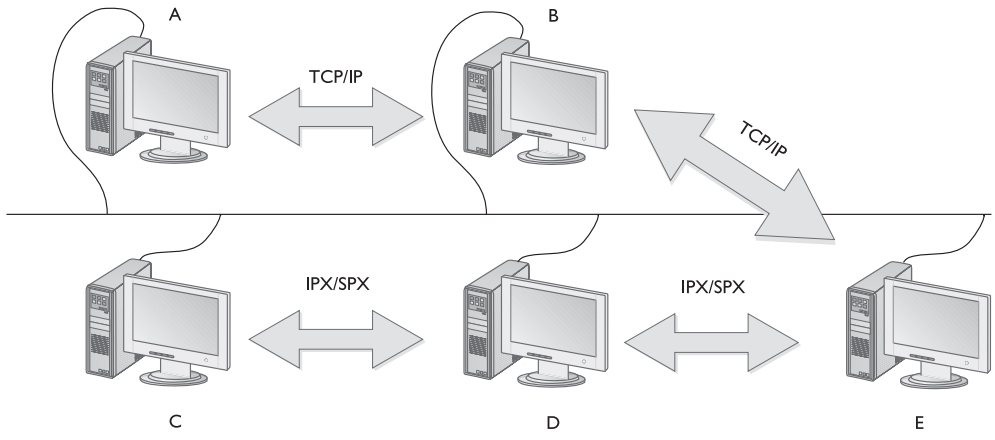
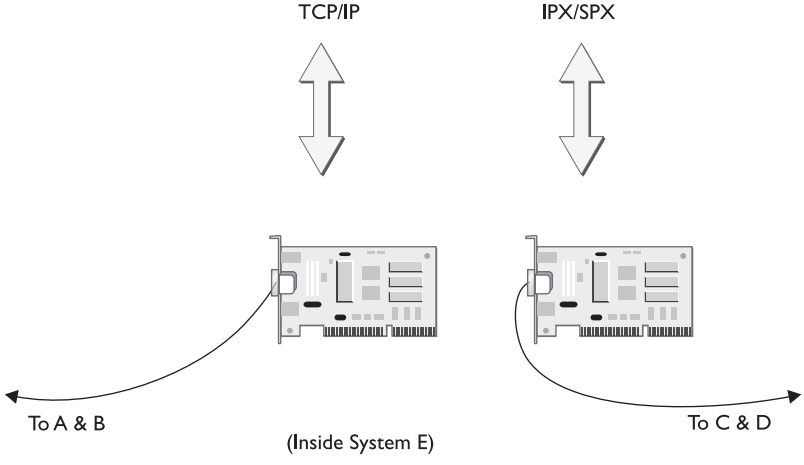


Figure 10-7 System *E* has both protocols installed and can thus see them all!

Look at Figure 10-7 and think about System *E* for a moment. You can correctly assume that both the IPX/SPX and TCP/IP protocols were bound to System E's NIC. Now look at the situation shown in Figure 10-8. In this case, System *E* has two NICs —one connected to systems *A* and *B,* and the other connected to systems *C* and *D*. This situation calls for binding IPX/SPX to one NIC, and TCP/IP to the other. Fortunately, Windows makes this binding process extremely easy. We'll save the actual process of binding for later. For now, just remember, at least one protocol must be bound to each NIC in a networked system.

Figure 10-8 TCP/IP is bound to one NIC, while IPX/SPX is bound to the other.



TIP At least one network protocol must be bound to each NIC in a networked system.

Before adding a new system to a network, you should check one of the existing systems to determine which network protocol the network uses. Almost all modern networks use TCP/IP, but older network operating systems tend to use other protocols. Older versions of Novell NetWare, for example, may use IPX/SPX, while a network containing only Windows 9*x* systems may use the NetBIOS/NetBEUI protocol.

Installation

Every computer on your network requires a network protocol and every network protocol needs installation. Despite the impression given to us by many operating systems, not every part of every network protocol's software is built into the operating system. Right now, TCP/IP is so predominant that every operating system preinstalls TCP/IP (assuming the operating system detects a NIC!), giving the impression that TCP/IP is somehow one and the same as the operating system. Preinstalling TCP/IP into your system is a pleasant convenience, but if you want to use another network protocol, you're probably going to have to do some installing and some configuring. Most versions of Microsoft Windows support all the common network protocols; depending on the version of Windows you use, you may or may not need to install a protocol. Regardless of the protocol you want to install, it's important to know where to go to do this on your Windows computers. In all Windows 9*x* systems, alternate-click (right-click) Network Neighborhood and select Properties to open the Network Properties dialog box. You may also click the Network Control Panel applet to get to this dialog box (Figure 10-9).

Look closely at Figure 10-9 and the line that starts with TCP/IP. Do you see the small arrow that points to the name of the NIC (AMD PCNET Family…)? That arrow shows TCP/IP is bound to that NIC.

NOTE Windows Me works the same way as Windows 9*x*, but Microsoft changed the name of Network Neighborhood to My Network Places.

Windows 2000, Server 2003, and XP have a different way to get to your protocols. Instead of a list showing all your networked devices and all your protocols, 2000 and XP separate each network connection into its own configuration dialog box. Alternate-click My Network Places and select Properties to open the Network Connections window (Figure 10-10). This window shows all the network devices on a system. Many computers have multiple network devices. Figure 10-11 shows a laptop system with lots of network connections!

To get to the protocols, alternate-click the connection you want to change and select Properties to get to that connection's Properties dialog box (Figure 10-12).

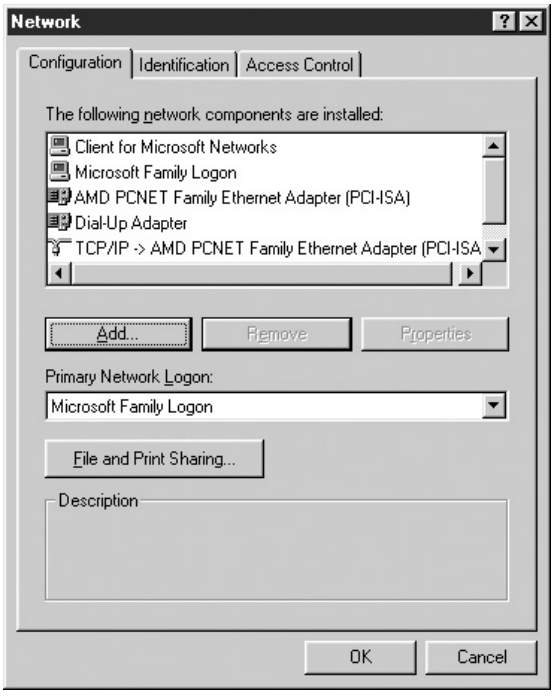


Figure 10-9 Windows 98 Network Properties

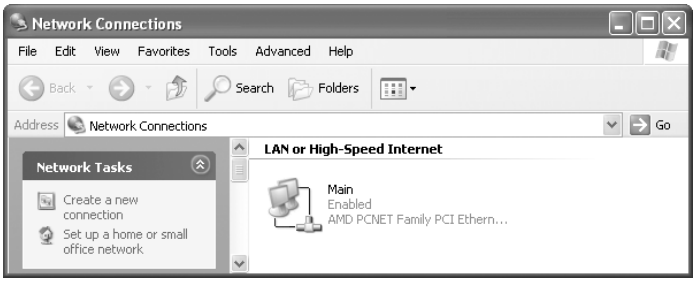


Figure 10-10 Windows XP Network Connections

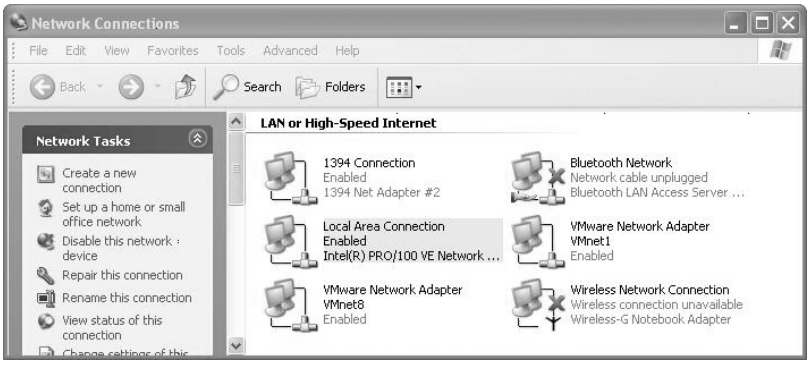


Figure 10-11 Lots of network connections!

Along with installation comes configuration. As we dive into the different network protocols, you'll soon discover that with the exception of TCP/IP, network protocols re-

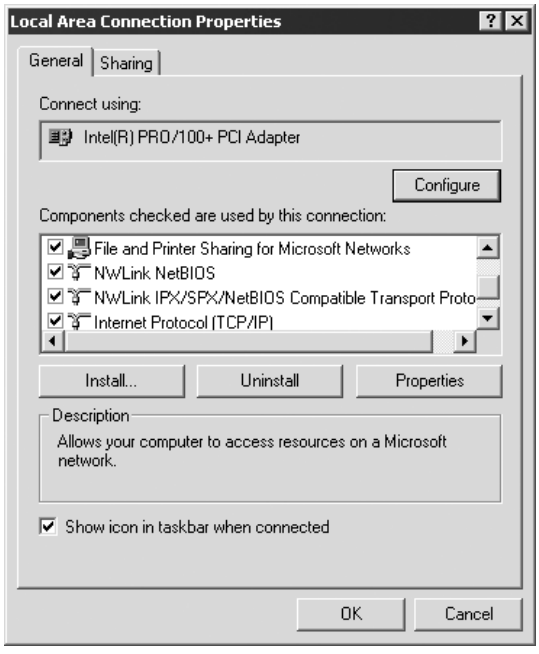


Figure 10-12 NIC Properties box in Windows 2000

quire surprisingly little configuration. Almost anything you want to do to a network protocol in terms of configuration takes place under the Network Properties.

Protocol Concepts

In this section, you've learned about a number of important protocols concepts. Using the OSI seven-layer networking model as your guide, you now have a pretty strong understanding of exactly what the network protocol software's doing on your systems. Now it's time to stop talking concept and dive into the three most common protocol suites: NetBIOS/NetBEUI, IPX/SPX, and TCP/IP.

NetBIOS/NetBEUI

NetBIOS/NetBEUI provides a fast, simple set of network protocols appropriate for use in smaller LANs. NetBIOS/NetBEUI was the primary network protocol stack used in Windows 9*x* and NT systems. Today, NetBIOS/NetBEUI primarily exists to support Microsoft networking using Windows NT or Windows 9*x* systems.

NOTE NetBIOS stands for Network Basic Input/Output System; NetBEUI is short for NetBIOS Extended User Interface.

Windows 2000 and XP still support NetBIOS/NetBEUI, but Microsoft is moving away from NetBIOS/NetBEUI for the more universal TCP/IP. This moving away is best illustrated by how Windows 2000 and Windows XP support NetBIOS/NetBEUI. Both Windows 2000 and Windows XP still support NetBIOS. However, Windows 2000 does not install NetBEUI by default, and installing NetBEUI in Windows XP requires locating the NetBEUI protocol software from a special location on the XP installation CD.

NetBIOS/NetBEUI's speed and ease of configuration make it a good choice for small networks, but because NetBEUI does not support routing, it is totally unacceptable for any but the smallest (fewer than 30 systems) networks. The NetBIOS/NetBEUI protocol stack contains two main protocols—in this case, NetBIOS and NetBEUI—which operate at the Session layer and the transport layer, respectively (see Figure 10-13).

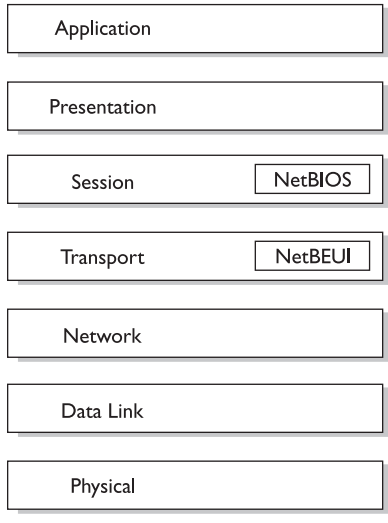


Figure 10-13 NetBIOS operates at the Session layer, while NetBEUI operates at the Transport layer.

One big benefit of NetBEUI, and a reason this protocol remains popular, is this: for simple networks, NetBIOS/NetBEUI requires no configuration—it just works. You'll appreciate this more when you see how much configuration TCP/IP takes! Let's start with a little history, and then look at the two parts of NetBIOS/NetBEUI—first NetBIOS, and then NetBEUI—to see how it all works

In the Early Days

Starting with DOS and continuing to the latest Windows operating systems, Microsoft has enjoyed tremendous success in the PC world. One of the many reasons for

Microsoft's success stems from its concentration on the customer's needs. Microsoft has always tried hard to make computing as easy for the user as possible. This attitude certainly came into play as Microsoft was choosing how to make network protocols for PC networks. By the mid-1980s, TCP/IP was well established as a network protocol for the Internet, but back then, the Internet wasn't anything like it is today. Microsoft—like everyone other than the geekiest of geeks—saw no reason to use the free, but complex and difficult to configure, TCP/IP. Instead, they chose to design a much simpler protocol: NetBIOS/NetBEUI. The simplicity of NetBIOS/NetBEUI comes, in part, because it's designed to share only folders and printers—why would anyone want to share anything else?

When Windows rolled around, Microsoft made a corporate decision that still haunts us today. Microsoft decided to build NetBIOS into the core of the Windows operating system. When you installed Windows, you were prompted to give the computer a name. That name was the NetBIOS name. The problem was that no one (again, except for the geekiest of geeks) appreciated that this was taking place. Finding the word NetBIOS anywhere on the computer was hard, and as a result, people never tied in the fact that the name of the computer was the NetBIOS name—it was just the "name of the computer." While this tight integration made life easy for those who used the first generations of Windows networks, it put Microsoft in a bad place. By tying NetBIOS into the operating system so tightly, it made things challenging years later when people wanted to use other network protocols—like TCP/IP or IPX/SPX that used completely separate and incompatible naming conventions—on their Windows computers. While we'll delve into this naming fiasco in detail in the next two chapters, keep in mind that on the first generations of Windows: NT, 95, 98, and Me, you had no way to turn on or turn off NetBIOS—it was just there!

TIP Over the years, separating NetBIOS from NetBEUI has become popular. This is because of a number of factors. First, Microsoft has always hidden NetBIOS from users. Second, Microsoft shows NetBEUI as a separate installable protocol. Third, NetBIOS most commonly now runs with TCP/IP. Despite the march of time, the fact remains that NetBIOS and NetBEUI were originally designed to work together as NetBIOS/NetBEUI.

NetBIOS at Session

Microsoft's NetBIOS handles the Session-layer functions for NetBIOS/NetBEUI networks. NetBIOS manages sessions based on the names of the computers involved. A NetBIOS name is based on a system's network name, which you can designate using the Network applet in the Control Panel. Figure 10-14 shows an example of the network name of a Windows 98 system, displayed by the Network applet.

NetBIOS names are made up of a system's network name, as specified in the Network applet, followed by a function-specific suffix. The system's network name can contain up to 15 characters. Each character is represented by a single 8-bit (one byte) ASCII code. For example, the 8-bit ASCII code 01100101 (65h in hexadecimal format), represents the capital letter *A*. NetBIOS limits the network name to 15 bytes, or characters, because it

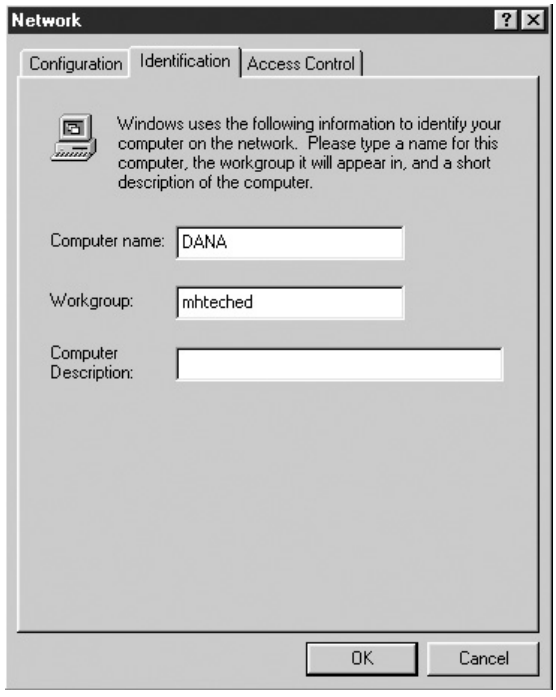


Figure 10-14 The Windows 98 Network Control Panel applet displays the computer's name.

reserves the 16th byte for the special number function code that defines the role the machine will play on the network in that particular session. A NetBIOS machine can take on several roles, depending on the needs of the session.

TIP Windows 2000 and XP use TCP/IP's *Domain Name Service (DNS)* to give individual computers more descriptive names than just an IP address. DNS names are much less restrictive than NetBIOS names, and this can cause some confusion with Windows. When you install a Windows XP system and give the new computer a name longer than 15 bytes or one that uses special characters, you'll get a warning screen that tells you the name is invalid or has been shortened. That's because NetBIOS systems cannot see any system that uses a name that does not fit into the NetBIOS naming convention.

Table 10-1 lists the common 16th byte codes used by NetBIOS to define the server and client functions of a machine.

Don't worry about memorizing all the functions and 16th byte codes listed in the table. Instead, let's look at an example using the three most commonly used extensions to understand how NetBIOS manages a session.

Hannah, a friendly neighborhood network tech, installs three Windows systems—named MHTECHED, JANELLE, and DANA—on her network. Hannah does not need to specify these NetBIOS names. NetBIOS, operating in the background, determines the

Table 10-1 NetBIOS Names and Functions

16th byte	Function
<00>	Workstation Service Name. The name registered by clients on the network.
<03>	Messenger Service Name. Used by applications such as WINPOPUP and NET SEND to deliver messages.
<1B>	Domain Master Browser
<06>	RAS Server
<1F>	NetDDE Service
<20>	File and Printer Server
<21>	RAS Client
<BE>	Network Monitor Agent
<BF>	Network Monitor Utility

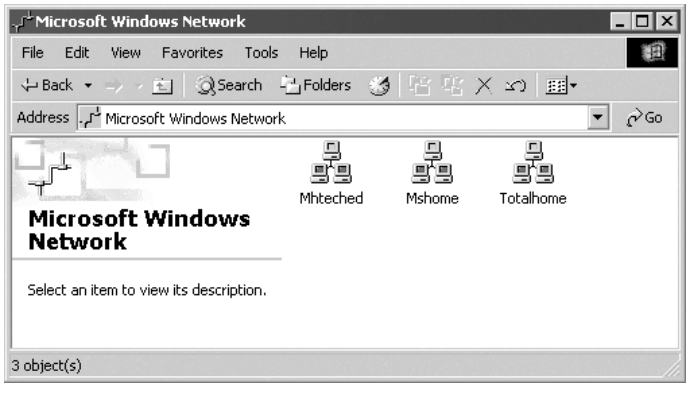
NetBIOS names automatically based on the computer names (MHTECHED, JANELLE, and DANA) that Hannah selected for these systems. Even though these names are determined at installation, you can easily change them. In Windows 98, get to the Network Properties applet and click the Identification tab (refer to Figure 10-14).

You can also specify the *workgroup* name here. NetBIOS contains the capability to group computers together into workgroups. Workgroups are nothing more than a convenient way to organize computers under Network Neighborhood/My Network Places (Figure 10-15). They do not provide any form of security. More advanced versions of Windows provide a much more powerful grouping called a domain—we'll save the details of Windows workgroups and domains for the next chapter.

NetBIOS also supports an optional descriptive computer name. These descriptive names do as they are named, providing a handy method to describe the machine in more detail than the NetBIOS name provides.

By default, all Windows computers act as clients. Any Windows computer can also act as a server, but must first be configured. Hannah configures the MHTECHED and JANELLE systems to act as servers, leaving the DANA system as only a client. According to what you've just learned, MHTECHED now has at least two names: MHTECHED<00>,

Figure 10-15 Typical Workgroups



identifying MHTECHED as a client, and MHTECHED<20>, identifying MHTECHED as a file and print server. JANELLE also has two names: JANELLE<00>, identifying JANELLE as a client, and JANELLE <20>, identifying JANELLE as a file and print server. DANA, by contrast, registers only one name, as a client: DANA<00>.

NOTE Any real machine using NetBIOS on a Microsoft Network will register several more names, to support other, less obvious functions. Those additional names have been left out of this discussion for the sake of simplicity.

When Hannah sits at JANELLE and accesses a file on MHTECHED, both JANELLE and MHTECHED must manage that connection. To open the connection, JANELLE the client, aka JANELLE<00>, opens a connection with MHTECHED the server, aka MHTECHED<20> (see Figure 10-16). As MHTECHED begins to send the requested file to JANELLE, another user, Barbara, sits down at DANA and opens another file on MHTECHED (see Figure 10-17). Each of the computers keeps track of these simultaneous conversations using their NetBIOS names (see Figure 10-18).

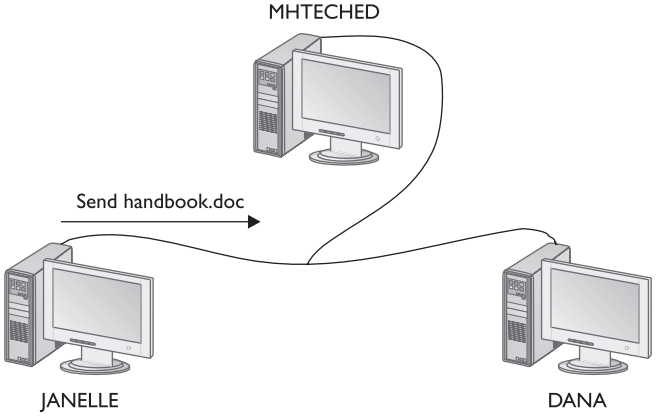


Figure 10-16 JANELLE the client opens a connection with MHTECHED the server.

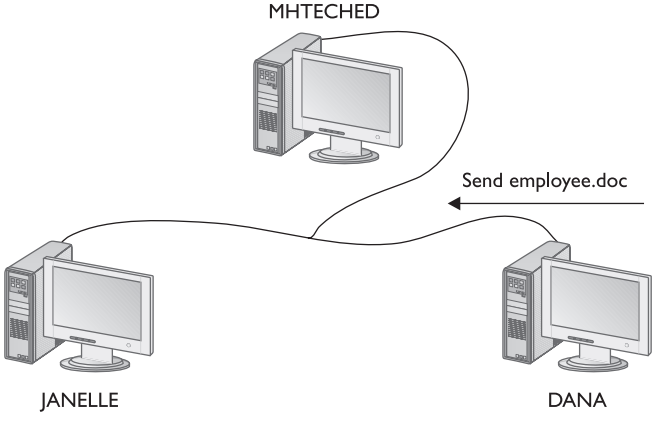


Figure 10-17 DANA the client opens a connection with MHTECHED the server.

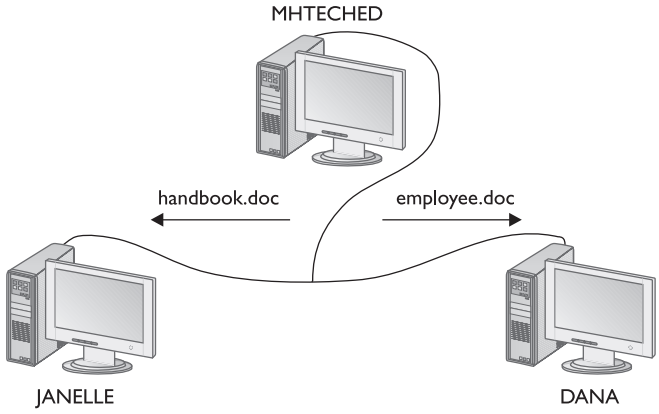


Figure 10-18 MHTECHED, JANELLE, and DANA use NetBIOS names to manage their connections.

By using a different NetBIOS name for each function, networked systems can keep track of multiple connections among them simultaneously. For example, let's say Barbara sits at MHTECHED and opens a file on JANELLE, causing MHTECHED<00> to establish a connection with JANELLE<20>. At the same time, Hannah can sit at JANELLE and open a file on MHTECHED, causing JANELLE<00> to establish a connection with MHTECHED<20>. The capability to use unique NetBIOS names for each server (<20> suffix) and client (<00> suffix) function enables MHTECHED and JANELLE to hold two (or more) simultaneous conversations (see Figure 10-19).

Without a NetBIOS name for a particular function, a system cannot perform that function when requested by another node on the network. For example, if Hannah sits at JANELLE and attempts to open a file on DANA, JANELLE will be unable to establish the connection. Why? Because DANA is not configured to function as a server. The request from JANELLE for a connection to DANA is addressed to DANA<20>. But the NetBIOS name DANA<20> does not exist; DANA can respond only to the client NetBIOS name DANA<00> (see Figure 10-20). When it sees the message for DANA<20>, DANA just assumes it's for some other system and ignores it; DANA doesn't even send a refusal message back to JANELLE.

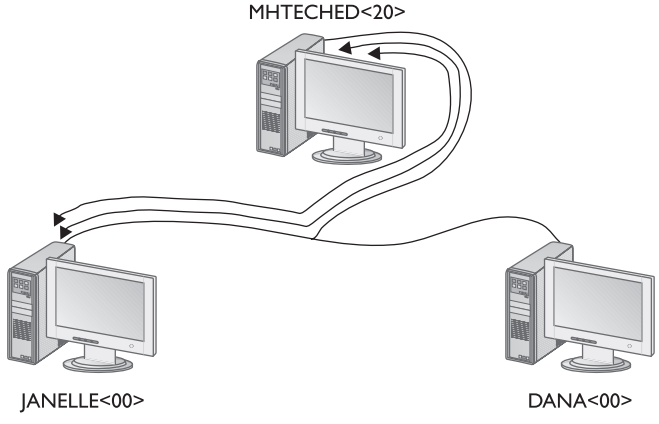


Figure 10-19 JANELLE and MHTECHED can have multiple conversations simultaneously.

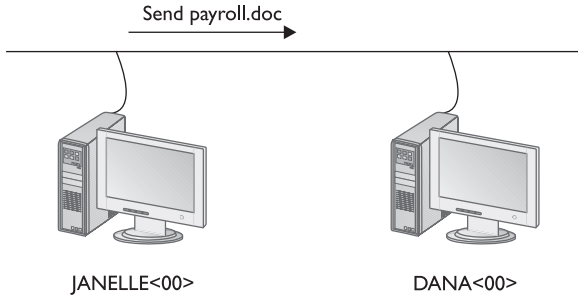


Figure 10-20 DANA ignores JANELLE because DANA<20> is not one of its NetBIOS names.

NOTE Even though Windows 2000 and XP no longer use the NetBIOS naming convention, all of the NetBIOS processes you've just learned—known as Server Message Blocks (SMBs)—are still very much a part of these newer operating systems. The way computers are named have changed, but everything else still works the same!

NetBEUI at Transport

NetBEUI functions at the Transport layer within the NetBEUI protocol suite, breaking larger chunks of data into smaller pieces on the sending machine, and reassembling them on the receiving end (see Figure 10-21). The NetBEUI protocol requires no setup beyond installation by the network tech. While its operational simplicity makes NetBEUI attractive for smaller networks, it deprives NetBEUI of a capability vital to larger networks: routing.

The NetBEUI protocol skips the Network layer and communicates directly with the data-link layer. When a router receives a NetBEUI packet, it doesn't find the routing information it needs, so it simply discards the packet (Figure 10-22).

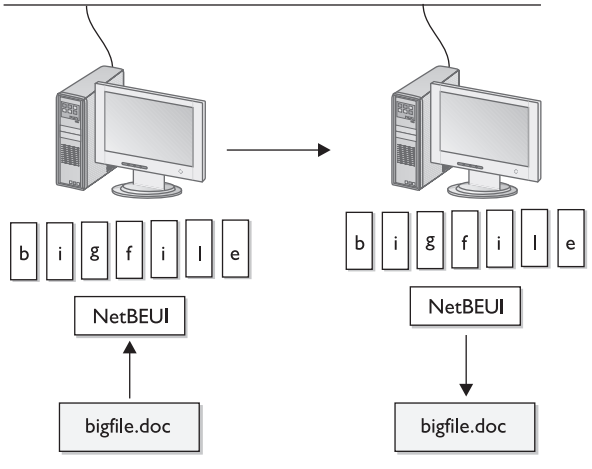


Figure 10-21 NetBEUI breaks the file into smaller pieces for transmission and reassembles the pieces on the receiving end.

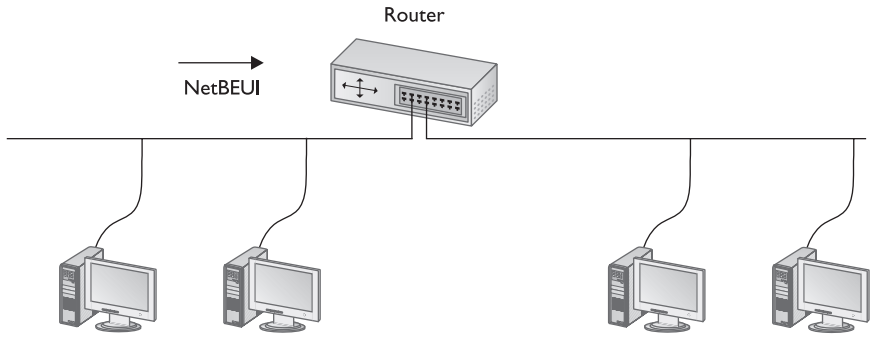


Figure 10-22 Routers discard NetBEUI packets.

NOTE The NetBEUI protocol suite's lack of any Network layer protocol illustrates the key weakness of the OSI model: not every network protocol follows the OSI completely.

NetBIOS/NetBEUI Naming Weaknesses

While NetBIOS provides an adequate means for managing connections on a small network, it does not scale well for larger networks. NetBIOS uses what is called a *flat name space,* meaning that the names for every machine in a network are drawn from one pool. Thus, the base NetBIOS name for each computer must be unique. Imagine a world where people only had first names. To prevent confusing any two people, no two people could have the name Mike, or Bob, or Johnny. Instead, people would have to come up with unique names like Johnny5, Fonzie, and Bluto. Finding unique names for a dozen people presents no problem. Placing a few thousand people in the same flat name space creates a big problem. In real life, most people have at least two, and sometimes as many as four or more names, and even then we often need other information, like addresses and identifying numbers, to tell people apart.

Network administrators working in a first-name-only NetBIOS world are often driven to give their systems bizarre, nondescriptive names, creating many administrative headaches. NetBIOS names are so restricted, it's hard to be usefully descriptive. On a network with only one server, simply calling that machine SERVER works fine. But let's take a more realistic example. Simon's network has 20 servers: ten accounting servers, five web servers, four file servers, and an e-mail server. Simon usually refers to one of his servers as "'Accounting Server 7"' in conversation, but he can't use that as the NetBIOS name for the machine. Remember, NetBIOS names must contain 15 or fewer characters (not counting the special 16th character that designates the machine's function). Instead of Accounting Server 7, Simon must name the server ACCOUNTSERV7. Not bad, but not optimal, and Simon's network is a relatively modest one by commercial standards.

The problem of ensuring name uniqueness is much more extreme in large WAN environments run by multiple administrators. In a large WAN run by 40 different administrators, guaranteeing that no two administrators ever assign the same name to any two of their 5000+ machines becomes an administrative nightmare, requiring extensive planning and ongoing communication. This is why network architects prefer a more scaleable naming scheme, such as the TCP/IP protocol suite's Domain Name Service (DNS—see Chapter 11, "TCP/IP"), for larger networks.

Installing NetBIOS/NetBEUI

The nature of NetBIOS's close connection in a Windows system compels us to install NetBIOS separately from NetBEUI. If you're using Windows NT or 9*x*, NetBIOS is automatically, permanently installed. Windows 2000 and Windows XP have replaced NetBIOS with TCP/IP's DNS (see Chapter 11). However, they respect the fact that you may want to connect to older computers that use NetBIOS and also install NetBIOS, but enable you to turn it off if you want to do so. Installing NetBIOS on a Windows computer is moot—you get NetBIOS. There is a way to remove NetBIOS, but it will also remove networking from your Windows machine!

NetBEUI is another issue entirely. Realizing long ago that NetBIOS might not run with NetBEUI, Microsoft treats NetBEUI as a standalone network protocol that must be installed via the Network Control Panel applet described earlier. Let's go through two installations of NetBEUI, once with Windows 98 and again with Windows 2000.

Windows 9*x*

To add the NetBEUI protocol to a Windows 9*x* system, click Add on the Configuration tab of the Network applet (see Figure 10-23). A Select Network Component Type screen appears. Select Protocol and click Add (see Figure 10-24). A Select Network Protocol screen appears. Select Microsoft as the Manufacturer and NetBEUI as the Network Protocol (see Figure 10-25).

NOTE The tradition in the industry of referring to entire protocol suites by the name of one or two of their constituent protocols has caused unnecessary confusion, pain, suffering, and gnashing of teeth. Read carefully whenever you see the word *protocol*—sometimes a writer means a specific protocol, but sometimes he or she means a protocol suite.

While you're still in the Network applet, head back to the Configuration tab and see if you can locate the Client for Microsoft Networks, and File and Print Sharing for Microsoft Networks (see Figure 10-26). Client for Microsoft Networks is NetBIOS, plus a few extra tools to make Windows networking function. Windows 98 calls its server component File and Printer Sharing for Microsoft Networks. To configure DANA as a client only, Hannah installs NetBIOS and the Client for Microsoft Networks. Hannah has created a three-node network, and she and her coworkers can sit down to do some serious gaming, errr . . . oops! I mean *work*.

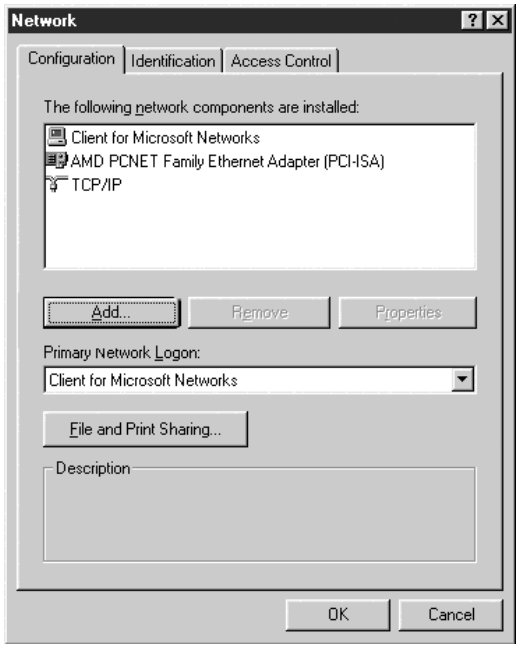


Figure 10-23
The Configuration tab of Windows 98's Network applet

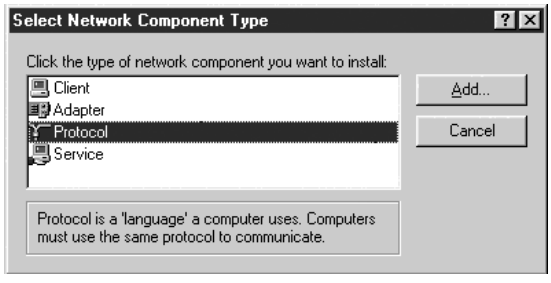


Figure 10-24
Selecting Protocol

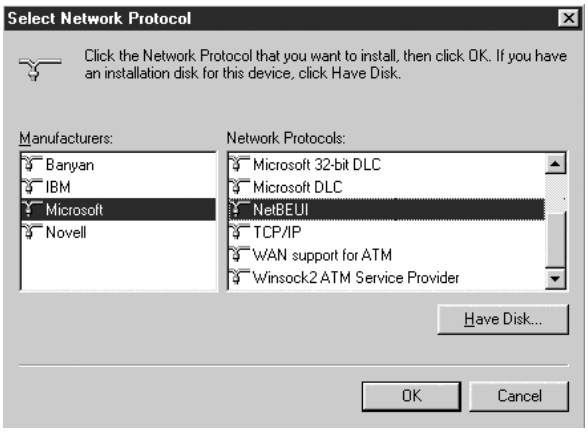


Figure 10-25
Selecting Microsoft, NetBEUI

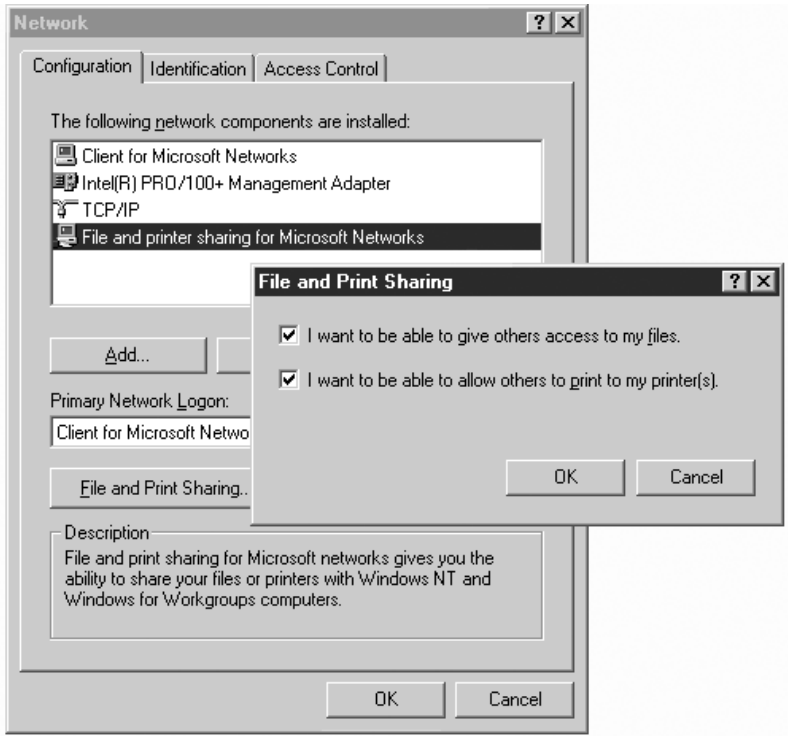


Figure 10-26 Installing File and Printer Sharing for Microsoft Networks

Windows 2000

Windows 2000 does not have NetBEUI support by default. Fortunately, installation is not difficult; in many ways, it parallels the installation process done in Windows 98. The big difference between Windows 2000 (and XP) and Windows 98 is you must choose the NIC you want to bind the protocol to, and then select properties for that NIC (Figure 10-27).

From this point, the process is pretty much identical to Windows 98. Click the Install button, select Protocol, and then select NetBEUI. You'll then see the NetBEUI protocol installed (Figure 10-28).

Installing NetBEUI in Windows is easy to do if you know where to go. In fact, this is the process to go through to install any protocol on a Windows system.

NetBIOS/NetBEUI—Fading Away

NetBIOS's reliance on a flat name space makes it difficult to use in large WAN environments, but its simplicity makes it an ideal choice for smaller LANs. As long as the network tech assigns every computer a unique name, NetBIOS does a fine job. Even though NetBIOS does a fine job on smaller networks and Microsoft has done some bits of magic

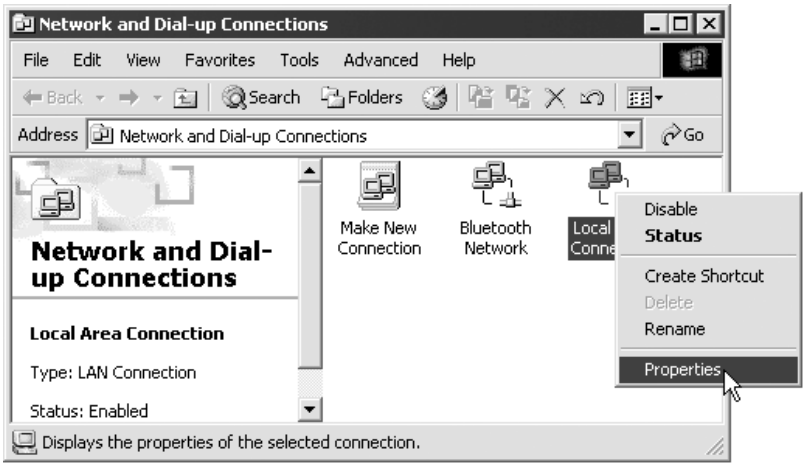


Figure 10-27 Selecting a NIC's properties

to make NetBIOS work on larger networks, NetBIOS is fading away, replaced by the more universal DNS. Equally, NetBEUI does a fine job at the Transport layer, but its lack of any Network layer functions make it unsuitable for larger networks.

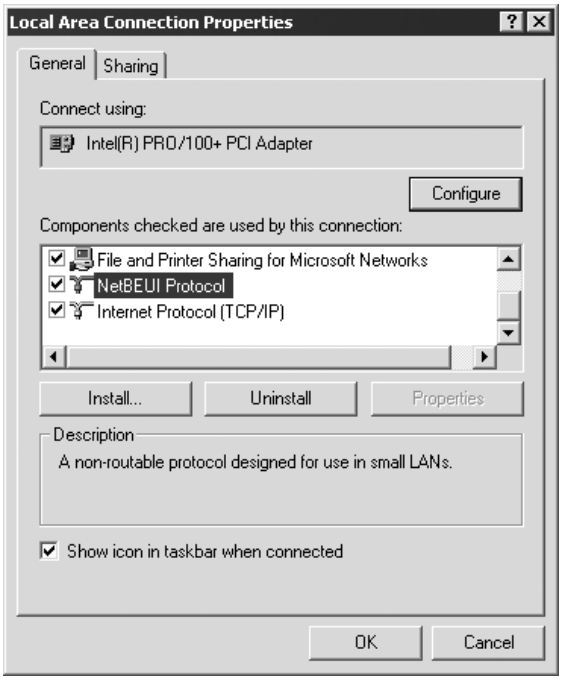


Figure 10-28 NetBEUI installed on Windows 2000

IPX/SPX

Novell's *IPX/SPX* protocol suite, used primarily by Novell NetWare-based networks, provides a more scaleable solution for networks compared to NetBIOS/NetBEUI. While the NetBIOS/NetBEUI protocol suite provides services at the Transport and Session layers, IPX/SPX includes a wide variety of protocols operating at OSI layers three through seven (the Network layer through the Application layer). Although more scaleable than NetBEUI, IPX/SPX bogs down in large networks due to excessive traffic. The latest versions of Novell NetWare still support IPX/SPX, but they default to TCP/IP.

TIP Make sure you know that IPX/SPX is mainly for Novell NetWare!

In a Novell NetWare network, IPX/SPX operates at layers 3 through 7 of the OSI model. Figure 10-29 shows how various IPX/SPX protocols relate to the OSI layers. The *NetWare Core Protocol (NCP)* handles a variety of Presentation and Application layer issues, the *Server Advertising Protocol (SAP)* handles the Session layer. SPX works at the Transport layer and IPX handles all the network functions.

NOTE The Network+ exam does not require knowledge of the individual protocols that make up the IPX/SPX suite.

NCP/SAP at Session

The most important aspect about IPX/SPX is that Novell invented this protocol to work in a client/server environment. Remember, Novell NetWare is not a client operating system, so most of your network configuration job is done to your NetWare servers. After

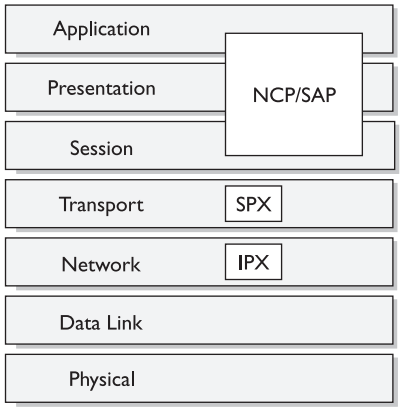


Figure 10-29 IPX/SPX includes protocols operating at OSI layers 3 through 7.

you set up your servers, you then configure your clients to whatever settings you made. This is interesting because it makes networking a bit simpler as, like NetBIOS/NetBEUI, you have a limited number of functions to configure, and those functions are all centered on file and print sharing.

NetWare servers have NetBIOS-like names but that is only for ease of use. In reality, all NetWare servers have a unique, eight hexadecimal character, *internal network number* that identifies that one server. A server's internal network number, for example, might be 87654321. In a typical NetWare network, servers advertise their presence to all clients (and to other servers) by broadcasting their internal network numbers every 60 seconds. Clients keep a list of the known servers and use this information to contact the servers as needed.

NCP supports all the applications used in a NetWare IPX/SPX network. When a client makes a request to a service, the server's NCP functions handle the session, presentation, and application functions on its side to enable a data transfer.

IPX/SPX at Transport and Network

If you check the official NetWare documentation, you'll see that SPX works at the Transport layers, while IPX works at the network layer. However, these two protocols are closely intertwined and are best discussed together. NetWare running IPX/SPX creates subnets by adding a special *external network number* (often referred to as "network number" causing confusion with the earlier mentioned internal network number, or as "segment address") to the MAC address of every computer on the network. Internal network addresses define servers, while external network numbers define every computer on the network—a subtle, but important, difference (see Figure 10-30).

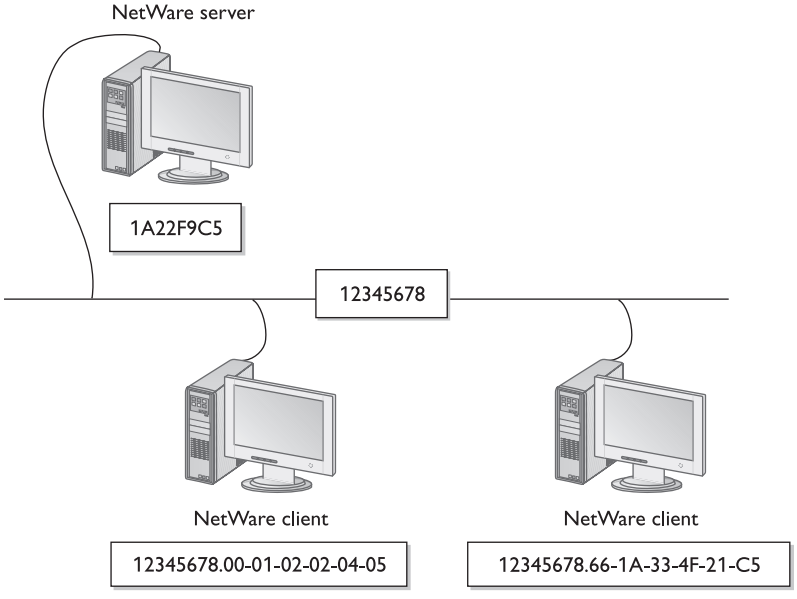


Figure 10-30 IPX/SPX naming conventions

When you install NetWare client software on your Windows computer, you must enter the name of a preferred server. Each client's preferred NetWare server assigns this network number to the client computers. IPX/SPX network numbers are eight hexadecimal digits. So, a single subnet would be the entire set of computers that share the network number of 12345678. Computer 12345678.00-01-02-02-04-05 and Computer 12345678.66-1A-33-4F-21-C5 are on the same subnet.

IPX/SPX, although routable, does not scale well for large WANs. Novell designed IPX/SPX to support its NetWare operating system, which treats NetWare servers as the ultimate focus of the network. In a NetWare environment, servers are servers and clients are clients, and never the twain shall meet. Unlike NetBEUI, which assumes that a machine can function as both a client and a server, IPX/SPX assumes that a proper network consists of a few servers and a large number of clients. While this configuration works well on small- and medium-sized networks, when a network grows to include hundreds of servers and thousands of clients, the increase in SAP traffic will take down the network. Until relatively recently, the danger of excessive SAP broadcasts did not impact the typical network tech, because most networks simply did not have enough servers for SAP broadcasts to cause congestion. But, as large networks become more common, IPX/SPX's reliance on SAP broadcasts becomes more of a problem. This has led most WAN designers to adopt a more scaleable alternative: TCP/IP.

Installing IPX/SPX

Installing IPX/SPX usually means installing either the Microsoft or the Novell clients for NetWare. Microsoft takes advantage of the widespread industry support of IPX/SPX with its own version of the protocol suite, referred to as either IPX/SPX-compatible Protocol (Windows 9*x*) or NWLink IPX/SPX/NetBIOS Compatible Transport Protocol (Windows 2000/XP/2003) (see Figure 10-31). Microsoft network operating systems, including Windows 9*x*, Windows 2000, and Windows NT, use IPX/SPX for two purposes: to connect to NetWare servers (usually), and to provide the transport and network layer functionality for Microsoft Networking (rarely). In the latter case, Windows uses NetBIOS over IPX/SPX. Windows 2000/XP/2003 manifest this clearly by adding a second protocol called *NWLink* NetBIOS when you install IPX/SPX.

TIP Rarely do techs manually install IPX/SPX in Windows systems. In most cases, they install either the Microsoft or the NetWare client and these clients, in turn, install IPX/SPX.

Unlike NetBIOS/NetBEUI, which requires no configuration beyond assigning each computer a name, IPX/SPX requires that a network tech potentially make configurations. The first area that may need configuration is the frame type. IPX packets vary in their format according to the data-link layer protocol used. IPX running on top of Ethernet, for example, can use one of four data structures, called frame types: Ethernet 802.3, Ethernet II, Ethernet 802.2, and Ethernet SNAP. If two network nodes use different frame types, they will be unable to communicate.

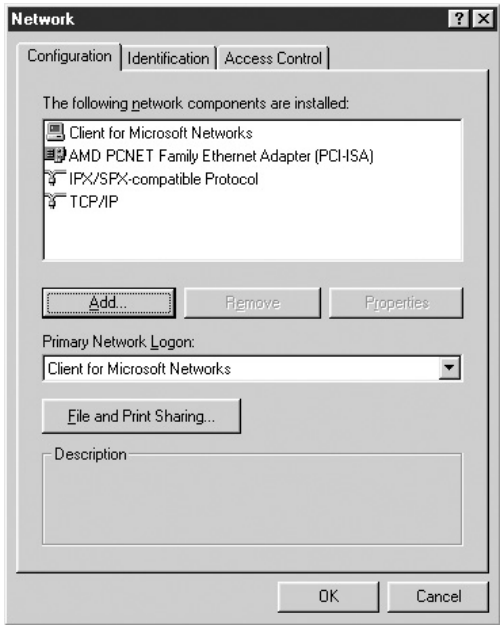


Figure 10-31 Microsoft calls its version of the IPX/SPX protocol suite either IPX/SPX-compatible Protocol or NWLink IPX/SPX/NetBIOS Compatible Transport Protocol.

NOTE Determining Ethernet frame types is not unique to IPX/SPX. All protocols need to identify the type of Ethernet frame used on the network. It's just that NetBIOS/NetBEUI and TCP/IP protocols determine the frame type automatically, while IPX/SPX traditionally does not.

In the days of DOS-based network clients, network techs set the frame type manually for each system on the network. All versions of Windows simplify the process by automatically detecting Ethernet traffic on the network and configuring themselves to use the first frame type they detect (see Figure 10-32). Because modern Windows systems automatically use whatever frame type they detect first, however, systems on networks using multiple types can end up trying to communicate using mismatched frame types. This is a rare situation, but one that is most probable to manifest itself in NetWare networks.

To ensure that every system on a network uses the same frame type, a system admin can set each system's frame type manually using the Network (Connections) applet (see Figure 10-33). The structural details of the different frame types do not affect the network tech—simply configure all systems to use the same frame type.

On some NetWare networks, you may need to enter the internal and external network numbers. This is usually done in the same dialog box as the frame type. Figure 10-34 shows the NWLink Windows XP dialog box setting the network numbers.

Splitting Protocols I: NetBIOS over IPX/SPX

Network protocol stacks aren't designed to work with anything but their own family of protocols. When Novell made IPX/SPX, for example, the company wasn't interested in it

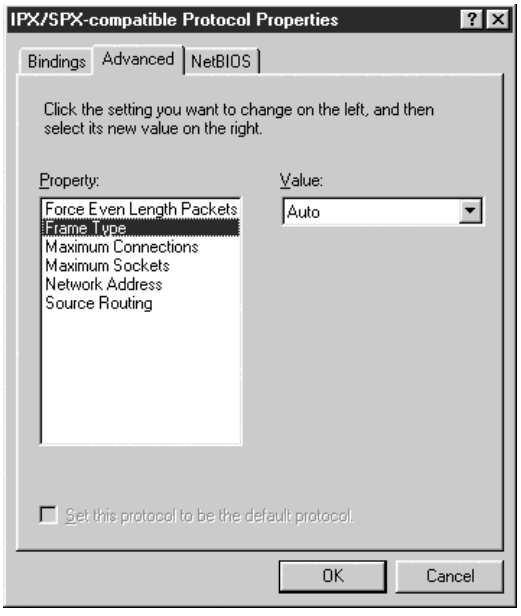


Figure 10-32 Modern Windows systems can autodetect the frame type being used.

working with other network protocols. The folks who made TCP/IP assumed that you would never use anything but TCP/IP, and Microsoft thought the same thing when they adopted and developed NetBIOS/NetBEUI. But, over the years, situations developed to change this attitude.

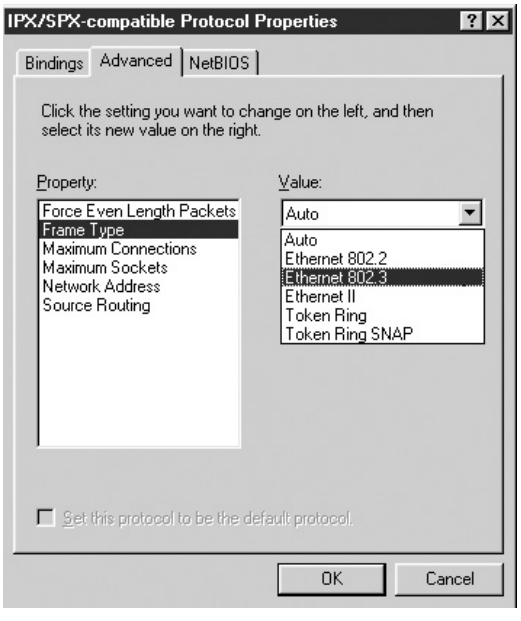


Figure 10-33 Setting the frame type manually

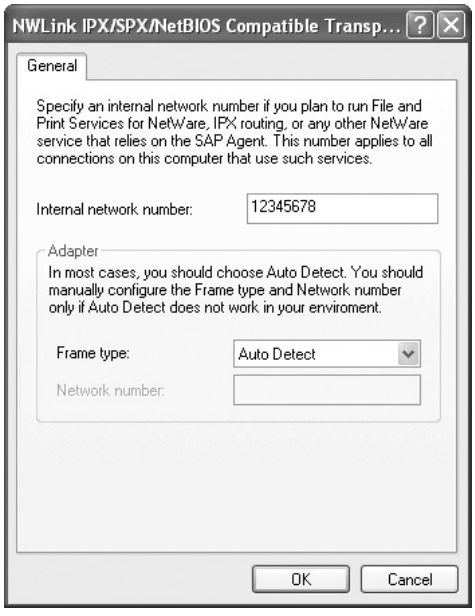


Figure 10-34 Setting NetWare network numbers

NOTE Technically, Microsoft didn't invent NetBIOS. IBM was the originator of NetBIOS, but Microsoft added much more functionality to NetBIOS, and adopted it for DOS and Windows PCs.

Microsoft's idea to incorporate NetBIOS closely into the operating system created a number of serious cross-platform issues. NetBIOS was designed to run with NetBEUI, but other systems, running other protocols, forced Microsoft to realize that the NetBEUI protocol was separating Windows computers from foreign systems running protocols such as IPX/SPX and TCP/IP. Additionally, NetBEUI was incapable of routing, making it useless in anything but small networks. Microsoft's solution was to dump NetBEUI, but to keep NetBIOS. This resulted in split protocols where Windows would still run NetBIOS, but instead of using NetBEUI, it would use the lower protocols the entire network used.

The first issue came about back in the days when Novell NetWare was the network operating system of choice. How do you make a bunch of Microsoft client computers running NetBIOS able to see the Novell NetWare servers running IPX/SPX?

When NetWare first became popular back in the 1980s, Microsoft simply ignored the networking issue. Novell supplied special (and free) client software that you installed on every system you wanted to see the Novell servers. When the first versions of Windows came out, Novell was right there with graphical networking-client tools. These client programs didn't use NetBIOS—they used the NetWare naming conventions.

TIP Chapter 18, "Interconnecting Network Operating Systems," discusses important details about NetWare clients.

Microsoft didn't like this one bit. Microsoft was developing the idea of the Windows Desktop and an integral part of that Desktop was Network Neighborhood, later called My Network Places. Microsoft didn't like the idea of Novell's client making its users go to a different program to access the network—Microsoft wanted everything in Network Neighborhood. By the time Windows 95 came out, Microsoft tried to beat NetWare to the punch by creating its own IPX/SPX software that could translate the NetWare Server's names into NetBIOS names the system could understand and place into Network Neighborhood. This was achieved by adding a program called Microsoft Client for NetWare. Of course, Novell quickly came out with its own client software that ran on Windows 9*x* and had the capability to make NetWare servers show up in Network Neighborhood.

This created an interesting issue. Both Microsoft and Novell now had client software that did the same thing—enable Windows computers to access NetWare servers. This created a competition throughout the 1990s and early 2000s, between Microsoft and Novell to get users to use their client software over the other. Fortunately, this competition recently came to a conclusion. Novell no longer uses IPX/SPX and has joined the TCP/IP bandwagon, making any special client software unnecessary.

TCP/IP

As usual, with greater functionality comes greater complexity. The *TCP/IP* protocol suite offers a more scaleable solution for the largest networks, but it requires significantly more configurations on the part of the network tech. TCP/IP began as a UNIX networking protocol suite, but its status as the Internet's *de facto* protocol suite has prompted both Microsoft and Novell to embrace it. Helping this along is the convenient fact that unlike NetBEUI and IPX/SPX, TCP/IP is an open standard, not controlled by any one company.

TCP/IP now stands as the network protocol of choice in the vast majority of today's networks, and certainly on any system that wants to connect to the Internet. This popularity, combined with the fairly high degree of complexity involved in making TCP/IP work properly and the fact that the Network+ exam really wants you to know TCP/IP in minute detail, requires several chapters just on the TCP/IP network protocol.

TCP/IP defines a large number of protocols that work from layers three through seven of the OSI seven layer model (Figure 10-35). In this section, we concentrate on concep tually how the TCP/IP works at multiple OSI layers and save the details for later chapters.

Applications

It's hard to talk about TCP/IP without taking time to talk about the Application layer. Unlike the proprietary protocols IPX/SPX and NetBIOS/NetBEUI, the designers of

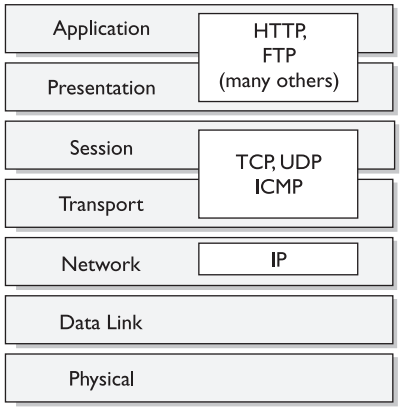


Figure 10-35
TCP/IP protocols

TCP/IP worked hard to make a protocol stack that worked well for any applications, not just file and print sharing. As a result, TCP/IP relies heavily on Application layer protocols for many critical network functions.

A great example is this DNS thing I keep referring to. DNS (as you'll see in great detail in the next two chapters) is what TCP/IP networks use to resolve a computer name like www.totalsem.com into the machine's actual IP address. This critical process runs at the Application layer.

All TCP/IP applications have their own distinct *port* number. DNS, for example, has port 53. Other famous Application layer functions have different port numbers. Your web browser uses port 80. E-mail uses ports 25 (outgoing) and 110 (incoming). TCP allows port numbers from 0 to 65,535. Port numbers from 0 to 1204 are called *well-known ports* and are used primarily by client applications to talk to server applications. No other application should use these ports. Some of the ports after 1204 are for less-known applications (and can be used by other applications), but most are for servers to use as an identifier to talk back to clients. The following line of text is cut from the `NETSTAT -n` command running on a Windows XP system. This shows a session running between my computer and the popular Google.com search engine on my web browser.

```
TCP    192.168.4.27:2357      216.239.41.104:80      ESTABLISHED
```

Ignore the words "TCP" and "ESTABLISHED" and concentrate on the IP addresses and ports. On the left is the IP address and port number used by the Google.com web server to talk to my system. On the right is the IP and port number my system uses to talk to the Google.com web server. Note that my system sends out on port 80, the well-known port used to talk to web servers. Note also that the web server uses the arbitrarily chosen port number 2357 to talk back to my system.

Earlier, we mentioned that TCP isn't one protocol, it's a family of protocols with names like TCP, UDP, or ICMP. The type of protocol is determined by the application.

TCP at Session

When you start an application on a TCP/IP computer, the application alerts the Session layer software to begin a session with a remote computer. Your system will send out a packet with the remote system's IP address and the port number of the service you are requesting. When that packet gets to the remote system, it will respond by assigning a port number for the session running from the serving system back to your system. This number is arbitrarily assigned from the pool of port numbers between 1024 and 49151.

TCP at Transport

TCP at the transport level is invisible. This layer, as already described, chops away at the outgoing data, breaking each piece into frames and adding sequence numbers. On the receiving side, the incoming packets are reassembled, checked, and then moved up to the next layer.

IP at Network

The cornerstone of TCP/IP's popularity is the robust and scalable IP addressing scheme. The IP addressing scheme, famous with its four dotted-decimal notation, has one quality unmatched in any other protocol—you can take a network, subnet it, subnet the subnets, and keep subnetting until you run out of numbers and TCP/IP can handle it! This, of course, assumes you know how to configure a complex TCP/IP network properly!

Take, for example, a block of IP addresses ranging from 10.0.0.0 up to 10.255.255.255. TCP/IP networks reserve the numbers 255 and 0 on the end of the IP address for special uses so you have all the IP addresses from 10.0.0.1 to 10.255.255.254. This would give you one network with (256 × 256 × 254 =) 16,646,144 different computers or *hosts,* as we would say in the TCP/IP vernacular. All the computers share the same first number: 10. We call that part of the IP address the network ID. The part that changes for every system is called the *host ID*. Let's pick one computer on our network: 10.168.43.7. Its network ID is 10.0.0.0 (we put zeroes on the end of network IDs) and the host ID is 168.43.7.

Here's where the power of IP shines. If we wanted to, we could take some of those IP addresses in the 10.0.0.0 network; let's pick all the addresses with the 10.14.0.0 network ID and make this a subnet of the bigger network. We could take the 10.14.0.0 network and subnet that into even smaller networks. Notice that each subnet has fewer host IDs than the larger network, so you eventually run out of numbers, but you get the idea. You'll see in Chapter 12, "Network Operating Systems," that you can subnet even more than what's shown here.

Installing TCP/IP

The first item to put into your head is that every modern OS installs TCP/IP by default, so it's rare for you to need to install TCP/IP. If, by some chance, you find yourself needing to install the TCP/IP protocol into Windows, just follow the installation examples shown in the earlier network protocols in this chapter.

The "joy" of TCP/IP comes from the configuration. TCP/IP, in its most basic form, takes a substantial amount of configuration. Fortunately, TCP/IP has evolved dramatically over the years to the point where most client systems need little or no configuration to work correctly.

Splitting Protocols II: NetBIOS over TCP/IP

It's been said that Bill Gates once called the Internet a passing fad. While I've never found a reliable reference for this quote, it is correct to say that Windows was the last major operating system to adopt TCP/IP. There was a good reason for this late entry—Windows had NetBIOS tightly embedded and NetBIOS just wasn't designed to work with TCP/IP.

The world continued to move toward TCP/IP and Microsoft, finally realizing the need for TCP/IP with Windows, responded to this issue in a series of steps. By the early 1990s, Microsoft provided a TCP/IP protocol—but you could not run a Microsoft network with the TCP/IP software provided from Microsoft. Back then, you added both the NetBEUI and the TCP/IP protocols to your Windows computers. NetBEUI handled your internal network and TCP/IP enabled you to run web browsers and such. It was inelegant, but it worked!

The next step was to take the flexibility of TCP/IP and add NetBIOS support. This was actually fairly simple, as long as you kept your networks small. Microsoft grabbed some unused port numbers and transformed NetBIOS into an Application-layer function on a TCP/IP network, using a process called *NetBIOS over TCP/IP (NetBT)*. It even invented a naming system called *Windows Internet Naming Service (WINS)* that enabled NetBIOS over TCP/IP to work over large networks. (See Chapter 11, "TCP/IP," for a discussion on WINS.) The vast majority of computers run TCP/IP and the vast majority of Windows computers run NetBIOS over TCP/IP.

The last step is still taking place: Microsoft is dropping support for NetBIOS. Networks composed exclusively of Windows 2000 and XP systems can turn off NetBIOS and still function as a network.

Also-Ran Protocols

Odds are good that you could spend your entire networking life and never see any protocols other than NetBIOS/NetBEUI, IPX/SPX, and TCP/IP. In fact, odds are good that you could go the rest of your life and only see TCP/IP! However, you need to know about a few other "also-ran" network protocols that are historically important and still supported by most operating systems.

AppleTalk

Just as IPX/SPX was invented in-house by Novell for its NetWare NOS, Apple invented the *AppleTalk* network protocol suite to run on Apple computers. AppleTalk was originally designed to run on top of the old LocalTalk networking technology. Roughly

speaking, AppleTalk does for Macs what NetBIOS does for PCs. When two Macintosh computers communicate using AppleTalk, their conversation looks quite a bit like a NetBIOS session, in that each name has an associated number function assigned to it. I won't go through this in detail, but you should know that AppleTalk uses a special function called Name Binding Protocol (NBP). NBP binds each system's name to its AppleTalk address, so that other systems on the network can see it. Modern Macintosh systems still rely on AppleTalk. All but the earliest versions of Windows come with the capability to install the AppleTalk protocol, so they can talk to Macintosh systems that still use AppleTalk. For the Network+ exam, at least know that if you want to talk to a Macintosh computer, you'll want to install the AppleTalk protocol on your Windows system.

TIP To communicate with an older Macintosh on a Windows-centric network, install the AppleTalk protocol on your Windows PCs.

DLC

The *Data Link Control (DLC)* network protocol was used for many years to link PCs to mainframe computers. Because Hewlett-Packard adopted the DLC protocol for use by network printers, DLC enjoyed a much longer life than it probably should have, given the existence of so many alternatives. All versions of Windows, including Windows XP, still support DLC, but unless you're installing an older network printer that only uses DLC, odds are good you'll never see it. Just know that DLC is a network protocol and you can install it if needed.

Chapter Review

Questions

1. What is the maximum length of a NetBIOS network name?
   - **A.** 8 characters
   - **B.** 23 characters
   - **C.** 15 characters
   - **D.** 256 characters
2. NWLink is Microsoft's version of which protocol suite?
   - **A.** NetBEUI
   - **B.** NetBIOS
   - **C.** IPX/SPX
   - **D.** TCP/IP

3. An IPX/SPX external network number is also known as
   - **A.** The subnet mask
   - **B.** An external node
   - **C.** A segment address
   - **D.** A default address
4. The NetBIOS protocol operates at which OSI layer?
   - **A.** Data Link
   - **B.** Network
   - **C.** Transport
   - **D.** Session
5. What part of the NetBEUI protocol suite operates at the Transport layer?
   - **A.** IPX
   - **B.** NetBIOS
   - **C.** NetBEUI
   - **D.** SPX
6. Travis, using a Windows 95 computer named TRACK3, complains that he cannot connect with another Windows 95 computer named SALES3. Jim, the friendly neighborhood network technician, determines that TRACK3 can connect successfully with other machines that reside on the same side of the router, but not with any machines on the far side of the router. Which of the following is the most likely cause of Travis's problem?
   - **A.** TRACK3 is connected to the network with a bad cable.
   - **B.** TRACK3's network card has failed and needs to be replaced.
   - **C.** TRACK3 is running NetBEUI.
   - **D.** TRACK3 is running NetBIOS.
7. Which of the following protocols are routable? (Select all that apply.)
   - **A.** NetBEUI
   - **B.** IPX/SPX
   - **C.** TCP/IP
   - **D.** NetBIOS
8. What do routers do with NetBEUI packets?
   - **A.** Send the packets out over the Internet.
   - **B.** Send the packets back to the sending machines.

C. Hold the packets until they receive a resend request.

D. Discard the packets.

9. The IPX/SPX protocol was originally designed for use with which network operating system?

A. Novell NetWare

B. Windows 3.1

C. UNIX

D. Linux

10. What are Ethernet 802.3, Ethernet II, Ethernet 802.2, and Ethernet SNAP?

A. Types of NIC cards

B. IPX/SPX data structures

C. TCP/IP protocols

D. OSI model layers

Answers

1. **C.** The maximum length of a NetBIOS network name is 15 characters. NetBIOS reserves the 16th byte for a special code that defines the function of the machine on the network.
2. **C.** NWLink is Microsoft's version of the IPX/SPX protocol. NetBIOS is part of the NetBEUI protocol.
3. **C.** An IPX/SPX external network address is also known as simply the network number or the segment address.
4. **D.** The NetBIOS protocol operates at the Session layer.
5. **C.** The NetBEUI protocol operates at the Transport layer. IPX and SPX are parts of a different protocol.
6. **C.** TRACK3 is running NetBEUI. NetBEUI is a nonroutable protocol. Because TRACK3 can communicate with some other machines on the network, a bad cable or NIC is unlikely to be the cause of the symptoms described.
7. **B, C.** IPX/SPX and TCP/IP are both routable protocols. NetBEUI is not a routable protocol, and NetBIOS is part of the NetBEUI protocol suite.
8. **D.** Routers discard NetBEUI packets because the packets lack the OSI Network layer routing information that routers use to do their job.
9. **A.** IPX/SPX was originally designed for use with Novell NetWare.
10. **B.** IPX running on top of Ethernet can use one of four data structures, called frame types: Ethernet 802.3, Ethernet II, Ethernet 802.2, or Ethernet SNAP.