Subnetting Made Simple

IP Subnetting without Tables, Tools, or Tribulations

Larry Newcomer
The Pennsylvania State University
York Campus

Abstract

Every networking professional should have a thorough understanding of TCP/IP subnetting. Subnetting can improve network performance by splitting up collision and broadcast domains. Subnets can reflect organizational structure and help support security policies. WAN links typically join different subnets. Subnets can define administrative units and hence support the structuring and delegation of administrative tasks. Unfortunately, mastering subnetting can pose difficulties for both professionals and students because of the binary mathematics that underlies the technology. While it is imperative to present subnetting concepts in terms of the underlying binary representation, most texts also present subnetting procedures in binary terms. Such an approach can make it difficult for students to learn how to actually carry out subnetting without tables or other reference materials, even when they understand the basic concepts. This paper presents a simple, alternative method for understanding and implementing subnetting without software, calculators, tables, or other aids. The only knowledge of binary arithmetic required is familiarity with the powers of 2 from 0 to 8 (2^x for $x = 0, 1, \ldots, 8$). With a little decimal arithmetic thrown in, the whole process is simple enough to be carried out mentally. This paper assumes the reader is already somewhat familiar with IP addressing, the role of subnet masks, and the uses for subnetting. It proceeds quickly from a brief introduction to a thorough discussion of simple techniques for determining the number of subnets and hosts, calculating the subnet mask, determining (sub)network id's, and figuring the available IP addresses for each subnet. The methodology is helpful both to those who aspire to be network professionals and to those who seek a simple way to teach subnetting in networking courses.

Introduction

Every networking student should have a solid understanding of TCP/IP subnetting (Loshin, 1997). Subnetting's importance in modern networking is reflected by its many and varied uses. It can enhance network performance by splitting up collision and broadcast domains in a routed network (Odom, 2000). Large networks can be organized into separate subnets representing departmental, geographical, functional, or other divisions (Feit, 1997). Since hosts on different subnets can only access each other through routers, which can be configured to apply security restrictions, subnetting can also serve as a tool for implementing security policies (Bulette, 1998). Dividing a large network into subnets and delegating administrative responsibility for each subnet can make administration of a large network easier. Routers can require that a WAN link connecting two networks must itself form a separate subnet (Bulette, 1998). Troubleshooting, diagnosing, and fixing problems in a TCP/IP internetwork typically require thorough familiarity with subnetting. Network design requires both the ability to understand and carry out subnetting.

A major stumbling block to successful subnetting is often a lack of understanding of the underlying binary math. In fact, the principles of subnetting are difficult to grasp without mastery of binary arithmetic, logic, and binary/decimal conversions. On the other hand, it is not necessary to be able to think in binary in order to plan, design, and implement simple subnetting. The methodology discussed below allows anyone with the following capabilities to successfully carry out all the subnetting essentials:

- Know the decimal values of the powers of 2 from 0 to 8 as presented in Table 1 below (e.g., $2^4 = 16$)
- Know how to add and subtract the *decimal* values of the powers of 2 in the following table (e.g., $2^7 – 2^5 = 128 – 32 = 96$). If this decimal arithmetic can be done mentally, then subnetting itself can be accomplished mentally.

Table 1

x	**2^x**	**2^x in Decimal**
0	2^0	1
1	2^1	2
2	2^2	4
3	2^3	8
4	2^4	16
5	2^5	32
6	2^6	64
7	2^7	128

x	2^x	2^x in Decimal
8	2^8	256

It is helpful to memorize Table 1 before proceeding. The discussion below presents a simple, step-by-step methodology for determining the number of subnets, the maximum number of hosts per subnet, the subnet mask, the network id's, and the valid IP addresses for each subnet. In short, the method covers all the major steps in the subnetting process. First, however, we present a rapid review of subnetting concepts.

Simple Subnetting Concepts

Before discussing the subnetting procedure, it is necessary to become familiar with the basic concepts of IP addressing, subnet masks, and the separation of an IP address into a network ID and a host ID. An IPv4 (IP version 4) address consists of a 32-bit binary number, which can be viewed as a series of 4 *octets* (Odom, 1999). Each octet consists of 8 bits, so 4 octets make up 4 * 8 = 32 bits, the exact number of bits in an IP address.

Since IP addressing operates at the OSI *network layer* (Layer 3), IP addresses must be able to identify both individual *hosts* (the TCP/IP term for any device connected by a network adapter to a TCP/IP network, such as a computer, printer, router, etc.) and individual *networks*. This is accomplished by dividing the 32-bit IP addresses into two parts: an initial *network ID* portion that identifies (i.e., addresses) individual networks, followed by a *host ID* portion that identifies (i.e., addresses) individual hosts on a given network (Lammle, 2000). Thus IP addressing works by uniquely specifying:

a) A particular TCP/IP network as identified by the *network ID* portion of the IP address

b) A particular host on that network as identified by the *host ID* portion of the IP address

It is critical to understand the difference between the MAC address (also known as the hardware, physical, or NIC address) at the Data-Link layer (Layer 2), and the IP address which operates at the Network layer (Layer 3). The MAC address uniquely identifies each network adapter (NIC or Network Interface Card) with a 48-bit binary number. The assignment of MAC addresses is supervised by the IEEE to ensure worldwide uniqueness. Each MAC address consists of two parts: the first part uniquely identifies the NIC manufacturer, and the second part uniquely identifies each NIC produced by a given manufacturer (Poplar, 2000). A device attached to a TCP/IP network via a NIC is called a TCP/IP *host*. Note that the MAC addressing scheme does *not* provide a way to identify individual *networks*, only individual *hosts*.

With IP addressing, individual hosts can still be uniquely identified, but in a different way. Hosts are identified by specifying both: a) the network on which the host resides (the network id), plus b) a unique host number on that network (the host ID). Thus Layer 3 devices can work either with individual networks by using just the network ID (the beginning portion of the IP address), or with individual hosts by using both the network ID and host ID, i.e., the entire IP address (Heywood, 1997).

Network layer (Layer 3) software and devices thus must be able to separate IP addresses into their network ID and host ID portions. This is accomplished with the help of another 32-bit binary number called a *subnet mask*. The job of the subnet mask is to tell which part of the IP address is the network ID, and which part is the host ID. Since the network ID is always the leading part of an IP address and the host ID is always the trailing part, a simple *masking* scheme can be used (Dulaney, 1998). A subnet mask always consists of a series of uninterrupted 1 bits followed by a series of uninterrupted 0 bits. These two portions of the subnet mask (all 1's and all 0's) correspond to the two parts of an IP address. The 1 bits in the subnet mask match up with the Network ID, and the 0 bits in the subnet mask match up with the Host ID in the IP address. By looking at the subnet mask, it is easy to tell which part of an IP address represents the network ID and which part represents the host ID. The following example illustrates the use of a subnet mask to separate the network ID and host ID portions of an IP address for a standard Class B network (for Class B networks the first 2 octets of the IP address make up the network ID and the last 2 octets make up the host ID):

```
32-bit IP Address:  10010010101010000000000000000111
32-bit Subnet Mask: 11111111111111110000000000000000
```

To enhance clarity we repeat the example above, this time adding some white space between the octets in both the IP address and the subnet mask:

```
IP Address:    10010010 10101000 00000000 00000111
Subnet Mask:   11111111 11111111 00000000 00000000
```

Observe how the subnet mask consists of 2 octets of uninterrupted 1's (indicating the network ID part of the corresponding IP address), and 2 octets of uninterrupted 0's (indicating the host ID part of the corresponding IP address). Using the subnet mask, we can readily separate the IP address into its two parts:

```
Network ID:    10010010 10101000
Host ID:                         00000000 00000111
```

Although it is easiest to understand the role of the subnet mask while working in binary, binary notation is in general too cumbersome for humans. Hence IP addresses and subnet masks are usually written in *dotted-decimal* notation (Cunningham, 1998), in which each octet is converted to its equivalent decimal number, and the four decimal numbers are separated with dots (i.e., periods). The IP address and subnet mask from the example above would appear in dotted-decimal as:

```
IP Address:    146.168.0.7
Subnet Mask:   255.255.0.0

Network ID:    146.168
Host ID:       0.7
```

Note that the network ID is usually seen written as 4 octets (which can be created by appending any necessary 0 octets to the end of the network ID as delimited by the subnet mask), and that leading 0 octets are usually dropped from the host ID, as in:

```
Network ID:    146.168.0.0

Host ID:       7
```

It is also important to remember that the octet 00000000 in binary becomes 0 in dotted-decimal notation, and the octet 11111111 in binary becomes 255 in dotted-decimal notation.

Finally, we consider the basic concepts of subnetting. Imagine that your organization has obtained an official "public" network address from your Internet Service Provider (ISP) for your organization to use on the Public Internet. You could equally well imagine that your organization has chosen a "private" IP address to use for an internal TCP/IP network that will not be connected to the Public Internet (i.e., an *intranet*). In either scenario, you have the same problem: Your organization has enough hosts that they cannot, for a variety of reasons beyond the scope of this paper, coexist on the same TCP/IP network (Craft, 1998). The network must be broken up (or *segmented*) into separate subnetworks. Reasons to segment a large network may include such things as: reducing the size of collision domains, reducing the size of broadcast domains, implementing security, organizing subnetworks to reflect corporate structures, joining networks across WAN links, and segmenting network administration responsibilities (Bulette, 1998).

To segment the original network, we must devise an addressing scheme that is able to identify each *sub*network within the (original) larger network. This will require the use of an additional *subnet ID* along with the original *network ID*. A Given host will then be uniquely identified by the combination of:

1. A *network ID* that uniquely specifies the network on which the host resides (if the network is on the public Internet, this network ID is the address that will identify the network (including all its subnets) on the public Internet)

2. A *subnet ID* that uniquely specifies the subnetwork (within the original network in item 1 above) on which the host resides

3. A *host ID* that uniquely specifies the host on the subnetwork in item 2 above

An IP address already accommodates a network ID and a host ID, so all that is required is some way to create the subnet ID field. Since we can't expand the size of the IP address (32 bits for IPv4), we most "borrow" some bits from the existing address to use for the subnet ID. We can't borrow bits from the network ID part of the IP address because this has been pre-assigned by our ISP to uniquely identify our organization's network. Changing the network ID would wreck our ISP's assignments of network ID's to the ISP's customers. Hence we are forced to borrow bits to create the subnet ID from the existing host ID field.

The process of borrowing bits from the host ID field to form a new subnet ID field is known as *subnetting*. The process is shown in Table 2 below:

Table 2

Network ID	3 Bits Borrowed for Subnet ID	Host ID (3 bits Shorter)
`10010010 10101000`	`000`	`00000 00000111`

Notice that when we "borrow" bits from the host ID for the subnet ID, the original subnet mask is no longer accurate. As shown in Table 3 below, the original subnet mask has binary 0's matching up with the bits in the new Subnet ID. Since binary 0's in the subnet mask indicate the Host ID field, the newly created "Subnet ID" field still appears to belong to the original Host ID field.

Table 3

	Network ID	3 Bits Borrowed for Subnet ID	Host ID (3 Bits Shorter)
IP Address:	`10010010 10101000`	`000`	`00000 00000111`
Original Subnet Mask:	`11111111 11111111`	**`000`** `(0 bits here make subnet ID appear to be part of host ID)`	`00000 00000000`

To eliminate confusion over what bits still belong to the original Host ID field and what bits belong to the new Subnet ID field, we must *extend* the binary 1's in the original Subnet Mask with enough 1 bits to match the size of the newly created Subnet ID field (and correspondingly *reduce* the number of 0's which originally identified the Host ID in the Subnet Mask by the same amount). The new subnet mask is called a *custom subnet mask*. After this adjustment, the total number of bits in the *custom subnet mask* will still be 32, but the number of binary 1's will have increased by the size of the subnet ID, and the number of binary 0's will have decreased accordingly. This operation is illustrated in Table 4 below:

Table 4

	Network ID	**Bits Borrowed for Subnet ID**	**Shortened Host ID**
IP Address:	`10010010 10101000`	`000`	`00000 00000111`
Original Subnet Mask:	`11111111 11111111`	`000`	`00000 00000000`
Custom Subnet Mask	`11111111 11111111`	**`111`** `(1 bits here indicate field belongs to network ID)`	`00000 00000000`

A critical issue when “borrowing” bits from the host ID to create the subnet ID is to accurately determine the following information:

1. How many subnets are needed
2. How many bits must be “borrowed” from the host ID field for the new subnet ID field to accommodate the required number of subnets
3. What is the largest number of hosts that will ever be on a given subnet
4. How many bits must be retained in the host ID field to accommodate the maximum number of hosts needed

These considerations mandate that careful planning should be carried out *before* the subnetting process is begun. It is obviously prudent to plan for future as well as for current needs. Once pre-planning is complete, the actual subnetting process involves the following steps:

1. Determine how many subnets are needed
2. Determine the maximum number of hosts that will be on any given subnet
3. Determine how many bits to borrow from the host ID field for the subnet ID field
4. Determine how many bits must remain in the host ID field (and therefore cannot be borrowed for the subnet ID)
5. Determine how many bits are in the original network ID and host ID fields
6. Check to ensure that the number of bits to be “borrowed” from the host ID does not exceed the number of bits to be retained for the host ID (i.e., check that the subnetting problem is solvable)
7. Set an optimal length for the subnet ID field, including room for future growth

8. Create a modified (*custom*) subnet mask for the network
9. Determine the valid subnet ID’s for the network
10. Determine the valid ranges of IP addresses for each subnet on the network

A Simple Subnetting Procedure

The following step-by-step procedure can be used to subnet a TCP/IP network. It is assumed that the reader is already familiar with IP addressing, subnet masks, the separation of an IP address into a network ID and a host ID, and basic subnet concepts as discussed above.

1. **Determine the required number of subnets and call it S (“big S”)**

   When estimating the required number of subnets, it is critical to consider not only your current subnet needs, but also to plan for future growth. If there is historical information available, use it as a guide to predict how many subnets will be needed next year, two years from now, three, etc. Remember to include both current and anticipated subnetworks in your total. If your WAN links are handled as separate subnets, then count each WAN link too.

   Remember to leave plenty of room for growth. Nothing is more embarrassing (and costly) than to have to redo a network design because of poor planning. Work closely with users and management to uncover upcoming changes that might affect future growth in ways not shown by historical data (such as an anticipated merger or acquisition, introduction of a new product line, etc.). In the steps that follow, assume that S = 5.

2. **Determine the maximum number of hosts per subnet and call it H (“big H”)**

   In TCP/IP terminology, a “host” is any device that attaches to the network via a network interface. The count should reflect the largest number of network interfaces that will ever be on a given subnet. Remember to include not only network interfaces for computers, but also any network interfaces in printers, routers, or any other networked devices. Some computers (called *multihomed hosts*) may have more than one NIC, in which case each NIC is counted separately. Other devices (such as bridges or routers) will also have more than one network interface.

   It is not necessary to tabulate the number of network interfaces for every subnet. The purpose of step 2 is to count the *maximum* number of interfaces that will ever be needed for the *largest* subnet.

   As with step 1, remember to plan for growth. Use historical data if available, but also look for upcoming changes that could lead to significant growth not reflected in the historical data. Again, nothing is more embarrassing than to have to redo a network design because of poor planning. In the steps that follow, assume that H = 50.

3. **Find the smallest integer s ("little s") such that $2^s - 2 \geq S$**

This step calculates the number of bits s ("little s") needed in the IP address for the subnet ID's. "Little s" is the smallest integer (whole number) such that $2^s - 2$ is at least S ("big S", the required number of subnets). As the following table illustrates, with s bits for the subnet ID, we can address 2^s different subnets. However, a subnet ID is not allowed to be either all 0's (which according to TCP/IP standards always means the *current subnet* and therefore cannot be used as a subnet ID for an actual subnet), or all 1's (which according to TCP/IP standards is always a *broadcast* address and therefore cannot be used as a subnet ID for an actual subnet) (Heywood, 1997). Hence with s bits for the subnet ID, the effective number of addressable subnets is $2^s - 2$, as shown in Table 5 below:

Table 5

s = number of bits for subnet ID	**$2^s - 2$ = number of addressable subnets**	**Valid subnet addresses (all 0s or all 1's are invalid and are shown crossed out)**
1 (produces no valid addresses)	$2^1 - 2 = 2 - 2 = 0$	~~0~~ ~~1~~
2	$2^2 - 2 = 4 - 2 = 2$	~~00~~ 01 10 ~~11~~
3	$2^3 - 2 = 8 - 2 = 6$	~~000~~ 001 010 011 100 101 110 ~~111~~

Calculating s is easier if we rewrite the inequality $2^s - 2 \geq S$ as $2^s \geq S + 2$. If you are comfortable with the entries in Table 1, you can quickly find the smallest s such that $2^s \geq S + 2$ as follows:

1. Find the smallest integer value in the right-hand column of Table 1 that is equal to or greater than S + 2 ("big S" plus 2)

2. Using Table 1, find the corresponding value of s ("little s") from the left-hand column

3. This is the number of bits needed for the subnet ID's in your IP addresses

For example, if S = 5, the smallest integer value from the right-hand column of Table 1 that is $\geq 5 + 2 = 7$ is 8. The corresponding value of s (from the left-hand column of Table 1) is 3. If on the other hand S = 7, the smallest integer value from the right-hand column of Table 1 that is greater than or equal to 7 + 2 = 9 is 16. Hence the value of s is 4.

4. Find the smallest integer h ("little h") such that $2^h - 2 \geq H$

This step calculates the number of bits h ("little h") needed in the IP address for the host ID's, and is similar to step 3. In fact, the TCP/IP standards state that a host ID cannot be all 0's, since a 0 host ID always refers to the *current host* (and so can never be used as the address for a particular host), or all 1's, since all 1's indicates a *broadcast* address (and so can never be used for the address of a particular host) (Heywood, 1997). Hence the formula for finding the number of bits needed for the host ID's is exactly parallel to that used to calculate the number of bits needed for the subnet ID's. Similar to step 3, we look for the smallest integer h such that $2^h - 2 \geq H$. By rewriting the inequality as $2^h \geq H + 2$, we can use a parallel procedure to that in step 3:

1. Find the smallest integer value in the right-hand column of Table 1 that is equal to or greater than H + 2 ("big H" plus 2)

2. Using Table 1, find the corresponding value of h ("little h") from the left-hand column

3. This is the number of bits needed for the host ID's in your IP addresses

For example, if H = 50, the smallest integer value from the right-hand column of Table 1 that is $\geq 50 + 2 = 52$ is 64. The corresponding value of h (from the left-hand column of Table 1) is 6. If on the other hand H = 30, the smallest integer from the right-hand column of Table 1 that is greater than or equal to H + 2 = 30 + 2 = 32 is 32. Hence h = 5.

5. Determine the total number of host ID bits in the *standard* subnet mask for your assigned address class. Call the number of host ID bits T ("big T").

Table 6 below shows the standard number of Host ID bits for each of the three major address classes, A, B, and C. To determine the *address class* of a network ID, look at the first octet in dotted-decimal notation. As Table 6 shows, if the first octet is between 1 – 126, the network is Class A; if the first octet is between 128 – 191, the network is Class B; if the first octet is between 192 – 223, the network is Class C. Once you know the address class, it is easy to determine the number of host ID bits from Table 6. For example, if you have been officially assigned a Class B address, T = 16; if you have been officially assigned a Class C address, T = 8; etc. To do mental subnetting, it is helpful to memorize Table 6.

Table 6

Address Class	Starting Octet for Network ID (in decimal)	Network ID Bits in Standard Subnet Mask	Host ID Bits in Standard Subnet Mask (T)
A	1 – 126	8	24
B	128 – 191	16	16
C	192 - 223	24	8

For example, assume the official network ID is 146.168.0.0. According to Table 6, this is a Class B address (since 146 is between 128 – 191, inclusive) and the number of host ID bits in the standard subnet mask is T = 16.

6. **If s + h > T, then you need more host ID bits than are available for your official address class. You cannot meet your subnetting requirements (i.e., the problem is not solvable using your assigned address class)**

   In this case your officially assigned Network ID requires so many bits in the IP address that there are not enough bits left over to satisfy your needs for subnet ID's and host ID's. In short, your subnetting requirements S (”big S”) and H (“big H”) when taken together are too large for your assigned address class. You will either have to reduce the values of S and/or H, or apply for an official network ID that requires fewer network ID bits and leaves more host ID bits (i.e., change from class C to class B, or from class B to class A). If s + h > T, start over again at step 1 after changing your requirements (i.e., the estimated number of subnets and/or the maximum required number of hosts per subnet), and/or changing your officially assigned address class.

   In our example, T = 16, s = 3, and h = 6. Hence s + h = 3 + 6 = 9, which is *not* greater than T = 16. Hence we can proceed to the next step.

7. **If s + h = T, skip the next step (step 8)**

   You have exactly enough bits for the desired subnetting. Skip step 8 and go on to step 9. In our example, s + h = 3 + 6 = 9 and T = 16, so we must carry out step 8.

8. If s + h < T, then you have T – s – h "extra" bits to distribute between s and h in a manner that best provides for unanticipated future growth. Calculate r ("little r") as r = T – s – h, the number of bits that you can deploy either for extra subnet ID bits and/or for extra host ID bits, then increase s and/or h accordingly

You have r ("little r") bits to distribute between s ("little s", the number of bits needed for subnet ID's) and h ("little h", the number of bits needed for host ID's). Increase s and/or h until you've used up all r bits (at which point T = s + h, as in step 7).

Since you are more likely to run out of subnets before you run out of host ID's on any given subnet, it is probably safer to make sure that s is comfortably large before increasing h. Assume that at the beginning of step 8, T = 16, s = 7, and h = 6. You then have T – s – h = 16 – 7 – 6 = 3 bits to distribute between s and h. Since it is probably a safer hedge to increase s, you might give 2 of the 3 "extra" bits to s (making s = 9), and give 1 of the 3 extra bits to h (making h = 7). If done correctly, the new values of s and h will sum to T (9 + 7 = 16).

In the example we've been following from previous steps, r = T – s – h = 16 – 3 – 6 = 7 bits to distribute between s and h. We will increase s by 1 (making the new value of s = 4), and increase h by 6 (making the new value of h = 12). Since it is often more important to increase s than to increase h, we should carefully question our decision to increase s by only one. Let us assume that we have a very high degree of confidence in our original estimate for S, but are less sure of our original estimate for H. Hence we (perhaps atypically) decide to favor h over s in this step.

9. Determine the custom subnet mask for your network

Start with the default (standard) subnet mask for your address class as shown in Table 7 below: We will extend the network ID portion of the default subnet mask by replacing its leftmost zero octet (shown bolded in the table) with a new value.

Table 7

Address Class	Default Subnet Mask	Leftmost Zero Octet
A	255.0.0.0	255.**0**.0.0
B	255.255.0.0	255.255.**0**.0
C	255.255.255.0	255.255.255.**0**

Calculate the new value for the leftmost zero octet in the standard subnet mask as:

$$256 - 2^{8 - s}$$

For example, if the adjusted value of s is 4, we calculate $256 – 2^{8 – 4} = 256 – 2^4 = 256 – 16 = 240$. This value will replace the leftmost zero octet in the default subnet mask for our network class, thus forming the custom subnet mask. Since in Step 5 we determined that our network ID was Class B, our default subnet mask from Table 7 is 255.255.**0**.0. Replace the leftmost zero octet (shown bolded) with the value 240 to obtain the *custom subnet mask* 255.255.240.0.

10. Determine the Valid Network ID's for the New Subnets

The next step is to determine the network (and subnetwork) ID's for the new subnets. Start by identifying the leftmost 0 octet in the original network ID for your network (expressed as four octets in dotted-decimal). This is the octet that corresponds to the leftmost 0 octet in the standard subnet mask (i.e., the octet shown bolded in Table 7). For the original subnet mask in our example, it would be the third octet from the left (shown bolded): 146.168.**0**.0. For a Class A network, this will always be the second octet (as in 13.**0**.0.0), for a class B network, this will always be the third octet (as in 146.168.**0**.0), and for a Class C network, this will always be the fourth octet (as in 193.200.17.**0**).

Note this particular octet will always have all 0's in the extended subnet ID area (the area "borrowed" from the original host ID), and so is *not* a valid subnetwork ID (recall that a zero value is not permitted for either a network or subnetwork ID).

To obtain the first valid subnetwork ID, add $2^{8 – s}$ to the leftmost 0 octet (as identified above) in the *original* network address. Now add $2^{8 – s}$ to the same octet in the first subnetwork ID to get the second subnetwork ID, add $2^{8 – s}$ to the same octet in the second to get the third, etc. Continue in this fashion until you have obtained $2^s – 2$ subnetwork ID's, or until you reach the value of your custom subnet mask. Note that the custom subnet mask value itself is not a valid network ID because the subnet ID is all 1's (the reserved broadcast address).

In our example, the original network ID is 146.168.**0**.0 (the leftmost zero octet is shown bolded), the updated value of s is 4, and $2^{8 – s} = 2^{8 – 4} = 2^4 = 16$. We expect $2^s – 2 = 2^4 – 2 = 16 – 2 = 14$ subnets, which we find as follows:

The first network ID is obtained by adding $2^{8 – s}$ (i.e., 16) to the leftmost 0 octet in the original network address, forming the first network ID, i.e., add 16 to the third octet (shown bolded) in 146.168.**0**.0 to yield

146.168.**16**.0 (first valid subnet ID)

The second subnet ID is obtained by adding $2^{8 – s}$ (16) to the same octet in the first valid subnet ID (shown bolded above), i.e., add 16 to the third octet (shown bolded) in 146.168.**16**.0 to yield

146.168.**32**.0

To form the third network ID, again add 2^{8-s} (16) to the same octet in the second valid subnet ID (shown bolded above), i.e., add 16 to the bolded octet in 146.168.**32**.0 to yield

146.168.**48**.0

Repeat this procedure until you have obtained the expected 14 subnetwork addresses (or until you reach the custom subnet mask from Step 9). The results are shown in Table 8 below:

Table 8

Original Network ID (Not a valid subnetwork address)	146.168.**0**.0
Network ID for Subnet 1	146.168.**16**.0
Network ID for Subnet 2	146.168.**32**.0
Network ID for Subnet 3	146.168.**48**.0
Network ID for Subnet 4	146.168.**64**.0
Network ID for Subnet 5	146.168.**80**.0
Network ID for Subnet 6	146.168.**96**.0
Network ID for Subnet 7	146.168.**112**.0
Network ID for Subnet 8	146.168.**128**.0
Network ID for Subnet 9	146.168.**144**.0
Network ID for Subnet 10	146.168.**160**.0
Network ID for Subnet 11	146.168.**176**.0
Network ID for Subnet 12	146.168.**192**.0
Network ID for Subnet 13	146.168.**208**.0
Network ID for Subnet 14	146.168.**224**.0
Custom Subnet Mask value (Not a valid subnetwork address)	146.168.**240**.0

11. Determine the Valid IP Addresses for Each Subnet

The final step in subnetting is to determine the valid IP addresses for each new subnetwork. To generate the valid IP addresses for a given subnetwork, start with that subnetwork's network address (as shown in Table 8, for example). Add 1 to the rightmost octet in the subnet address to obtain the first valid IP address on that subnet. In our example:

Network ID of first subnet:	146.168.16.**0**
First valid IP address on that subnet:	146.168.16.**1**

Continue to add 1 to the rightmost octet until one of the following three conditions occurs:

1. The octet that you are incrementing reaches 255. When incrementing the value 255, instead of adding 1 (to get 256), roll the 255 back to 0 and add 1 to the next octet to the left. This operation is similar to a *carry* in ordinary decimal addition. For example, assume you have just added 1 to 146.168.16.**254** to obtain 146.168.16.**255**. The next step would *not* be to add 1 again to obtain 146.168.16.**256** (which is not a valid IP address). Instead, roll the 255 back to 0 and add 1 to the next octet to the left (the 16), yielding 146.168.**17.0**. From this point, continue to increment as before to obtain additional IP addresses for the current subnet

2. While incrementing, you get to the point where another increment would reach one less than the network ID for the next subnet. In this case, you have listed all the valid IP addresses for the current subnet, and you must move on to the next subnet (by starting with its network ID and repeatedly incrementing the rightmost octet by 1)

3. You reach a total of $2^h - 2$ IP addresses for a given subnet. This is equivalent to condition 2 above, and in fact is just another way of looking at the same situation. As in condition 2, you have listed all the valid IP addresses for the current subnet. Move on to the next subnet by starting with its network ID and repeatedly incrementing by 1

Repeat this process for all subnetworks to obtain a complete list of valid IP addresses for each subnet.

In our example, we start with 146.168.16.0, the network ID for the first subnet (see Table 8). Add 1 to the rightmost octet to obtain the first valid IP address for this subnet, namely 146.168.16.1. Again, add 1 to the rightmost octet to obtain the second valid IP address for this subnet, namely 146.168.16.2. Continue in this fashion until reaching 146.168.16.254, which after incrementing yields an IP address of 146.168.16.255. Note that this *is* a valid IP address on the subnet. The next valid IP address is found by rolling the 255 back to 0 and incrementing the next octet to the left, yielding 146.168.17.0. Continue incrementing until reaching 146.168.17.255, which is followed by 146.168.18.0. Again, the process repeats until we hit 146.168.18.255, which is followed by 146.168.19.0. This process will continue all the way to 146.168.30.255, which is followed by 146.168.31.0. We continue to increment until reaching 146.168.31.254. We are now at the point where yet another increment would yield one less than the next subnet's network ID (i.e., if we were to carry out one more increment we would be at 146.168.31.255, which if it were itself incremented would yield the subnet ID for the next subnet, 146.168.32.0). At this point we have a complete list of all valid IP addresses for the first subnet. We would then have to repeat the entire process for the second subnet, etc. Table 9 summarizes the IP addresses for the first subnet:

Table 9

IP Addresses for Subnet 1 (Network Address 146.168.16.0)
146.168.16.1 to 146.168.16.255
146.168.17.0 to 146.168.17.255
146.168.18.0 to 146.168.18.255
146.168.19.0 to 146.168.19.255
146.168.20.0 to 146.168.20.255
146.168.21.0 to 146.168.21.255
…
146.168.30.0 to 146.168.30.255
146.168.31.0 to 146.168.31.254

Do not be confused by the fact that some valid IP addresses end in 0 or 255. This happens normally when subnetting, and the rules about not having network, subnetwork, or host ID's equal to all 0's or all 1's are not necessarily violated just because an *octet* is equal to all 0's or all 1's. The rules place restrictions on the values of network, subnetwork, and host ID's, *not* on the values of octets. To understand this, consider the IP address 146.168.17.0 from Table 9 and analyze it according to the custom subnet mask for our example network, 255.255.240.0.

Table 10

	Standard Network ID Part of IP Address	**Bits Borrowed from Host ID to form Subnet ID Part of IP Address**	**Shortened Host ID Part of IP Address**
IP Address 146.168.17.0	10010010 10101000	0001	000**1** 00000000
Custom Subnet Mask 255.255.240.0	11111111 11111111	1111	0000 00000000

Notice that although the rightmost octet of the Host ID consists of all zero bits, the full Host ID is a total of 12 bits and is *not* all 0's (the sole one bit is shown bolded).

For a second example, consider the IP address 146.168.21.255 from Table 9. Although the last octet is 255 (eight 1's in binary), the following analysis shows that the full host ID is *not* all 1 bits (the two zero bits in the host ID are shown bolded):

Table 11

	Standard Network ID Part of IP Address	**Bits Borrowed from Host ID to form Subnet ID Part of IP Address**	**Shortened Host ID Part of IP Address**
IP Address 146.168.21.255	10010010 10101000	0001	**0**1**0**1 11111111
Custom Subnet Mask 255.255.240.0	11111111 11111111	1111	0000 00000000

A Class C Example

Suppose an ISP assigns a Class C network address of 193.200.35.0 to an organization (call it Widgets, Inc., or just Winc). We will work through the 11 steps presented above in order to subnet this Class C network.

1. After meetings with relevant Winc personnel and study of historical growth trends, it is determined that Winc currently needs 2 subnets, with practically no likelihood of adding other subnets in the future. Therefore, we set S (“big S”) at 2.

2. After meetings with relevant Winc personnel and study of historical growth trends, it is determined that Winc currently needs at most 25 hosts on any subnet. In the future, subnet size is not expected to pass 30 hosts. Hence, we set H (“big H”) at 30.

3. To find the smallest integer s such that $2^s - 2 \geq S$, we first rewrite the inequality as $2^s \geq S + 2$. Since S = 2, this becomes $2^s \geq 2 + 2$ or $2^s \geq 4$. Reference to Table 1 shows that the smallest such integer s is 2.

4. To find the smallest integer h such that $2^h - 2 \geq H$, we first rewrite the inequality as $2^h \geq H + 2$. Since H = 30, this becomes $2^h \geq 30 + 2$ or $2^h \geq 32$. Reference to Table 1 shows that the smallest such integer h is 5.

5. Winc’s assigned network address is 193.200.35.0, which begins with 193. Hence Winc has a Class C network address for which T (“big T”) is 8 (see Table 6).

6. Now we can calculate s + h = 2 + 5 = 7, which does not exceed the value of “big T” (T = 8). Hence we have a solvable subnetting problem and can proceed to step 7.

7. Since s + h = 2 + 5 = 7 which is not equal to 8 (the value of T), we must carry out step 8

8. Since s + h = 2 + 5 = 7 is less than T = 8, we have r = T – s – h = 8 – 2 – 5 = 1 bit left over to increase the value of either s or h. Since in general Winc is more likely to run short of subnets rather than hosts on a subnet, we allocate the extra bit to s, incrementing s so that now s = 3. Note that now s + h = 3 + 5 = 8 = T.

9. To determine the custom subnet mask for Winc’s network, we start with the standard (default) subnet mask for Class C (Winc’s network class), which according to Table 7 is 255.255.255.0. We will replace the leftmost zero octet in the original subnet mask (i.e., the 0 in 255.255.255.**0**), with a new octet that will extend the subnetwork ID into the host ID. Calculate the new value for the original leftmost zero octet as $256 - 2^{8-s}$, which is $256 - 2^{8-3}$ or $256 - 2^5$ or 256 – 32 or 224. Hence the custom subnet mask for Winc’s network is 255.255.255.224.

10. Now we determine the valid network ID’s for the new subnets by identifying the leftmost 0 octet in the original network ID assigned by the ISP. Since this network ID is 193.200.35.**0**, the leftmost 0 octet (the only 0 octet) is also the rightmost 0 octet (shown bolded). We now add $2^{8-s} = 2^{8-3} = 2^5 = 32$ to this 0 octet to get the new value for the octet in the first subnet ID: 32 + 0 = 32. Thus the network ID for the first subnet is

193.200.35.**32**

We continue adding 2^{8-s} to this octet until we either reach the value of the custom subnet mask (255.255.255.224) or until we have network addresses for $2^s - 2$ subnets (these two conditions are equivalent and so will occur at the same time). In our case, $2^s - 2 = 2^3 - 2 = 8 - 2 = 6$ subnets, so we continue adding 2^{8-s} five more times (for a total of six times) as shown below:

Table 12

Original Network ID (not a valid subnet address since subnet ID is all 0's)	193.200.35.**0**
Address for subnet 1	193.200.35.**32**
Address for subnet 2	193.200.35.**64**
Address for subnet 3	193.200.35.**96**
Address for subnet 4	193.200.35.**128**
Address for subnet 5	193.200.35.**160**
Address for subnet 6	193.200.35.**192**
Custom Subnet Mask (not a valid subnet address since subnet ID is all 1's)	193.200.35.**224**

11. To determine the valid IP addresses for each subnet, we begin with the network ID for the subnet. Let us start with the first subnet whose address (as seen from Table 12) is 193.200.35.32. To find the first IP address on the subnet, we add 1 to the rightmost octet of the subnet address: 32 + 1 = 33. Thus the first IP address on subnet 1 is

    193.200.35.33

    We will continue incrementing until we reach 255, or until the next increment would reach two less than the next subnet address, or until we have generated $2^h - 2$ IP addresses (these last two conditions are equivalent and will always occur at the same time). Since in our case h = 5, we can expect $2^5 - 2 = 32 - 2 = 30$ IP addresses per subnet. The valid IP addresses for subnet 1 are shown in the following table:

Subnet 1 Address #	IP Address
1	193.200.35.33
2	193.200.35.34
3	193.200.35.35

Subnet 1 Address #	IP Address
4	193.200.35.36
5	193.200.35.37
6	193.200.35.38
7	193.200.35.39
8	193.200.35.40
9	193.200.35.41
10	193.200.35.42
11	193.200.35.43
12	193.200.35.44
13	193.200.35.45
14	193.200.35.46
15	193.200.35.47
16	193.200.35.48
17	193.200.35.49
18	193.200.35.50
19	193.200.35.51
20	193.200.35.52
21	193.200.35.53
22	193.200.35.54
23	193.200.35.55
24	193.200.35.56
25	193.200.35.57
26	193.200.35.58
27	193.200.35.59
28	193.200.35.60
29	193.200.35.61
30	193.200.35.62

Note that if we increment the last octet of the 30th IP address (see Table 12), we get 63, which is one less than the network ID for the next subnet. Hence 193.200.35.62 is indeed the final IP address on subnet 1.

The IP addresses for the remaining 5 subnets can be found in a similar manner.

References

Bulette, G. *TCP/IP MCSE Study Guide*, Foster City, CA, IDG Books Worldwide, 1998.

Craft, M., Mayo, B., Pherson, J., et. al., *CCNA Cisco Certified Network Associate Study Guide (Exam 640-407)*, Berkeley, CA, Osborne McGraw-Hill, 1998

Cunningham, S., Frederick, B., First, T., et. al., *MCSE Microsoft TCP/IP on Windows NT 4.0 Study Guide*, Berkeley, CA, Osborne McGraw-Hill, 1998

Dulaney, E., Sherwood, L., Scrimger, R., *MCSE Training Guide TCP/IP*, Indianapolis, IN, New Riders Publishing, 1998.

Feit, S. *TCP/IP*, New York, NY, McGraw-Hill, 1997.

Heywood, D., Scrimger, R., *Networking with Microsoft TCP/IP Certified Administrator's Resource Edition*, Indianapolis, IN, New Riders Publishing, 1997.

Lammle, T., *CCNA Cisco Certified Network Associate Study Guide (Exam 640-507)*, 2nd ed., Alameda, CA, SYBEX, 2000

Loshin, P. *TCP/IP Clearly Explained*, San Diego, CA, Academic Press, 1997.

Odom, W., *CCNA Exam Certification Guide*, Indianapolis, IN, Cisco Press, 1999

Odom, W., *Cisco CCNA Exam #640-507 Certification Guide*, Indianapolis, IN, Cisco Press, 2000

Poplar, M., Waters, J., McNutt, S., and Stabenaw, D., *CCNA Routing and Switching*, Scottsdale, AR, Coriolis, 2000