CHAPTER 11

TCP/IP

The Network+ Certification exam expects you to know how to

- 2.5 Identify the components and structure of IP (Internet Protocol) addresses (IPv4, IPv6) and the required setting for connections across the Internet
- 2.6 Identify classful IP ranges and their subnet masks (for example: Class A, B, and C)
- 2.7 Identify the purpose of subnetting
- 2.8 Identify the differences between private and public network addressing schemes
- 2.10 Define the purpose, function, and use of the following protocols used in the TCP/IP suite: TCP, UDP, FTP, TFTP, SMTP, HTTP, POP3/IMAP4, Telnet, ICMP, ARP/RARP, NTP, NNTP, LDAP, IGMP
- 2.11 Define the function of TCP/UDP (Transmission Control Protocol/User Datagram Protocol) ports

To achieve these goals, you must be able to

- Recognize properly formatted IP addresses
- Describe the function of subnet mask and default gateway
- Define and calculate classful and classless subnets
- Describe the functions of DNS, DHCP, and WINS
- Recognize the port numbers and the functions of popular TCP, UDP, and ICMP applications
- Describe the need for IPv6 and recognize properly formatted IPv6 addresses.

TCP/IP has the unique distinction of being the only popular network protocol designed from the ground up by the government of the United States. Like many government endeavors, TCP/IP is big, messy, ponderous, hard to configure (relative to other protocols, at least), and tries to be all things to all users. Compared to IPX/SPX or NetBIOS/NetBEUI, TCP/IP is a pain in the posterior at best. So why does every computer on the Internet use TCP/IP? Why do networks not even connected to the Internet almost exclusively use TCP/IP? Why is TCP/IP overwhelmingly the most popular network protocol on the face of the Earth? The answer is *flexibility*. TCP/IP works well for both small and large networks in a way that no other network protocol matches, and this chapter's job is to show you that flexibility.

In this chapter, you'll learn the components and configurations of TCP/IP. We'll go over IP addresses in detail. Next, we'll follow with a discussion of how a single TCP/IP network can divide itself into multiple subnetworks. Then we'll take a look at some other parts of the TCP/IP suite, including the all-important TCP and UDP protocols, to see what role they play in setting up systems to run in TCP/IP networks. Finally, we'll review the functions of various key ports, and take a quick look at the upcoming IPv6 protocol.

Historical/Conceptual

IP Address Basics

The greatest danger facing those learning about IP addressing stems from the fact that so much of TCP/IP has reached the common vernacular. The average user has heard words like "IP address" and "default gateway" and might already have an idea about what they mean and why TCP/IP networks use them. Odds are also good that many of you have an idea as to what an IP address looks like. In most cases, knowledge is a good thing, but in this case, we need to get on the same mental playing field. The best way to get there is to inspect the format of a typical IP address.

Test Specific

IP Address Format

Every system in an IP network (in TCP/IP parlance, every *host*) must have a unique IP address. IP addresses look complex, but are quite simple once you know how they work.

To see a typical IP address, go to a system that connects to the Internet (so you know it uses TCP/IP) and run *WINIPCFG* (Windows 9*x*/Me/XP) or *IPCONFIG* (Windows NT/98SE/Me/2000/XP/2003) from the Start | Run menu. Both tools enable you to view network settings at a glance, making them invaluable as first-line troubleshooting tools. Figure 11-1 shows the IP address for a Windows 98 system.

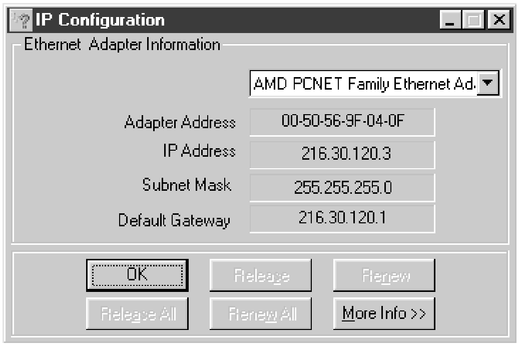

IP Configuration
Ethernet Adapter Information
AMD PCNET Family Ethernet Ad.
Adapter Address: 00-50-56-9F-04-0F
IP Address: 216.30.120.3
Subnet Mask: 255.255.255.0
Default Gateway: 216.30.120.1

Figure 11-1 WINIPCFG showing a system's IP address

An IP address consists of four numbers, each number being a value between 0 and 255. We use a period to separate the numbers. No IP address may be all 0s or all 255s (I'll explain this in a moment). Here are a few valid IP addresses:

216.30.120.3

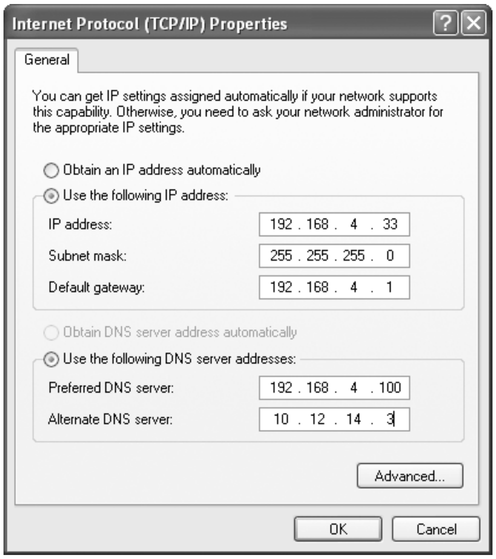
224.33.1.66

123.123.123.123

and here are a few invalid IP addresses:

216.30.120	(Must have four numbers)
255.255.255.255	(Can't be all 255s)
6.43.256.67	(Every number must be between 0 and 255)
32-1-66-54	(Must use periods between numbers)

These numbers are entered into every system on the IP network. Figure 11-2 shows where to enter these values in a Windows system: the Internet Protocol (TCP/IP) Properties dialog box.


Internet Protocol (TCP/IP) Properties

General

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

Obtain an IP address automatically [✓ Use the following IP address:]

IP address:	192 . 168 . 4 . 33
Subnet mask:	255 . 255 . 255 . 0
Default gateway:	192 . 168 . 4 . 1

Obtain DNS server address automatically [✓ Use the following DNS server addresses:]

Preferred DNS server:	192 . 168 . 4 . 100
Alternate DNS server:	10 . 12 . 14 . 3

Figure 11-2 Internet Protocol (TCP/IP) Properties dialog box showing IP addresses

Note in Figure 11-2 the value labeled Subnet mask, as well as options that say Default gateway, Preferred DNS server, and Alternate DNS server. Then there's the Advanced button that leads to even more options. What are these all about? Hey, I told you IP is by far the messiest of all protocols to configure! That's what this chapter is all about: understanding these IP entries. But to understand how to configure these settings, you must understand what an IP address is. So, let's grab an arbitrary IP address and start tearing it up into its real components: binary numbers.

Converting IP Addresses

While we commonly write IP addresses as four groups of digits separated by periods, this is just convenient shorthand for a 32-bit binary address. Here's a typical IP address in its primeval binary form:

11000101101010010101111001010010

While computers have no difficulty handling long strings of ones and zeroes, most humans can't handle speaking in these terms. Instead, techs use a special shorthand for IP addresses called dotted decimal notation. Dotted decimal notation works as follows. First, divide the IP address into four pieces of eight bits:

11000101 10101001 01011110 01010010

A group of eight bits has a limited number of permutations of ones and zeroes, ranging from all zeroes (00000000) to all ones (11111111), with a lot of different combinations of ones and zeroes in between. The exact number of permutations is 2^8 or 256 different patterns of ones and zeroes. Each group of eight ones and zeroes corresponds to a number between 0 and 255. If you write down all of the possible permutations for eight bits, the list starts like this:

00000000 = 0

00000001 = 1

00000010 = 2

00000011 = 3

00000100 = 4

I'll skip about 246 entries here and show the end of the list:

11111011 = 251

11111100 = 252

11111101 = 253

11111110 = 254

11111111 = 255

Each value of 0 to 255 to represent eight bits is called an *octet*. Wait a minute! Isn't eight binary characters a byte? Then why not call them bytes? Well, they most certainly could be bytes, but the difference here is the numbering system. Normally when we discuss binary values, we use hexadecimal, but the TCP/IP folks wanted something easier than hexadecimal, so they instead use this 0-to-255 numbering system called octets. Any binary octet can be represented by the values 0 to 255. Let's take a look at the first example I just gave you:

11000101 10101001 01011110 01010010

We represent this number in dotted decimal notation as 197.169.94.82. While you may have some idea how I did this, let's take a moment to work through how to convert a true binary IP address into dotted decimal, and back again into binary. The trick is to convert the individual octets, one at a time.

The problem with using decimal notation to display an octet is that most folks find it difficult to convert between binary to decimal notation, at least without a calculator handy. The secret to success is to appreciate that each position in the binary value corresponds to a decimal value. Look at this list:

00000001 = 1

00000010 = 2

00000100 = 4

00001000 = 8

00010000 = 16

00100000 = 32

01000000 = 64

10000000 = 128

If you can memorize these eight binary/decimal equivalents, you can convert any binary octet into its decimal equivalent—not only useful in your networking duties, but an excellent trick you can use to amaze your friends at parties! To make conversion easier, every good IP network person memorizes the form in Figure 11-3.

Figure 11-3
Form to help with binary/decimal conversion

Position	8th	7th	6th	5th	4th	3rd	2nd	1st
Value	128	64	32	16	8	4	2	1

To convert an octet from binary to decimal, enter the binary number into the empty spaces at the bottom of the form, copy the decimal values of the columns where there are ones in the binary number, and add those decimal values together to get a final answer. Let's convert the binary value 10010011 to a decimal number (see Figure 11-4). In the 8th position enter a 1, and write down 128, which is the decimal value in that column. 128 is the first number in your decimal addition problem. In the 7th and 6th positions enter 0's, and for completeness, write down zeroes. Your decimal addition problem now reads 128 + 0 + 0. Enter a 1 in the 5th position, and write down 16, the decimal value of that column. In the 4th and 3rd positions, enter two more 0's, and write those down. In the 2nd position enter a 1, and add its value, 2, to your equation that follows. Finally, place a 1 in the 1st position, which we know equals 1. Add a 1 to the end of your addition problem, and then do the math! The decimal equivalent of 10010011 is 147. Converting binary numbers to decimal is easy with this handy form!

Figure 11-4 Converting the binary value of 10010011 into a decimal value

Position	8th	7th	6th	5th	4th	3rd	2nd	1st
Value	128	64	32	16	8	4	2	1
	1	0	0	1	0	0	1	1

128 + 0 + 0 + 16 + 0 + 0 + 2 + 1 = 147

Now let's return to our binary IP address example, and convert each binary octet to decimal:

11000101=197 10101001=169 01011110=94 01010010=82

Then separate the values with dots, like this:

197.169.94.82

You've now translated an IP address from binary to the more familiar dotted-decimal IP address format! Go drink the beverage of your choice in celebration—but only one, because you'll need your wits about you for the next step: converting a decimal value into binary!

You can use the same form to convert numbers from dotted decimal to binary. Let's use a sample decimal value of 49. Whip out a nice blank form and write the number 49 at the top. Start on the far left-hand side of the form and ask yourself, "How many 128s are there in 49?" You should answer yourself, "None, because 128 is larger than 49!" Because zero 128s fit into 49, place a zero in the 128 spot. Repeat this exercise with the next value, 64. Each time you hit a value that produces an answer of one, place a one in the corresponding position. Subtract that value from the decimal value and continue the procedure with the remainder.

Here's how it works with the 49 example:

- How many 128s in 49? None—so put a zero in the 128s place.

- How many 64s in 49? None—so put a zero in the 64s place.
- How many 32s in 49? One—put a one in the 32s place and subtract 32 from 49, leaving 17.
- How many 16s in 17? One—put a one in the 16s place and subtract 16 from 17, leaving 1.
- How many 8s in 1? None—so put a zero.
- How many 4s in 1? None—so put a zero.
- How many 2s in 1? None—so put a zero.
- How many 1s in 1? One—put a one in the 1s place and subtract 1 from 1, leaving zero.

This does—and because you've reached the end of the procedure, it most definitely should—leave you with zero (Figure 11-5). If you don't get zero at this point, you made a mistake. No big deal—it can take some practice to get the hang of it. Try it again!

Figure 11-5 Did yours come out to zero?

Position	8th	7th	6th	5th	4th	3rd	2nd	1st
Value	128	64	32	16	8	4	2	1
	0	0	1	1	0	0	0	1

How many	128s	in	49?	0	
How many	64s	in	49?	0	
How many	32s	in	49?	1	49 - 32 =17
How many	16s	in	17?	1	17 - 16 =1
How many	8s	in	1?	0	
How many	4s	in	1?	0	
How many	2s	in	1?	0	
How many	1s	in	1?	1	1 - 1 =0

Using this handy form, we've deduced that the decimal value 49 is 00110001 in binary. Because every IP address has four values, you must go through this process three more times to convert a typical IP address. You'll find that, with practice, this becomes a fast process. Stop here; make up some IP addresses, and practice converting back and forth with values you create until you're comfortable.

Now that you've toiled to learn conversion by hand, I have a confession to make: you can use the Calculator applet that comes with every version of Windows (Start | Run, type **calc**, and press ENTER). The Scientific mode of the calculator has a nice series of radio buttons on the top left. Make sure that the Decimal radio button is checked, type in the octet's decimal value, and then click the Binary radio button. Voilà! Instant conversion!

NOTE The Network+ exam does not expect you to know how to convert dotted-decimal IP addresses into binary and back, although any network tech worth his salt can do this in his sleep. Also, it's a skill you need to understand a number of other aspects of TCP/IP that you are about to learn.

I always laugh when I think about the first time I tried converting IP addresses from decimal to binary and back. When I first had to learn all this, it took me three solid days to get the whole thing straight in my head, while my friend Taylor had it nailed in about five minutes. Even though a calculator will do the conversions for you, don't give up too quickly on doing it yourself—you'll find that being able to do this manually brings some big benefits out in the real world!

Local vs. Remote

Back in the early 1970s, the world was waking up to the possibility of taking computing beyond individual systems. People began realizing the dream of a vast network of systems across the United States and, eventually, the world—what we now call the Internet. The U.S. Defense Advanced Research Projects Agency (DARPA) was tasked to create a series of protocols to support this massive network. This network needed a protocol, so DARPA had to get to work designing one—but not just any network protocol! The *D* in DARPA stood for "Defense." DARPA needed a protocol that could handle a crisis—in particular, the loss of one or more network interconnections from an atomic bomb hit (remember, this was during the Cold War). DARPA wanted a large network designed such that, in the event one piece suddenly went down, the remaining parts of the network would realize the break and automatically reroute network traffic to avoid the downed areas.

Thus came the birth of TCP/IP. TCP/IP was designed from the ground up to support groups of separate, interconnected networks. Each of the individual networks connected to one or more individual networks via special computers called routers. Routers direct network traffic by reading the IP addresses in each incoming packet, and then use that information to send the packet out on its way toward the intended recipient. (The creators of the TCP/IP protocol suite literally invented the concept of routers.)

In general, a single TCP/IP network consists of some number of computers connected together into a single collision domain. If this TCP/IP network wants to interconnect to other TCP/IP networks, it also needs a router connected to the same network, as shown in Figure 11-6.

If you're like most people, you look at Figure 11-6 and say, "What's with that cloud? Is that the Internet?" Let's get some misconceptions out of the way right now. The terms "Internet" and "TCP/IP" are so closely interwoven that most folks assume that all TCP/IP networks are part of the Internet. Not true! First, a TCP/IP network doesn't have to connect to any other TCP/IP network. Nothing prevents you from taking a few Windows computers, installing TCP/IP on them, and connecting them to a single switch. This little TCP/IP network will run perfectly and it's done all the time. Second, you can connect two or more TCP/IP networks together and not be on the Internet. For example, you can take two TCP/IP networks in the same room and connect them together with a router. These two interconnected networks use TCP/IP and a router, but they're not part of the Internet. The cloud is a representation of "the rest of the TCP/IP network." The cloud might represent one other network, a number of other networks, or even the Internet—it makes no difference.

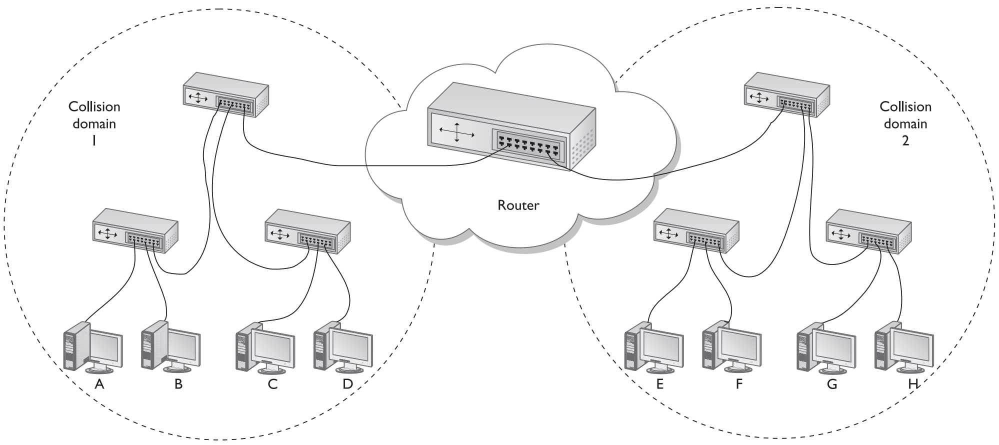


Figure 11-6 Typical TCP/IP network

TIP Just because a computer runs TCP/IP, don't assume it's connected to the Internet!

If a single TCP/IP network connects to a group of interconnected networks, would it be safe to assume that some of the traffic going in and out of any one computer stays *local*—within its own network? Equally, won't all of the traffic destined for *remote* computers (outside the local network) need to go out through the router? This local vs. remote issue is the cornerstone of TCP/IP configuration: there must be a method to enable a system to know when it needs to send data to its local network and when it needs to send data to the router.

TCP/IP most certainly does have a method—an elegant one! To learn this method, let's begin by looking at a simple TCP/IP network, a network not connected to any other network, and see how TCP/IP works to send data from one system to another.

ARP

With all this talk about IP addresses, it's easy to forget that in a simple (no routers) TCP/IP network, all of the systems share the same network technology. In this case, the computers must put their packets inside of frames to move data from one system to another, and frames use MAC addresses—not IP addresses—to do that job.. So how does the sending system know the MAC address of the receiving system? The TCP/IP protocol suite has a neat little function called *address resolution protocol (ARP)* that performs this task.

To see how ARP works, let's add TCP/IP to Hannah's MHTechEd network you saw in the previous chapter. In this case, she configured DANA to use the IP address 216.30.120.2 and JANELLE gets the IP address 216.30.120.3. You also know that each system has a MAC address—let's give each system a MAC address, as shown in Figure 11-7.

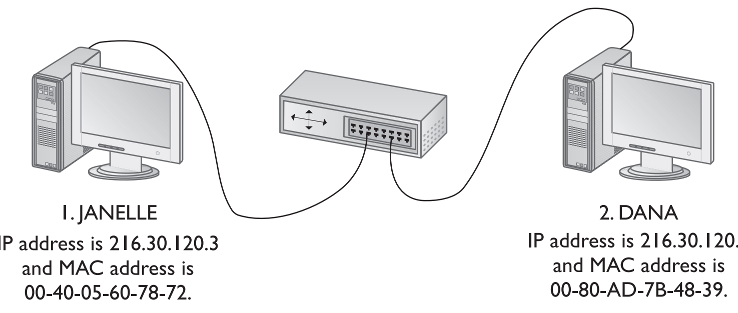


Figure 11-7 MHTechEd network showing IP and MAC addresses for each system

Let's say DANA wants to send data to JANELLE. It knows the IP address (216.30.120.3) but does not know the MAC address, and without the MAC address, it cannot make a frame. So, DANA sends out a special frame, addressed to a special MAC address called the broadcast address. This MAC address is all 1s (FFFFFFFFFFFF in hexadecimal). It's special because all systems on the network receive and process frames sent to the broadcast address. This frame asks every system on the local network: "What is the MAC address for IP address 216.30.120.3?" We call this frame an *ARP request*. The system with that IP address replies to DANA with an ARP reply (see Figure 11-8).

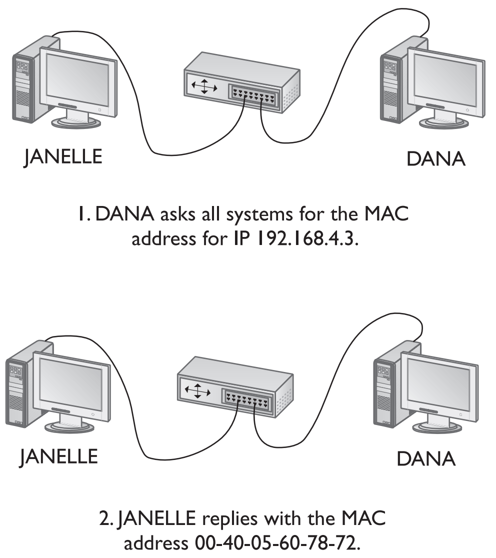
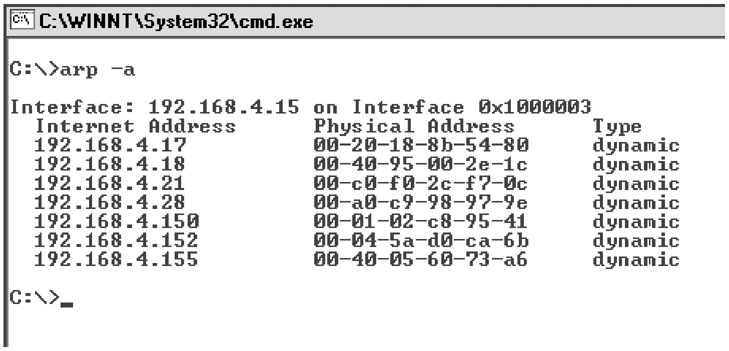

JANELLE
DANA
1. DANA asks all systems for the MAC address for IP 192.168.4.3.
JANELLE
DANA
2. JANELLE replies with the MAC address 00-40-05-60-78-72.

Figure 11-8 DANA sending an ARP request and getting an ARP reply

Once DANA gets the MAC information for JANELLE, it stores this in a cache. You can see the ARP cache in your system by typing the command **ARP -a** (note: the **-a** is case-sensitive) from a command prompt in any version of Windows. Figure 11-9 shows the **ARP** command in action!

In some situations, a computer knows another computer's MAC address, but needs an IP address—the exact opposite of an ARP. To get a MAC address, the system broad-



```
C:\>arp -a

Interface: 192.168.4.15 on Interface 0x1000003
  Internet Address      Physical Address      Type
  192.168.4.17          00-20-18-8b-54-80     dynamic
  192.168.4.18          00-40-95-00-2e-1c     dynamic
  192.168.4.21          00-c0-f0-2c-f7-0c     dynamic
  192.168.4.28          00-a0-c9-98-97-9e     dynamic
  192.168.4.150         00-01-02-c8-95-41     dynamic
  192.168.4.152         00-04-5a-d0-ca-6b     dynamic
  192.168.4.155         00-40-05-60-73-a6     dynamic

C:\>_
```

Figure 11-9 The ARP command in action

casts a RARP (Reverse ARP) command. While ARP is very common, RARP is rare in LAN environments as only a few applications need RARP.

ARP works perfectly well for a simple TCP/IP network, but what if you want to send data from a computer to another computer on a network connected to your network with a router? Routers cannot forward ARP requests because ARP requests are broadcasts. ARP alone fails the moment you want to send data outside your local network. This is where a special setting called a gateway comes into play.

Gateways

ARP works great when one IP system needs to know the MAC address of another IP system in the same local network, but remember that one of the cornerstones of IP is its assumption that your local network will be connected to a larger network. TCP/IP assumes—and this is important—that a system knows, or at least can find out on the fly (see the DNS discussion that follows), the IP address of any system on the Internet. It is not possible, however, for one system to know the MAC addresses of all the millions of other systems on the Internet. If a system wants to send data to another system on another network, a local network must connect to a larger network via a router. The router connecting a local network must know how to address packets for other systems that are not part of its local network—we call this router the *default gateway*, *gateway router*, or sometimes just *gateway*.

NOTE The term "gateway" is used in a number of different, often unrelated ways in the networking world. In this case, I refer to a router that connects a local network to a larger network.

A gateway might be a router or it might be a PC that runs routing software. Most versions of Windows, as well as other operating systems like Linux, also have the capability to make a regular PC function perfectly well as a router. Whether your router is a special box or a PC, the gateway must have at least two network connections—one connecting to the local network and a second connecting to another network. Figure 11-10 shows

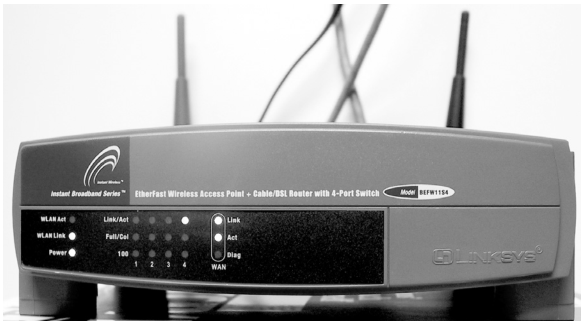
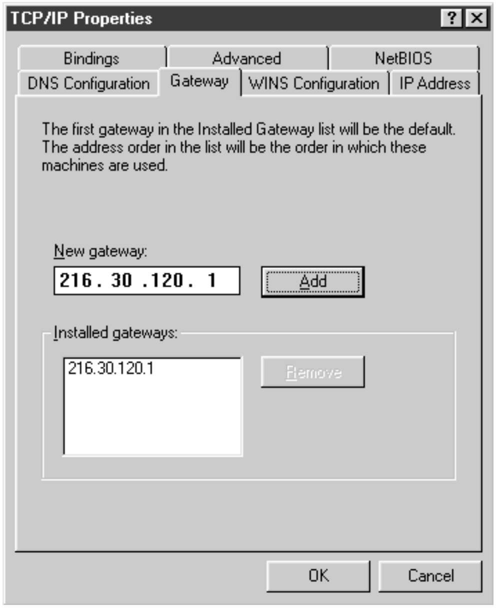


Figure 11-10 The Totalsem gateway

the gateway system in my office. This little box connects to (a) my network and (b) the DSL line to the Internet. Note the two thicker network cables—one runs to the switch for my network and the other runs to my DSL line. (The third, thinner cable is the power cable. The other two projections are antennae for its wireless capabilities.)

When setting up a local network, you must enter the IP address of the gateway in the TCP/IP Properties dialog box of every system. The IP address of the gateway is called, somewhat confusingly, the default gateway. Figure 11-11 shows my TCP/IP Properties dialog box with the default gateway setting added.



The first gateway in the Installed Gateway list will be the default. The address order in the list will be the order in which these machines are used.

New gateway: 216. 30 .120. 1

Installed gateways: 216.30.120.1

Figure 11-11 The TCP/IP Properties dialog box showing the default gateway setting

Continuing with our MHTechEd network, if DANA wants to talk to another system that is not part of the local network, it cannot ARP that far away system, because the Internet does not allow any form of broadcast frames. My gosh, imagine if it did! That would mean DANA would have to ARP every other computer on the Internet! Now imagine every system on the Internet doing this—there'd be no bandwidth left for data! Don't worry about this happening; every router on the Internet is designed to block ARP requests. Instead, the DANA system sends its data to the local network's default gateway, which will work with all of the other routers on the Internet to get the packet to the proper location.

NOTE Be careful at this point! You might think that remote data heading out of the local network doesn't need ARP. It does—but only to get data to the default gateway.

Okay, let's review. Any system that wants to send a data packet over a network must be able to tell whether the recipient system is local or remote, because it handles locally and remotely bound IP packets in two completely different ways. If the address is local, the sending system can use ARP. If the address is on a remote network, it creates packets with the remote system's IP address and runs an ARP to determine the MAC address of the default gateway. Armed with the default gateway's MAC address, the sending system tells its NIC to make frames with the gateway's MAC address and sends frames to the default gateway. As each frame comes into the default gateway, it strips off the frame, leaving the IP packets (which still have the IP address of the remote system as its destination). The default gateway then inspects the IP packets, wraps them up in whatever type of frame the outgoing connection needs, and sends them toward the intended system.

Well, the local vs. remote question is half-answered. You now know what your system will do if it determines a packet is local or remote, but now it's time for the other half of the big question: How does it know if the packet is local or remote?

Subnet Masks and Subnetting

Every system on a TCP/IP network needs a special binary value called a *subnet mask* to enable them to distinguish between local and remote IP addresses. But before we can dive into subnet masks, you need to understand how TCP/IP can organize networks into distinct chunks. Think about it, we've got all these IP addresses, from 0.0.0.0 all the way up to 255.255.255.255. That's over 4 billion IP addresses! Clearly it's not optimal to treat them all as one big network—you couldn't even daisy-chain enough switches together to do it! We need an organizational scheme to divvy up this huge pool of IP addresses into groups of IP addresses that make sense and that we can use. And lucky us, this has been done! The official term for these groups of addresses is *network IDs*.

Network IDs

In the earlier discussion, Hannah configured DANA to use the IP address 216.30.120.2 and JANELLE to use the IP address 216.30.120.3. Notice anything about these two IP addresses? Look carefully. They are almost identical—only the final octet is different, and that is not a coincidence! It has to do with the organization of networks. Consider postal addresses for a moment. Why is your address 305 Main Street, and your next-door neighbor's 306 (or 307, depending on how your street is numbered) Main Street? How about if we make your address 1313 Mulberry Lane, and your neighbor's address 451 Bradbury Avenue? Imagine the horror on the face of your mail carrier! For the same reason we group street addresses using street names and sets of consecutive numbers, we divide network addresses into two parts: one part designating a group of computers, and one part designating individual computers. Let's watch this happen. Here's an IP address in binary form: 11010101101010010101111001010010.

Which part is which? Ha! Not easy, is it? It's easier than it looks. (The answer to this lies in the *subnet mask*, but you need a bit more background for this to make sense.)

Look at the DANA and JANELLE IP addresses again. The first part, 216.30.120, is the same for each system. In fact, it's the same for every machine in the MHTechEd network! This part of the address, the part that's the same for all the MHTechEd systems, is called the *network ID*. It's the number by which the rest of the Internet knows the MHTechEd network. The last part of the IP address, the part that is—and must be—different for each host system on the MHTechEd network, is called—catch this originality—the *host ID*. So how many hosts can the MHTechEd network support, given that each one must have a unique IP address? Remember the binary conversion stuff we just did (I hope)? Each binary octet translates into a decimal number between 0 and 255, for a total of 256 possible IP addresses using MHTechEd's network ID, starting with 216.30.120.0 and running through 216.30.120.255.

Great. Only there's a small complication, courtesy of those quirky folks who designed the Internet: no host ID can be either all zeroes or all ones. So 216.30.120.0 and 216.30.120.255 are invalid IP addresses. The rest of them are okay, however, so the range of IP addresses available to the MHTechEd network starts at 216.30.120.1 and runs to 216.30.120.254, for a whopping total of 254 unique IP addresses.

If some other network admin asks the MHTechEd network admin for his network ID, the answer will be 216.30.120.0. This is just a convention among network folks—any time you see an IP address with a final octet of zero, you are looking at a network ID, not the address of any individual system.

TIP Whenever you see an IP address that ends with zeroes, it is a network ID.

Okay, you're thinking, so the first three octets are the network ID, and the last one is the host ID, right? In MHTechEd's case that's true, but that's not always the case! Here are two more IP addresses, for two systems that share the same network ID: 202.43.169.55 and 202.43.67.123. Uh oh, this time only the first two octets are the same! Care to guess what the network ID is for this network? It's the first two octets: 202.43. Why would one network have a three-octet network ID, and the other only two? Because the one with the two-octet network ID has many more computers on its local network and needs a lot more IP addresses for its host systems. The two-octet network's IP addresses range from

202.43.0.1 to 202.43.255.254. Now, look carefully—what part of 202.43.0.1 is the host ID? The 1? Nope, it's the final two octets: 0.1!

Oh no, that host ID has a zero! A host ID can't have all zeroes or all ones, right? Granted, the first octet is 00000000 (all zeroes), but the second octet is 00000001 (not all zeroes). The IP address 202.43.0.0 won't fly, but 202.43.0.1 works just fine. By the same token, we can't use 202.43.255.255, because the final two octets would be 11111111 and 11111111 (all ones), but we can use 202.43.255.254, because its final two octets are 11111111 and 11111110 (not all ones). I never said that an octet in an IP address can't be all zeroes or all ones; what I said was that a host ID can't be all zeroes or all ones. Big difference!

We've established that MHTechEd has a network ID of 216.30.120.0. Where did it come from, and why is it a different size from the other one we just looked at? Because no two systems anywhere in the world on the Internet can have the same IP address, we need a single body to dispense IP addresses. The *Internet Assigned Numbers Authority (IANA)* is the ultimate source of all network IDs. You can't just call them up and ask for one, however; only the big boys play in that arena. Most small networks get their network ID assignments from their Internet service provider (ISP); the ISPs and some large end users can go directly to one of the IANA-authorized Regional Internet Registries that collectively provide IP registration services to all regions around the globe. Okay, so MHTechEd probably got its network ID from its ISP, but who decided what size it would be? MHTechEd's network admin did, by deciding how many individual IP addresses MHTechEd needed. MHTechEd is a small company, so it only asked its ISP for 254 IP addresses, and the ISP gave MHTechEd a three-octet network ID.

Subnet Mask

Now that you know where network IDs come from, let's return to the second part of the big question: how does a host system tell whether an IP address is local or remote? It compares the IP address to its own IP address to see if they have the same network ID. All machines with the same network ID are, by definition, on the same network, so if the IDs match, it knows the other system is local, not remote. Every TCP/IP computer uses the *subnet mask* to compare network IDs.

Subnet masks are always some number of ones, followed by enough zeroes to make a total of 32 bits. This is—not coincidentally—the same length as an IP address. Every network has a subnet mask, determined by the length of its network ID. Your system uses the subnet mask like a filter. Everywhere there is a 1 in the subnet mask, you are looking at part of the network ID (see Figure 11-12). Everywhere there is a zero, you are looking at part of the host ID. By placing a subnet mask on top of an IP address, a computer can tell which part of the IP address is the network ID and which part is the host ID. A sending system holds up its local network's subnet mask to both its own and the recipient's IP addresses, to see whether the part of the address under the 1's (the network ID) is the same for both systems. If the other system shares the sending system's network ID, it's local; if it does not, it's remote (see Figure 11-13). This process is technically called *ANDing*.

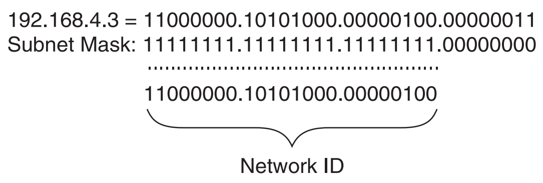


Figure 11-12 The part of the IP address under the 1's is the network ID

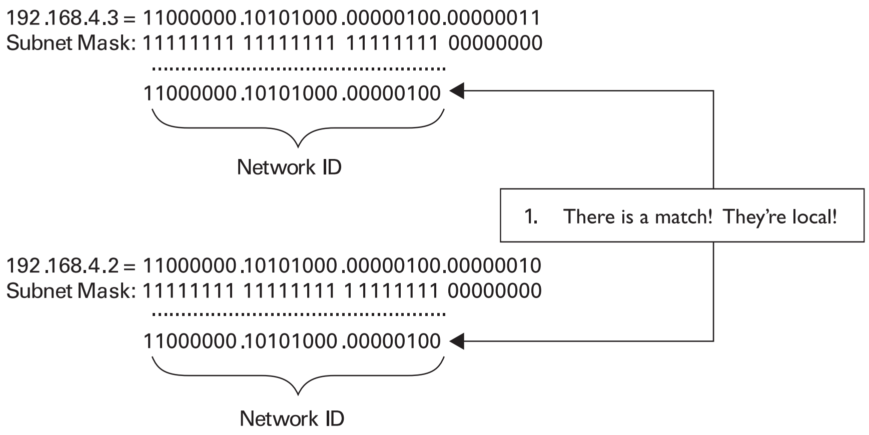


Figure 11-13 Comparing two network IDs using the subnet mask

Subnet masks are represented in dotted decimal just like IP addresses—just remember that both are really 32-bit binary numbers. All of the following (shown in both binary and dotted decimal formats) can be subnet masks:

11111111111111111111111100000000 = 255.255.255.0

11111111111111110000000000000000 = 255.255.0.0

11111111000000000000000000000000 = 255.0.0.0

Most network folks represent subnet masks using special shorthand: a / character followed by a number equal to the number of ones in the subnet mask. Here are a few examples:

11111111111111111111111100000000 = /24 (24 ones)

11111111111111110000000000000000 = /16 (16 ones)

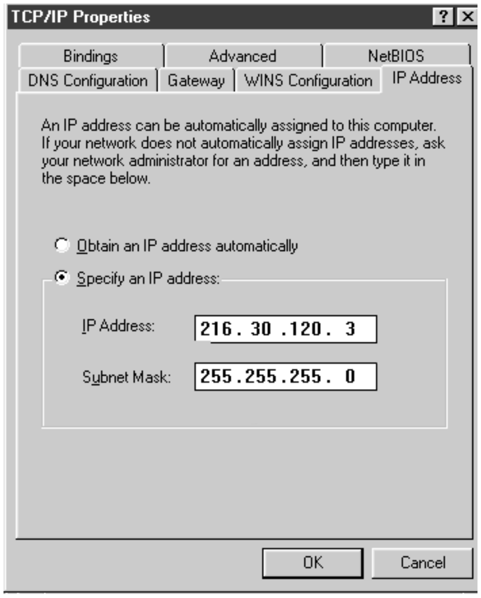
11111111000000000000000000000000 = /8 (8 ones)

An IP address followed by the / and number is telling you the IP address and the subnet mask in one statement. For example, 201.23.45.123/24 is an IP address of 201.23.45.123, with a subnet mask of 255.255.255.0. Similarly, 184.222.4.36/16 is an IP address of 184.222.4.36, with a subnet mask of 255.255.0.0.

Fortunately, computers do all of this subnet filtering automatically. Network administrators need only to enter the correct IP address and subnet mask when they first set up their systems, and the rest happens without any human intervention.

TIP By definition, all computers on the same Network will have the same subnet mask and network ID.

Let's take another peek at the TCP/IP Properties dialog box on my system to see the subnet mask (see Figure 11-14). Are you getting the impression that this dialog box is an important place for configuring your IP settings? You bet it is!



An IP address can be automatically assigned to this computer. If your network does not automatically assign IP addresses, ask your network administrator for an address, and then type it in the space below.

( ) Obtain an IP address automatically
(●) Specify an IP address:

IP Address: 216 . 30 .120 . 3

Subnet Mask: 255 .255 .255 . 0

Figure 11-14 The TCP/IP Properties dialog box showing the subnet mask

Class Licenses

Until fairly recently, if you wanted to put some computers on the Internet, you (or your ISP) asked for a block of unused IP addresses from the IANA. The IANA passes out IP addresses in contiguous chunks called *class licenses* (see Figure 11-15). Classes *D* and *E* addresses were not distributed (only Classes *A*, *B*, and *C* were).

	First Decimal Value	Addresses	Hosts per Network
Class A	1–126	1.0.0.0–126.0.0.0	16.7 Million
Class B	128–191	128.0.0.0–191.255.0.0	65534
Class C	192–223	192.0.0.0–223.255.255.0	254
Class D	224–239	224.0.0.0–239.255.255.255	Multicast addresses
Class E	240–255	240.0.0.0–255.255.255.254	Experimental addresses

Figure 11-15 IP classes

TIP Make sure you memorize the Class A, B, and C IP class licenses! You should be able to look at any IP address and tell its class license. A trick to help: The first binary octet of a Class A address always begins with a 0 (0*xxxxxxx*); for Class *B*, it's 10 (10*xxxxxx*), and for Class *C*, 110 (110*xxxxx*).

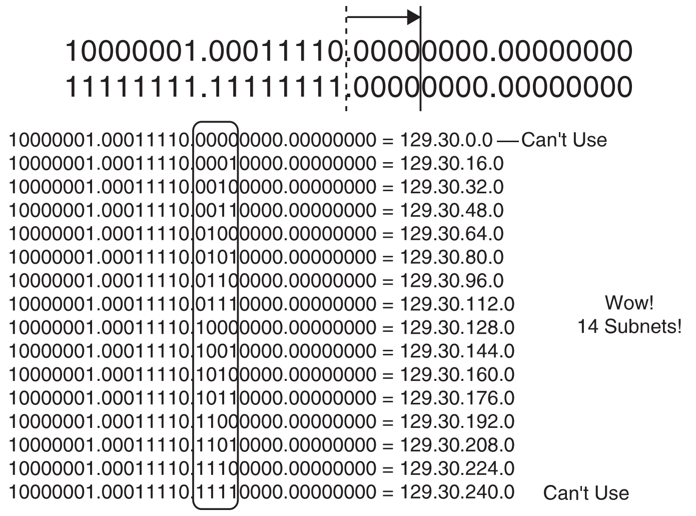
In most cases, you get an address class that you need to subnet, which means the subnet always hits at one of the dots (/8, /16, or /24). Using this, you can make a chart showing the number of subnets created by moving the subnet mask from zero to eight places (see Figure 11-16). It makes subnetting a lot faster! Note that there are only certain numbers of subnets. If you need 22 subnets, for example, you need to move the subnet mask over 5 places to get 30. You then only use 22 of the 30 subnets.

Figure 11-16 Subnet possibilities

.00000000 =	0 Subnets	2^8 -2 = 254 Hosts
.10000000 = .128 = 2^1 -2 =	0 Subnets	2^7 -2 = 126 Hosts
.11000000 = .192 = 2^2 -2 =	2 Subnets	2^6 -2 = 62 Hosts
.11100000 = .224 = 2^3 -2 =	6 Subnets	2^5 -2 = 30 Hosts
.11110000 = .240 = 2^4 -2 =	14 Subnets	2^4 -2 = 14 Hosts
.11111000 = .248 = 2^5 -2 =	30 Subnets	2^3 -2 = 6 Hosts
.11111100 = .252 = 2^6 -2 =	62 Subnets	2^2 -2 = 2 Hosts
.11111110 = .254 = 2^7 -2 =	126 Subnets	2^1 -2 = 0 Hosts
.11111111 =	0 Subnets	

The previous example used a Class C license, but subnetting works the same for Class A and Class B licenses, just with a larger number of hosts and subnets. Let's say you get a network ID of 129.30.0.0 and need to create 12 subnets. All IP addresses starting with 128.0.xxx.xxx up to 191.255.xxx.xxx are by definition Class *B*, so we can represent our network as 129.30.0.0/16. Just as you did before, first write out the starting subnet mask and start moving it to the right until you have enough subnets. If you move it over three places, you get $2^3 = 8 - 2$ (the two you can't use) = 6 subnets. Not enough. If you move the subnet mask four places, you get $2^4 = 16 - 2$ (again, minus the two you can't use) = 14 subnets (see Figure 11-17).

Figure 11-17
Moving the subnet mask four places



```
10000001.00011110.00000000.00000000
11111111.11111111.00000000.00000000

10000001.00011110.00000000.00000000 = 129.30.0.0 — Can't Use
10000001.00011110.00010000.00000000 = 129.30.16.0
10000001.00011110.00100000.00000000 = 129.30.32.0
10000001.00011110.00110000.00000000 = 129.30.48.0
10000001.00011110.01000000.00000000 = 129.30.64.0
10000001.00011110.01010000.00000000 = 129.30.80.0
10000001.00011110.01100000.00000000 = 129.30.96.0
10000001.00011110.01110000.00000000 = 129.30.112.0        Wow!
10000001.00011110.10000000.00000000 = 129.30.128.0     14 Subnets!
10000001.00011110.10010000.00000000 = 129.30.144.0
10000001.00011110.10100000.00000000 = 129.30.160.0
10000001.00011110.10110000.00000000 = 129.30.176.0
10000001.00011110.11000000.00000000 = 129.30.192.0
10000001.00011110.11010000.00000000 = 129.30.208.0
10000001.00011110.11100000.00000000 = 129.30.224.0
10000001.00011110.11110000.00000000 = 129.30.240.0     Can't Use
```

Just out of curiosity, can you figure out how many host numbers you get for each new network ID? As you can see from the previous figure, the host ID has a total of 12 places. For each network ID, you get $2^{12} - 2$ or 4094 hosts. Figure 11-18 lists some of the IP addresses for one of the new network IDs.

Figure 11-18
Some of the IP addresses for network ID 129.30.0.0/16

```
10000001.00011110.10010000.00000001 = 129.30.144.1
10000001.00011110.10010000.00000010 = 129.30.144.2
10000001.00011110.10010000.00000011 = 129.30.144.3
10000001.00011110.10010000.00000100 = 129.30.144.4
10000001.00011110.10010000.00000101 = 129.30.144.5
.......
10000001.00011110.10010000.00011110 = 129.30.144.30
10000001.00011110.10010000.00011111 = 129.30.144.31
10000001.00011110.10010000.00100000 = 129.30.144.32
10000001.00011110.10010000.00100001 = 129.30.144.33
10000001.00011110.10010000.00100010 = 129.30.144.34
.......
10000001.00011110.10010000.11111010 = 129.30.144.250
10000001.00011110.10010000.11111011 = 129.30.144.251
10000001.00011110.10010000.11111100 = 129.30.144.252
10000001.00011110.10010000.11111101 = 129.30.144.253
10000001.00011110.10010000.11111110 = 129.30.144.254
```

Subnetting has three secrets. First, remember to start with the given subnet mask and move it to the right until you have the number of subnets you need. Second, forget the dots. Never try to subnet without first converting to binary. Too many techs are locked into what I call Class-C-itis. They are so used to working only with Class *C* licenses that they forget there's more to subnetting than just the last octet. Third, practice subnetting. Use the questions at the end of this lesson as a guide. You should be able to create your own scenarios for subnetting. Stick to these three secrets and you'll soon find subnetting a breeze to do.

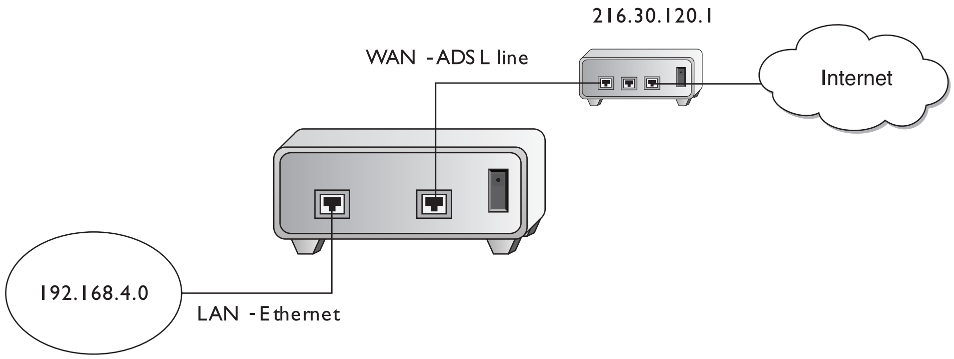
Now that you understand subnets, let's take a moment to understand why you can't use all zeroes or all ones in an IP address. First of all, you can use zeroes and ones—just not all zeroes and ones for the host ID. Let's look at a network of 201.44.13/24. The IP address of 201.44.13.0 is the network ID and is used by systems to determine how to ship a packet, so we can't use it. 201.44.13.255 is the IP broadcast address. This address is used by many different TCP/IP applications to broadcast packets to every other computer on the local network. This IP broadcast is similar to the ARP broadcast you saw at the beginning of this chapter. In an ARP broadcast, however, the destination IP address is the IP address of the computer the ARP is trying to locate. In an IP broadcast, in contrast, the destination IP address is the IP broadcast address itself.

[Mike switches to Evil Overlord mode] Cackle! Muhawhawhaw! Now that I've had some fun torturing your pathetic little minds with subnets, I'm going to let you in on a little secret. Few network administrators deal with subnetting anymore, because most systems nowadays don't need their own unique IP addresses. Instead, they use special IP addresses that are invisible outside of their local network, thanks to the machinations of special systems called NATs (I cover NATs in Chapter 15, "TCP/IP and the Internet"). Is that whimpering I hear? How satisfying! All that pain for nothing! Well, the folks who administer any network with routers, including the thousands of techs who work at every ISP, do this stuff every day. So, learn subnetting—and come visit me in my underground volcano complex when you've figured it out! Muhawhawhaw!!!

Classless Subnetting

At first glance, you may discount the network ID as relatively unimportant; most of us have never had to enter a network ID into anything. We enter the IP address, the gateway, and the subnet mask on each PC, but there's no place to type in the network ID. Nevertheless, the network ID is critical to making an IP network function. Individual systems don't use network IDs, but routers need them badly! Why do the routers need network IDs? Because routers use router tables, and router tables use network IDs. Let me explain. Figure 11-19 shows a typical router setup at a small office. I even added the IP addresses for each NIC in the router. (Each NIC in a router, by the way, is commonly referred to as an *interface*.) Below the diagram is the router table for that router. The router uses this table to determine the interface (the far-right column) through which it should route packets, based on the network ID of the recipient (the left-most column). As you can see, it needs the network ID of the receiving system to know how to route the packet. What if a packet's network ID doesn't match any of the routes in the table? Ah, those clever network designers are way ahead of you! If a router doesn't know what to do with a packet, it sends it along to its own default gateway, which will then distribute it to other routers, one of which will eventually know exactly where it should be routed. Very clever, and quite efficient, if you think about it for a minute!

This data in the router table doesn't just appear magically in the router. Someone needs to enter this information into the router when the router is first installed. But once this data is entered, the router will get the packets to the network.



Routing Table Entry List

Destination LAN IP	Subnet Mask	Default Gateway	Hop Count	Interface
0.0.0.0	0.0.0.0	216.30.120.1	1	WAN
192.168.4.0	255.255.255.0	0.0.0.0	1	LAN
216.30.120.0	255.255.255.0	0.0.0.0	1	WAN

Figure 11-19 Router setup and corresponding router table

Now that you appreciate the importance of network IDs, let's complicate the networking situation. Let's use a more advanced router with three NICs to create two separate networks (see Figure 11-20). This type of setup is commonly used to reduce network traffic or increase security. We need to make all of this work!

Let's also assume for a moment that you are in charge of setting up this network. You have been given a total of 256 IP addresses: 216.30.120.0/24. This means you are in control of all the IP addresses from 216.30.120.0 to 216.30.120.255. If this were only one network, you wouldn't have any problems. You could simply set up each system with a unique IP address between 216.30.120.1 and 216.30.120.254.

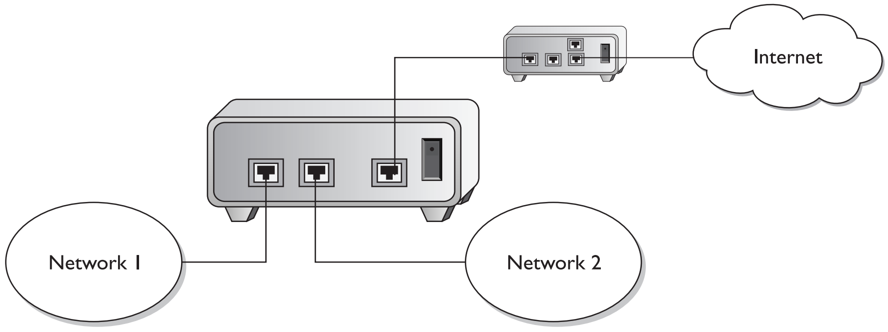


Figure 11-20 Adding another router to make separate networks—how does this work?

In this case, however, you can't do that. This router has two NICs—if each host system is randomly assigned an IP address from the total, how does the router decide which NIC to route packets through to be sure they get to the correct system? No, you can't just route everything through both NICs! The solution: divide those 256 IP addresses into two distinct groups in such a way that your router person can configure the router table to ensure that packets get to the right system. Just as we divide our neighborhoods into streets, we need to take this one Network ID with its Class C license and turn it into two distinct subnetwork IDs. Friend, you need to do classless subnetting!

Classless subnetting means to make subnets that aren't Class *A*, *B*, or *C* by defining the subnet mask at some point other than /8, /16, or /24. Classless subnetting is simple in concept, but complex in practice, largely because it requires us to break away from dividing our network IDs "at the periods." What does "at the periods" mean? I'm referring to the periods in a dotted decimal IP address, which you will recall are nothing more than a handy method for representing a set of binary values. So far, every network ID you've seen uses a *classful subnet*—the subnet mask always stops the first one, two, or three octets. There is no reason network IDs must end "on the periods." The computers, at least, think it's perfectly fine to have network IDs that end at points between the periods, such as /26, /27, or even /22. The trick here is to stop thinking about network IDs, and subnet masks just in their dotted decimal format, and instead go back to thinking of them as binary octets. Remember the binary/decimal conversion form? If I were you, I'd put this book on a photocopier right now and make yourself a bunch of copies of that form—you will need them!

We need to divide the Class C 216.30.120.0/24 network ID into two classless subnets. Note that the default subnet mask—the mask given to us by whoever gave use these IP addresses—is /24.We always start the classless subnetting process by writing out the default subnet mask in binary. The Class C subnet mask, equal to /24, is

11111111111111111111111100000000

To make classless subnets, we must extend the original subnet mask to the right, thereby turning our one network ID into multiple network IDs. The only downside is that we will have fewer host numbers in each of our new subnets. That's the cornerstone of understanding classless subnetting: you take the network ID you get from your ISP, and then chop it up into multiple, smaller network IDs, each of which supports fewer host numbers. The final step in this process is changing the IP addresses and subnet masks on all your systems, and entering the two (in this case) new network IDs into your gateway system. The hard part is knowing what to enter into the systems and routers.

If you order real, unique, ready-for-the-Internet IP addresses from your local ISP, you'll invariably get a classless set of IP addresses. More importantly, when you order them for clients, you need to be able to explain why their subnet mask is 255.255.255.192, when all the books they read tell them it should be 255.255.255.0!

Okay, let's stop and work from what you know. You know you need two subnets, right? So let's extend the subnet mask one place to the right (see Figure 11-21). This creates a three-section IP address: (1) the original network ID; (2) the now one-digit smaller host ID; and (3) nestled in between, what I call the subnet ID.

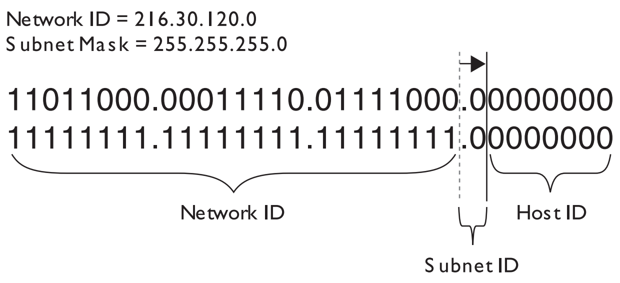


Figure 11-21 An IP address in three sections

Uh oh. Sorry, you can't do that—by rule of the Internet powers that be, you're not allowed to have a subnet ID that is all ones, or all zeroes either! We get around this by moving the subnet mask two places, as shown in Figure 11-22. How many new network IDs does that create? That would be 2^2, which is four, did I hear you say? Careful—remember the "all ones or zeroes" prohibition! Two of the subnets (00 and 11) are useless according to the rules of subnetting, so we only get two new network IDs.

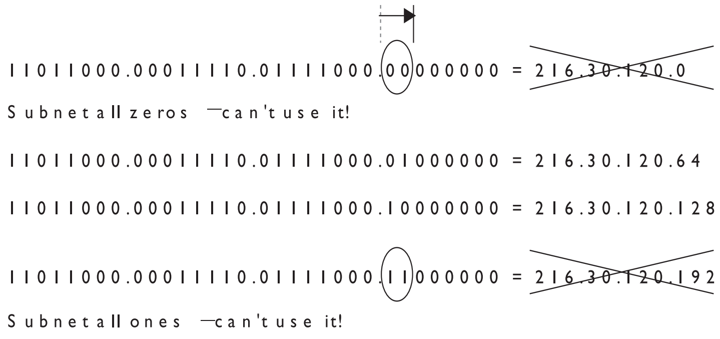


Figure 11-22 Extending the subnet mask two places

NOTE Even though the official rules do not allow subnets with all zeroes or all ones, most networking systems—certainly all versions of Windows—run just fine if you use them. But for the test, stick to the rules!

Take a good look at the two new dotted decimal network IDs. Do you see how they no longer end in a zero? Earlier I said you could tell a dotted decimal network ID because it ends in a zero, but these do not! True enough, but convert those network IDs to binary form. They still end in zeroes, but not at the periods. Only the last *six* digits of the IP address, which in this example are the host ID, are zeroes.

Now think about the subnet masks for each of the new subnets—how would you write a subnet mask of /26? That would be 255.255.255.192! Hey, these new subnet masks don't end in a zero either, or do they? Yes indeedy, they do, too! Remember that we count zeroes in the binary version of the address. The last six values in the fourth binary octet are zeroes. 11000000 equals 192 (128 + 64) in decimal, so we represent that subnet mask in dotted decimal as 255.255.255.192.

This "dotted decimal ID must end in a zero" issue drives a lot of folks crazy when they're first learning this stuff—they just can't "get past the periods," as I like to put it. It takes effort to stop thinking in terms of classful network IDs and subnets masks and instead think in a classless way. With some effort you will be able to appreciate that whether or not the subnet masks and network ID end in a zero when you write them as dotted decimal, everything still works the same way. It's useful to memorize the fact that any time you see a dotted decimal network ID ending in 128, 192, 224, or 240, you are looking at a classless network ID which, in its binary format, ends in zero.

NOTE I confess that this drove me crazy when I first learned subnetting years ago. Roll with it and you, too, can spend the rest of your life huddled next to a computer at odd hours writing books.

So, we now have two subnets: 216.30.120.64/26 and 216.30.120.128/26. If that isn't clear, stop reading now and go back to the previous text and graphics until it is clear. How do you use these two subnets in our example network? Well, the new subnet mask for every system is /26, so, first, we need to update the subnet mask settings in all our systems. We also know that our resident router geek must update the router to reflect the two new network IDs. The only question left is this: which IP addresses go to each subnet? Well, every IP address must include the network ID, leaving only six places for unique host numbers. In network ID 216.30.120.64/26, that gives us 64 different unique IP addresses. Just as no subnet may contain all zeroes or all ones, no host ID may be all zeroes and all ones, leaving a total of 62 usable IP addresses: 216.30.120.65 to 216.30.120.126 (see Figure 11-23). Contrast those with the IP addresses for the 216.30.120.128/26 network ID (see Figure 11-24).

Figure 11-23 Showing the IP addresses for 216.30.120.64

Network ID: 216.30.120.64 =

```
11011000.00011110.01111000.01000000

11011000.00011110.01111000.01000001 = 216.30.120.65
11011000.00011110.01111000.01000010 = 216.30.120.66
11011000.00011110.01111000.01000011 = 216.30.120.67
11011000.00011110.01111000.01000100 = 216.30.120.68
.......
11011000.00011110.01111000.01111011 = 216.30.120.123
11011000.00011110.01111000.01111100 = 216.30.120.124
11011000.00011110.01111000.01111101 = 216.30.120.125
11011000.00011110.01111000.01111110 = 216.30.120.126
```

Clearly, there is a relationship between the number of network IDs you create and the number of unique IP addresses available for each subnet. This is the tradeoff of subnetting: the more subnets, the fewer available host numbers per subnet. Also notice

Figure 11-24 Showing the IP addresses for 216.30.120.128

Network ID: 216.30.120.128 =

```
11011000.00011110.01111000.10000000

11011000.00011110.01111000.10000001 = 216.30.120.129
11011000.00011110.01111000.10000010 = 216.30.120.130
11011000.00011110.01111000.10000011 = 216.30.120.131
11011000.00011110.01111000.10000100 = 216.30.120.132
.......
11011000.00011110.01111000.10111011 = 216.30.120.187
11011000.00011110.01111000.10111100 = 216.30.120.188
11011000.00011110.01111000.10111101 = 216.30.120.189
11011000.00011110.01111000.10111110 = 216.30.120.190
```

that subnetting has reduced our original complement of 256 unique IP addresses to only 124. Here's why:

256	(IP addresses to start)
–128	(The two subnets that would be all zeroes or all ones)
–4	(Two host numbers in each subnet that would be all zeroes or all ones)
=124	(Usable IP addresses)

Wow! Subnetting sure costs you a lot of IP addresses! No wonder most systems let you cheat the official subnetting rules and use subnets that contain all zeroes or all ones! If your systems allow it (Windows, NetWare, and Linux do, although you may need to tweak them), and if your router allows it (most do, but your router person may have to configure some special settings), you don't have to lose all these IP addresses. But remember, for the test, follow all the rules strictly.

Let's now plug some real values into our example network. Note that we had the router person update the router table for each subnet (see Figure 11-25).

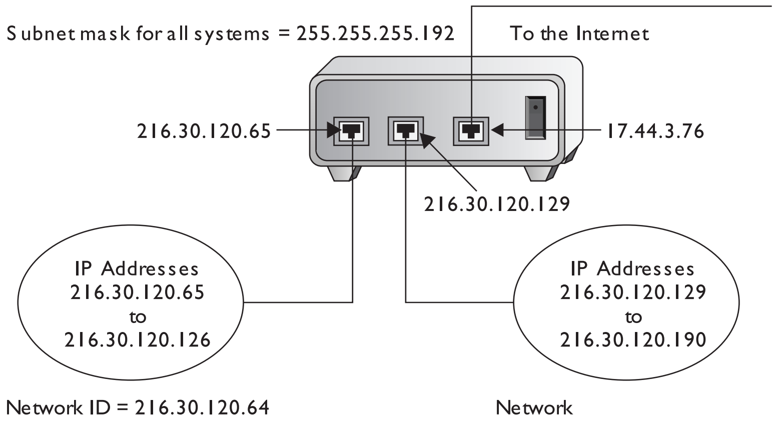

Subnet mask for all systems = 255.255.255.192
To the Internet
216.30.120.65
17.44.3.76
216.30.120.129
IP Addresses 216.30.120.65 to 216.30.120.126
IP Addresses 216.30.120.129 to 216.30.120.190
Network ID = 216.30.120.64
Network

Figure 11-25 Working subnet router settings

Special IP Addresses

The folks who invented TCP/IP created a number of special IP addresses you need to know about. The first special address is 127.0.0.1, the famous loopback address. When you tell a device to send data to 127.0.0.1, you're telling that device to send the packets to itself. The loopback address has a number of uses; one of the most common is to use it with the **PING** command. We use the command **PING 127.0.0.1** to test a NIC's capability to send and receive packets.

NOTE Even though by convention we use 127.0.0.1 as the loopback address, the entire 127.0.0.0/8 subnet is reserved for loopback! You can use any address in the 127.0.0.0/8 subnet as a loopback address.

Lots of folks use TCP/IP in networks that either aren't connected to the Internet or that want to hide their computers from the rest of Internet. Certain groups of IP addresses, known as private IP addresses, are available to help in these situations. All routers are designed to destroy private IP addresses so they can never be used on the Internet, making them a handy way to hide systems! Anyone can use these private IP addresses, but they're useless for systems that need to access the Internet—unless you use one of those mysterious NAT things to which I just referred. (Bet you're dying to learn about NATs now!) For the moment, however, let's just look at the ranges of addresses that are designated private IP addresses:

- 10.0.0.0 through 10.255.255.255 (1 Class A license)
- 172.16.0.0 through 172.31.255.255 (16 Class B licenses)
- 192.168.0.0 through 192.168.255.255 (256 Class C licenses)

We refer to all other IP addresses as public IP addresses.

TIP Make sure you can quickly tell the difference between a private and a public IP address for the Network+ exam!

Other Critical TCP/IP Settings

You now understand the three most important settings on every TCP/IP computer: the IP address, the default gateway, and the subnet mask. But most computers on a TCP/IP network will need a few more critical settings. Let's take a look at DNS, WINS, and DHCP.

DNS

Anyone who's ever used a web browser is used to seeing Internet addresses like www.totalsem.com, ftp.microsoft.com, and so on; these invariably end with .com, .org,

.net, or some other three-character name (or two-character country code, like .uk for the United Kingdom). Now just a minute here—haven't we spent the last several pages explaining how every computer on the Internet has a unique IP address that distinguishes it from every other computer on the Internet? What's with this name business all of a sudden? Well, the folks who invented the Internet decided early on that it was way too painful for human beings to refer to computer systems using dotted decimal notation. There had to be some sort of Internet-naming convention that allowed people to refer to systems by human-friendly names, rather than cryptic but computer-friendly numbers. They couldn't just do away with the numbers, however, unless they were planning to teach the computers to read English, so they developed a procedure called *name resolution*. The idea was to have a list of IP addresses matched up with corresponding human-friendly names, that any computer on the Internet could use to translate a computer name into an IP address.

The original IP specification implemented name resolution using a special text file called *HOSTS*. A copy of this file was stored on every computer system on the Internet. The HOSTS file contained a list of IP addresses for every computer on the Internet, matched to the corresponding system names. Remember, not only was the Internet a lot smaller then, there weren't yet rules about how to compose Internet names, like they must end in .com or .org, or start with www or ftp. Anyone could name their computer pretty much anything they wanted (there were a few restrictions on length and allowable characters) as long as nobody else had snagged the name first. Part of an old HOSTS file might look something like this:

```
192.168.2.1    fred

201.32.16.4    school2

123.21.44.16   server
```

If your system wanted to access the system called fred, it looked up the name fred in its HOSTS file, and then used the corresponding IP address to contact fred. Every HOSTS file on every system on the Internet was updated every night at 2 A.M. This worked fine when the Internet was still the province of a few university geeks and some military guys, but when the Internet grew to about 5,000 systems, it became impractical to make every system use and update a HOSTS file. This motivated the creation of the *Domain Name Service (DNS)* concept.

Believe it or not, the HOSTS file is still alive and well in every computer. You can find the HOSTS file in the \WINDOWS folder in Windows 9*x*, in the \WINNT\SYSTEM32\DRIVERS\ETC folder in Windows NT/2000/2003, and in \WINDOWS\SYSTEM32\DRIVERS\ETC in Windows XP. It's just a text file that you can open with any text editor. Here are a few lines from the default HOSTS file that comes with Windows. See the # signs? Those are remark symbols that designate lines as comments rather than code—take them off and Windows will read the lines and act on them. While all operating systems continue to support the HOSTS file, it is rarely used in the day-to-day workings of most TCP/IP systems.

```
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#         102.54.94.97      rhino.acme.com             # source server
#          38.25.63.10      x.acme.com                 # x client host
127.0.0.1          localhost
```

How DNS Works

The Internet guys, faced with the task of replacing HOSTS, first came up with the idea of creating one super computer that did nothing but resolve names for all the other computers on the Internet. Problem: even now, no computer is big enough or powerful enough to handle the job alone. So, they fell back on that time-tested bureaucratic solution: delegation! The top-dog DNS system would delegate parts of the job to subsidiary DNS systems, who, in turn, would delegate part of their work to other systems, and so on, potentially without end. These systems run a special DNS server program, and are called, amazingly enough, DNS servers. This is all peachy, but it raises another issue: you need some way to decide how to divvy up the work. Toward this end, they created a naming system designed to facilitate delegation. The top dog DNS server is a bunch of powerful computers dispersed around the world and working as a team, known collectively as the DNS root. The Internet name of this computer team is "."—that's right, just "dot." Sure, it's weird, but it's quick to type, and they had to start somewhere! DNS root has the complete definitive name resolution table, but most name-resolution work is delegated to other DNS servers. Just below the DNS root in the hierarchy is a set of DNS servers that handle what are known as the top-level domain names. These are the famous COM, ORG, NET, EDU, GOV, MIL, and INT. (Even these are getting full—you may have seen news stories about new additions to this top-level domain list.) These top-level DNS servers delegate to thousands of second-level DNS servers; they handle the millions of names like totalsem.com and whitehouse.gov that have been created within each of the top-level domains.

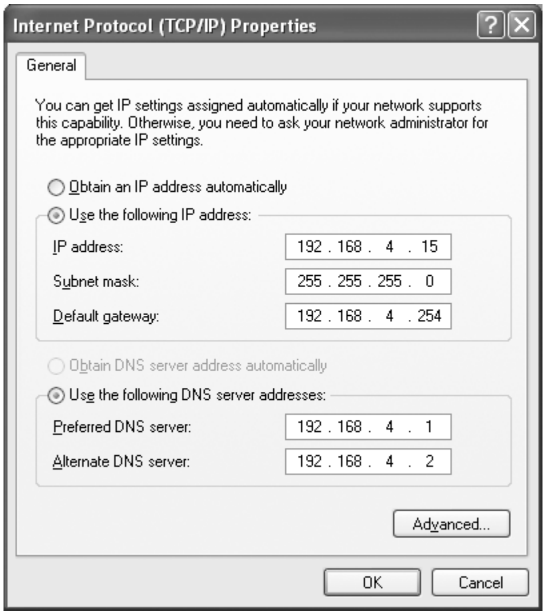
These domain names must be registered for Internet use with an organization called ICANN (http://www.icann.org). They are arranged in the familiar "second level.top level" domain name format, where the top level is COM, ORG, NET, and so on, and the second level is the name of the individual entity registering the domain name. For example, in the domain name microsoft.com, "microsoft" is the second-level part and "com" is the top-level part. As we just learned, the Internet maintains a group of powerful and busy DNS servers that resolve the top-level parts of domain names. The second-level parts of domain names are resolved by the owners of those domain names, or by an ISP.

A DNS network may also have subdomains. The subdomain names are added to the left of the domain name. Let's use our standard example: the MHTechEd network, which has the domain name mhteched.com, may have subdomains with names like north.mhteched.com and south.mhteched.com. The owner of the second-level domain, MHTechEd in our example, maintains any subdomains on its own DNS servers or its ISP's DNS servers. Subdomains may also contain a further layer of subdomains (for example, bravo.north.mhteched.com), but this is rare, and usually is handled by in-house DNS servers where it does exist.

Every system on the Internet also has a host name added on to the left of its domain name, such as “www” for a web server or perhaps “domain1” for a domain controller in a Windows network. A *fully qualified domain name (FQDN)* contains the complete DNS name of a system, from its host name to the top-level domain name. I have a system in my office with an FQDN of vpn.totalsem.com that handles my virtual private network. I also have a system with the FQDN of www.totalsem.com that runs my web site. How about that—seems you’ve been using FQDNs for years and didn’t even know it.

A system will have its own list of domain names, stored either in its HOSTS file or in a cache. When a system needs to resolve a domain name to an IP address and it doesn’t have this information, it queries the DNS server listed in its DNS server settings. If the DNS server cannot resolve the name, the DNS server asks the root server. The root server then redirects your DNS server to a top-level DNS server. The top-level DNS server, in turn, points you to a second-level DNS server that will resolve the domain name. We’ll look at how DNS works in much more detail in Chapter 14; right now, let’s see how you configure a Windows client system to use DNS.

You configure DNS in Windows using the TCP/IP Properties dialog box. Figure 11-26 shows the DNS settings for my system. Note that I have more than one DNS server setting; the second one is a backup in case the first one isn’t working. Two DNS settings is not a rule, however, so don’t worry if your system shows only one DNS server setting, or perhaps more than two. You can check your current DNS server settings in Windows using either the **WINIPCFG** or the **IPCONFIG** command.



You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

Obtain an IP address automatically [✓ Use the following IP address:]

IP address: 192 . 168 . 4 . 15
Subnet mask: 255 . 255 . 255 . 0
Default gateway: 192 . 168 . 4 . 254

Obtain DNS server address automatically [✓ Use the following DNS server addresses:]

Preferred DNS server: 192 . 168 . 4 . 1
Alternate DNS server: 192 . 168 . 4 . 2

Figure 11-26
DNS settings in Windows XP

That's pretty much all we need to talk about right now. What you need to know about DNS for the Network+ exam is how to set up the DNS server for your system if given the configuration information. In Chapter 14, "Going Large with TCP/IP," we look at DNS in more detail, including what it takes to make a DNS server!

DHCP

Before you read this, you must promise not to hit me, okay? So far we've discussed four items that must be configured for every system in an IP network: the IP address, the default gateway, the subnet mask, and the DNS server. Does this sound like a bit of a hassle? Imagine being informed by some network muckety-muck that your organization is changing its DNS servers, requiring you to reset the DNS server settings on every system in the network. Doesn't sound like too much fun, eh?

Fortunately for you, there's something called the *Dynamic Host Configuration Protocol (DHCP)*. DHCP enables individual client machines on an IP network to configure all of their IP settings (IP address, subnet mask, DNS, and more) automatically. DHCP requires a special system running a special program called a DHCP server. Autoconfiguration has been a boon to networks that change a lot (like mine), systems that travel around (like laptops), and dial-up systems. DHCP is extremely popular and heavily used in almost all IP networks.

TIP DHCP is used mainly by Windows networks. There's another autoconfiguration protocol called BOOTP that is used for diskless workstations. They both work basically the same way.

How Does DHCP Work?

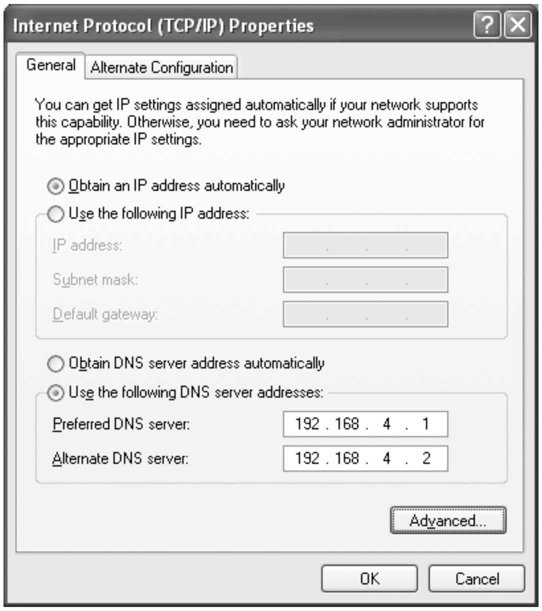
All of this DHCP automatic self-configuration doesn't let administrators completely off the hook, of course. Somebody has to set up the DHCP server before it can make the magic happen throughout the network. The DHCP server stores all of the necessary IP information for the network, and provides that information to clients as they boot into the network.

You also have to configure your network clients to use DHCP. Configuring a Windows system for DHCP is easy. Yes, once again, go to the TCP/IP Properties dialog box. This time, set the IP Address to *Obtain An IP Address Automatically*, and you're ready to go (see Figure 11-27). Just be sure you have a DHCP server on your network!

WINS

Years ago, Microsoft and TCP/IP didn't mix too well. Microsoft networks leaned heavily on the NetBIOS/NetBEUI protocols, and NetBIOS used a completely different naming convention than TCP/IP. To implement an IP network protocol in Windows NT while still using NetBIOS names, the programmers realized that NT needed some way to resolve NT's NetBIOS names into IP addresses. At the time, Microsoft didn't foresee the impact of the Internet; it seemed to make more sense to ignore DNS, because dropping

Figure 11-27 Configuring a Windows XP system for DHCP



Internet Protocol (TCP/IP) Properties

General | Alternate Configuration

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

[✓ Obtain an IP address automatically]
Use the following IP address:
IP address: . . .
Subnet mask: . . .
Default gateway: . . .

Obtain DNS server address automatically
[✓ Use the following DNS server addresses:]
Preferred DNS server: 192 . 168 . 4 . 1
Alternate DNS server: 192 . 168 . 4 . 2

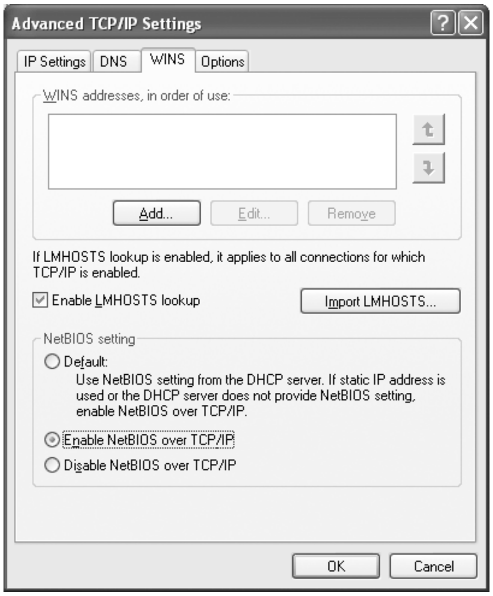
NetBIOS would require a major redesign of Windows NT; instead they came up with an equivalent that used the already existing NetBIOS functions.

NOTE Windows 2000/XP/2003 systems no longer need NetBIOS; they now use DNS, although they can support WINS if needed. Windows 9*x* and NT systems, however, still need WINS to use NETBIOS names across routed networks!

But why make a naming system that resolves IP addresses to NetBIOS names? As we saw in the previous chapter, name resolution in Windows has always depended on individual hosts broadcasting to the network their desire to use a particular NetBIOS name. Broadcasting was the problem! Routers do not allow broadcasting in an IP environment. Microsoft solved this dilemma with the creation of the *Windows Internet Naming Service (WINS)*.

WINS is a hierarchical naming system, similar to DNS in many ways. Just as DNS depends on DNS servers, WINS revolves around special WINS server software. A WINS server tracks all of the requests for NetBIOS names on a Windows TCP/IP network. When computers first log on to a WINS network, they register with the WINS server, and the server puts the client's NetBIOS name and IP address into its WINS database. Any system on the network can query the WINS server for NetBIOS name resolution.

Happily for network techs, Microsoft automated many aspects of WINS, making the actual setup and configuration of WINS almost trivially easy. Setting up WINS on a Windows client requires little more than a quick trip to the WINS Configuration tab in—can you guess?—the TCP/IP Properties dialog box, where you can select Use DHCP for WINS resolution (see Figure 11-28). There are other setup options as well, which we discuss in Chapter 14, "Going Large with TCP/IP."



WINS addresses, in order of use:

If LMHOSTS lookup is enabled, it applies to all connections for which TCP/IP is enabled.

[✓] Enable LMHOSTS lookup

NetBIOS setting

○ Default:
Use NetBIOS setting from the DHCP server. If static IP address is used or the DHCP server does not provide NetBIOS setting, enable NetBIOS over TCP/IP.

[✓ ● Enable NetBIOS over TCP/IP]

○ Disable NetBIOS over TCP/IP

Figure 11-28 WINS settings in Windows XP

The toughest part about DNS, WINS, and DHCP is the potential to confuse what each one does, so here's a quick review to help you remember each one's job.

- **DNS** A TCP protocol that resolves domain names (for example, www.totalsem.com) into actual IP addresses
- **WINS** A Windows-only protocol used to resolve NetBIOS names into IP addresses
- **DHCP** A protocol that automatically configures a system's IP information for end users

You may have noticed that DNS, WINS, and DHCP all require a server. Be careful with this terminology. In this context, server does not mean a physical computer—it means a program that handles DNS, DHCP, or WINS requests from client computers. It's fairly common, especially on smaller networks, for one server system to run all three of these server programs simultaneously.

IP Ports

The term TCP/IP is a bit of a misnomer. It implies that TCP is the only type of data that IP packets transport. In reality, while TCP is certainly the most common type of data found in IP packets, it's certainly not the only one. IP supports a number of different types of data, each one used for a very specific purpose. Let's take a moment to look at TCP and two other important IP data types—UDP and ICMP—and see what they do.

TCP

Most folks who work in a network environment, especially a network with systems running Microsoft Windows, think that a network's main function is to share folders and printers. Back when the Internet was young, however, the folks who designed TCP/IP didn't think exactly in those terms. They looked at a network as a way to share terminals (remember this discussion from earlier in the book?), exchange e-mail, and perform other functions unrelated to our concept of networks as a way to share files and printers. But lucky for us, they also realized they would probably be using TCP/IP to share things they hadn't yet invented, so they designed it with a degree of flexibility that enables you and me today to share files, surf the Web, listen to streaming media, and play Everquest online with a thousand of our closest buddies at once. This is TCP's reason for being.

The inventors of TCP/IP assign a special number called a port to each separate network function, such as e-mail, web browsing, even online games like Everquest. This port number is placed inside every IP packet and is used by the sending and receiving systems to figure out which application to give the packet to. Possibly the most famous TCP port of all is good old port 80. If a packet comes into a computer with the port number 80 embedded in it, the system knows that the packet needs to go to your web browser. Packets sent to port 80 use a special protocol called HyperText Transfer Protocol (HTTP).

UDP

If you get a chance, take a look at the link light on your NIC when you first start your system. You'll notice that the light is flickering away, showing that some kind of communication is taking place. Granted, not all of that communication is for your system, but trust me, a lot of it is—and you haven't even started your e-mail or opened a web browser! Clearly your PC is talking on the network even though you haven't asked it to do anything! Don't worry—this is a good thing. TCP/IP does lots of boring maintenance-type stuff in the background that you neither need nor want to care about. Most of these maintenance communications are simple things like an ARP request, or any one of about 500 other little maintenance jobs that TCP/IP handles for you automatically.

While these things are important, they do not require all the TCP information in an IP packet. Knowing this, the folks who designed TCP/IP created a much simpler protocol called *User Datagram Protocol (UDP)*. UDP transmits much less information than TCP. UDP packets are both simpler and smaller than TCP packets, and do most of the behind-the-scenes work in a TCP/IP network. UDP packets are what is called connectionless—that is, they don't worry about confirming that a packet reached its destination. TCP packets, in contrast, are connection-oriented—in other words, they must create a connection between the sending and receiving systems to ensure that the packet reaches its destination successfully. Important functions like e-mail will never use UDP.

Some older applications, most prominently TFTP (Trivial FTP), use UDP to transfer files. But even TFTP is quite rare today; UDP is almost completely relegated to important, but behind-the-scenes jobs on the network.

ICMP

Applications that use the *Internet Control Message Protocol (ICMP)* are even simpler than UDP applications. ICMP messages consist of a single packet and are connectionless; like UDP. ICMP packets determine connectivity between two hosts. As such, they carry only a tiny set of responses, such as *echo reply*, *protocol unreachable*, or *host unreachable*. In a way, ICMP is simply a way to see things at the IP level. ICMP packets don't transfer data *per se*, but rather tell you how the IP packets between any two hosts are doing. The most famous ICMP application is PING.

NOTE There is another type of IP data called IGMP (Internet Group Management Protocol). IGMP is used to take advantage of multicasting—broadcasting a series of packets to very specific systems. IGMP is used for some videoconferencing and other applications.

Table 11-1 shows a list of the most common of all the TCP and UDP ports and their uses. Keep in mind that Table 11-1 shows only the most common ones—there are literally hundreds more than these! For the Network+ exam, at least be sure you memorize this table.

Here are brief explanations of these ports, most of which I'll revisit in detail in later chapters.

HTTP (Port 80)

Web servers use *HyperText Transfer Protocol (HTTP)* to send web pages to clients running web browsers such as Internet Explorer or Mozilla Firefox.

FTP (Ports 20 and 21)

File Transfer Protocol (FTP) transfers data files between servers and clients. All implementations of TCP/IP support FTP file transfers, making FTP an excellent choice for transferring files between machines running different operating systems (Windows to UNIX, UNIX to Macintosh, and so on). FTP uses port 21 for control messages and sends the data using port 20. FTP servers can require users to log in before downloading or uploading files. Most operating systems include a command-line FTP utility.

SFTP (Port 22)

FTP has one rather ugly aspect: by default, FTP transfers are not encrypted. Anyone clever enough to monitor an FTP session can intercept the username, password, and data. Secure FTP encrypts everything moving between the server and the client, so that the username, password, and data are safe from prying eyes.

Port Number	Service	Description
20	FTP DATA	File Transfer Protocol – Data, used for transferring files
21	FTP	File Transfer Protocol – Control, used for transferring files
23	TELNET	Telnet, used to gain "remote control" over another machine on the network
25	SMTP	Simple Mail Transfer Protocol, used for transferring e-mail between e-mail servers
69	TFTP	Trivial File Transfer Protocol, used for transferring files without a secure login
80	HTTP	HyperText Transfer Protocol, used for transferring HTML (HyperText Markup Language) files, i.e., web pages
110	POP3	Post Office Protocol version 3, used for transferring e-mail from an e-mail server to an e-mail client
119	NNTP	Network News Transfer Protocol, used to transfer Usenet newsgroup messages from a news server to a newsreader program
123	NTP	Network Time Protocol, used to synchronize the time of a server or workstation to another server
137	NETBIOS-NS	NetBIOS Name Service, used by Microsoft Networking
138	NETBIOS-DG	NetBIOS Datagram Service, used for transporting data by Microsoft Networking
139	NETBIOS-SS	NetBIOS Session Service, used by Microsoft Networking
161	SNMP	Simple Network Management Protocol, used to monitor network devices remotely
389	LDAP	Lightweight Directory Access Protocol, used to communicate with network directories/databases
443	HTTPS	HyperText Transfer Protocol with Secure Sockets Layer (SSL), used to transfer secure HTML files

Table 11-1 Common TCP and UDP ports

TFTP (Port 69)

Trivial File Transfer Protocol (TFTP) transfers files between servers and clients. Unlike FTP, TFTP requires no user login. Devices that need an operating system, but have no local hard disk (for example, diskless workstations and routers), often use TFTP to download their operating systems.

SMTP (Port 25)

Simple Mail Transfer Protocol (SMTP) sends e-mail messages between clients and servers or between servers. From the end user's perspective, SMTP handles outgoing mail only.

POP3 (Port 110)

Post Office Protocol version 3 (POP3) enables e-mail client software (for example, Outlook Express, Eudora, Netscape Mail) to retrieve e-mail from a mail server. POP3 does not send e-mail; SMTP handles that function.

SNMP (Port 161)

Simple Network Management Protocol (SNMP) enables network management applications to remotely monitor other devices on the network.

Telnet (Port 23)

Telnet enables a user to log in remotely and execute text-based commands on a remote host. Although any operating system can run a Telnet server, techs typically use Telnet to log in to UNIX-based systems.

NetBIOS (Ports 137, 138, 139)

Networks using NetBIOS over TCP/IP use ports 137, 138, and 139 for name resolution and other NetBIOS-specific tasks.

In most cases, you will never have to deal with ports. When you install an e-mail program, for example, your system will automatically assume that all packets coming on ports 110 and 25 are e-mail and will send them to the e-mail application. You don't have to do anything special to configure the port values—they just work. In later chapters, we will discuss configuring TCP/IP applications, and we'll see that, in some cases, you may have to change port numbers, but for now just be sure to memorize the port numbers in Table 11-1.

TIP Although CompTIA claims to be vendor-neutral; TCP/IP questions on the Network+ exam generally assume that the client system uses a Microsoft Windows operating system.

NTP (Port 123)

Network Time Protocol (NTP) is a UDP protocol that has only one job: to announce the time. Think of it as the Big Ben of TCP/IP.

LDAP (Port 389)

The Lightweight Directory Access Protocol (LDAP) is the common language used to communicate with network directories. See Chapter 12, "Network Operating Systems," for more details on the many types of directories used by network operating systems.

IPv6

One of the big problems with TCP/IP stems from the fact that we seem to be running out of IP addresses. In theory, the 32-bit IP address under the current *IP version 4* specification (we officially call it *IPv4*) allows for 2^{32} or over 4 billion addresses. However, due to many restrictions, only about 1.7 billion are available, and many of those are wasted by organizations that take more IP addresses than they need. As a result, the Internet Engineering Task Force (IETF) developed a new IP addressing scheme, called *IP version 6*, abbreviated *IPv6* (first known as IP Next Generation [IPng]), that is expected to gradually replace IPv4 in coming years. IPv6 extends the 32-bit IP address to 128 bits, allowing up to 3.4×10^{38} addresses! Probably enough to last us for awhile, eh?

Remember that the IPv4 addresses are written as 197.169.94.82 using four octets. Well, IPv6 has now changed all that. IPv6 addresses are written like this:

FEDC:BA98:7654:3210:0800:200C:00CF:1234

IPv6 uses a colon as a separator, instead of the period used in IPv4's dotted decimal format. Each group is a hexadecimal number between 0000 and FFFF. As a refresher for those who don't play with hex regularly, one hexadecimal character (for example, *F*) represents four bits, so four hexadecimal characters make a 16-bit group. When writing IPv6 addresses, leading zeroes can be dropped from a group, so 00CF becomes simply CF, and 0000 becomes just 0. To write IPv6 addresses containing strings of zeroes, you can use a pair of colons (::) to represent a string of consecutive 16-bit groups with a value of zero. For example, using the :: rule you can write the IPv6 address FEDC:0000:0000:0000:00CF:0000:BA98:1234 as FEDC::CF:0:BA98:1234. Notice that I could not use a second :: to represent the third-to-last group of four zeroes—only one :: per address! There's a good reason for this rule. If more than one :: was used, how could you tell how many sets of zeroes were in each group? Table 11-2 shows some of the reserved IPv6 addresses.

IPv6	:: rule applied to IPv6 address	Function
1080:0:0:0:8:800:200C:417A	1080::8:800:200C:417A	Unicast address
0:0:0:0:0:0:0:1	::1	Loopback address
0:0:0:0:0:0:0:0	::	Unspecified address

Table 11-2 IPv6 Reserved Addresses and Their Function

NOTE The unspecified address (all zeroes) can never be used and neither can an address that contains all ones.

Chapter Review

Questions

1. A host is
   - **A.** Any server on a TCP/IP network
   - **B.** Any device on a TCP/IP network that can send or receive data packets
   - **C.** A device on a TCP/IP network that forwards data packets to other networks
   - **D.** A device on a TCP/IP network that resolves names to IP addresses
2. Before a TCP/IP host can communicate with another host on a different network, which of the following settings must it have configured correctly? (Select all that apply.)
   - **A.** IP address
   - **B.** Subnet mask
   - **C.** DNS server
   - **D.** Default gateway
3. What port number does HTTP use?
   - **A.** 443
   - **B.** 110
   - **C.** 80
   - **D.** 43
4. The binary number 11000101 has the decimal equivalent of
   - **A.** 197
   - **B.** 169
   - **C.** 94
   - **D.** 82
5. What is the loopback address for IPv4?
   - **A.** 127.0.0.1
   - **B.** 0:0:0:0:0:0:0:0
   - **C.** ::1
   - **D.** 0:0:0:0:0:0:0:1
6. The protocol that enables network management applications to monitor devices remotely is known as
   - **A.** SNMP
   - **B.** SMTP

C. POP3

D. TFTP

7. John is running Windows 98. He wants to find out his MAC address and IP address. What utility can John use to provide this information?

   A. ARP

   B. WINIPCFG

   C. IPCONFIG

   D. MACIP

8. Mike is running Windows 2000. He wants to find out his MAC address and IP address. What utility can Mike use to provide this information?

   A. ARP

   B. WINIPCFG

   C. IPCONFIG

   D. MACIP

9. Scott's system wants to send data to Roger's system. Scott's system knows Roger's IP address, but it doesn't know the MAC address, which it needs. What does the system use to request a MAC address for a known IP address?

   A. ARP

   B. WINIPCFG

   C. IPCONFIG

   D. MACIP

10. The IP address 192.23.45.123 has a default subnet mask of

    A. 255.0.0.0

    B. 255.255.0.0

    C. 255.255.255.0

    D. 255.255.255.255

Answers

1. **B.** Any device on a TCP/IP network that can send or receive data packets is called a host.
2. **A, B, D.** A host on a TCP/IP network must have its IP address, subnet mask, and default gateway correctly configured before it can communicate with hosts on other networks.
3. **C.** HyperText Transfer Protocol (HTTP) uses port 80.

4. **A.** The binary number 11000101 has the decimal equivalent of 197.
5. **A.** The loopback address for IPv4 is 127.0.0.1.
6. **A.** Simple Network Management Protocol (SNMP) enables network management applications to monitor devices remotely.
7. **B.** On a Windows 98 system, the WINIPCFG utility gives the MAC address and IP address currently used by that system.
8. **C.** On a Windows 2000 system, the IPCONFIG utility gives the MAC address and IP address currently used by that system.
9. **A.** The sending system sends out an address resolution protocol (ARP) request to get the MAC address for a known IP address.
10. **C.** The IP address 192.23.45.123 is in Class C and has the default subnet mask of 255.255.255.0.