

A Six-Question Approach to Subnetting

Introduction

Overview

Understanding What Subnetting Actually Does

- 1) How Many Sub-Networks Do You Need?
- 2) How Many Bits Did You Have To Use?
- 3) What Is Your Subnet Mask?
- 4) What Is Your Block-Size?
- 5) What Are Your Subnets?
- 6) What Are The Number Of Hosts And IP Ranges For Each Subnet?

Subnetting Practice Problems Using The 6 Question Approach

Variable Length Subnet Making (VLSM)

VLSM Practice Problems Using The 6 Question Approach

Supernetting And Classless Interdomain Name Routing (CIDR)

Introduction

If you've read through countless papers on TCP/IP and understand most of what you've read and yet subnetting still remains an enigmatic process, then this paper is for you. After reading this article, you should be able to do subnetting mostly in your head. Subnetting isn't really that difficult, the problem lies with authors who write cryptically difficult to understand articles on how to go about tackling the process. **The purpose of this paper is to simplify the subnetting process, nothing more.**

In your career you may find yourself only subnetting a network a few times or never. However if you have aspirations of becoming network or systems engineer you need to know how to subnet, and of course you will need to know it for many of the various certifications available today.

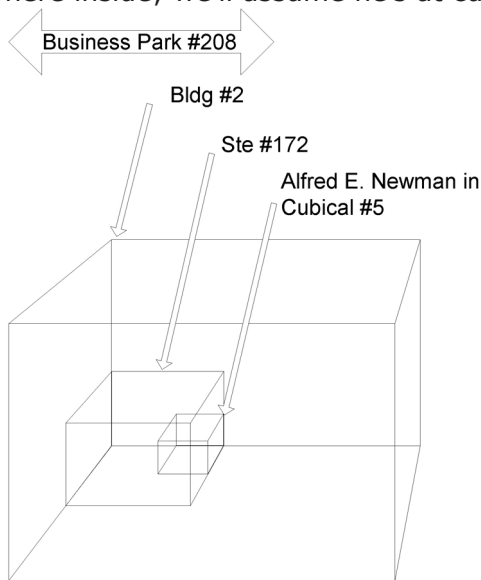
Understanding What Subnetting Actually Does

What makes subnetting seem difficult to understand is its not physically tangible; all too often it's presented as a mathematical equation "you just have to learn". We will first start by using a real-world example to make the process seem more tangible and expose what TCP/IP subnetting really does at a conceptual level. A business park is a great physical example of how subnetting works since it's simply a subdivision of a large physical area.

Let's assume you need to send a letter to Alfred E. Newman at the following business park address:

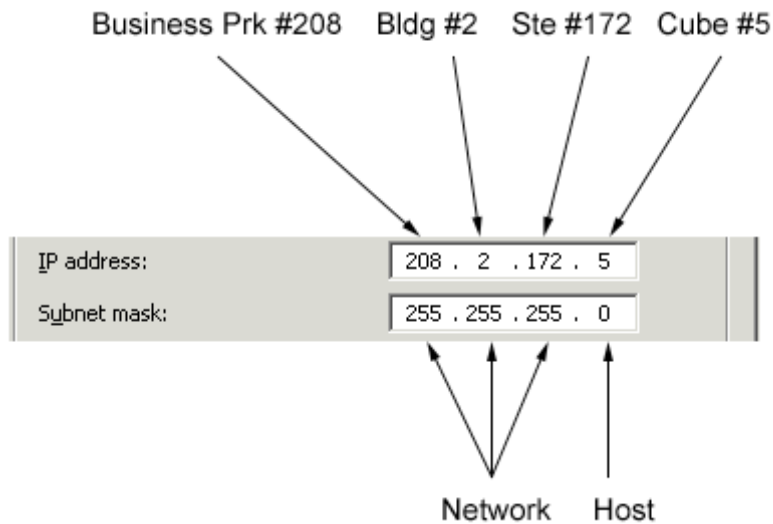
Attn: Alfred E. Newman
Mad Magazine
208 Mad St.
Bldg #2, Ste #172
Mad City, CA 94563

Were sending mail to a **network** of buildings at 208 Mad Street, to a **network** of offices in building #2, to a **network** of smaller offices or cubicles in suite #172 until our letter reaches the **host** Alfred E. Newman somewhere inside; we'll assume he's at cubical 5 writing the next issue of MAD magazine. (Figure 1)



(Figure 1)

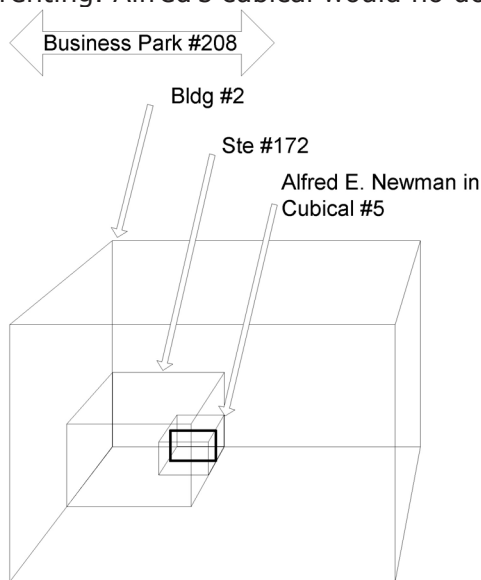
If we were to duplicate this example as it relates to a TCP/IP address on your Network Interface Card (NIC) in Windows it would appear as follows:



(Figure 2)

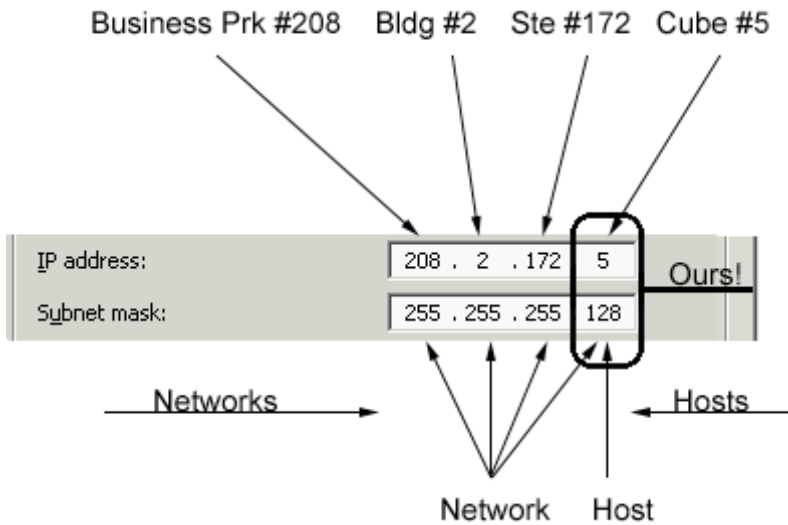
This brings us to an important point of understanding subnetting; even though there are no physical buildings, walls, or offices in the binary world of TCP/IP, conceptually the process of segmenting computers into manageable groups so they can operate is very similar to the process of segmenting work spaces and people into manageable groups in which they can operate. The difference is we are using 32 electronic bits to identify workspaces (networks) and then ultimately a host to communicate with, typically a computer.

But what happens when you need to split your office into two offices? Looking at next image you can see if we wanted to divide Ste #172 we would have work within Ste #172 by adding an additional wall lest we want to upset the building and our next-door neighbors by tearing down the wall to another suite we aren't renting. Alfred's cubical would no doubt end up in one of the now two resulting offices also.



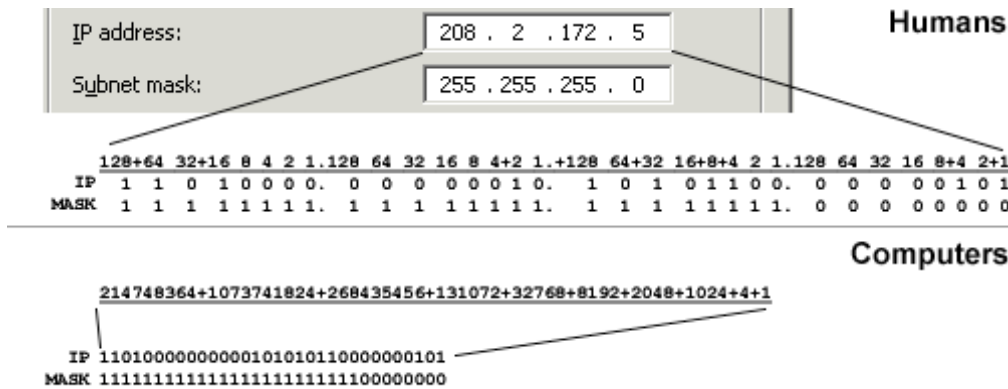
(Figure 3)

This concept holds true with TCP/IP also, we have been handed the keys to the 172 network, inside our parent network of 2, which is also inside its parent network of 208. If we want to split our 172 network into two subnetworks then we will have to take space from within the 4th octet similar to our previous workspace example. We don't get to modify our parent networks, our 172 is just a small fry in the grand scheme of things, this is why networks typically push to the right and hosts push from the left.



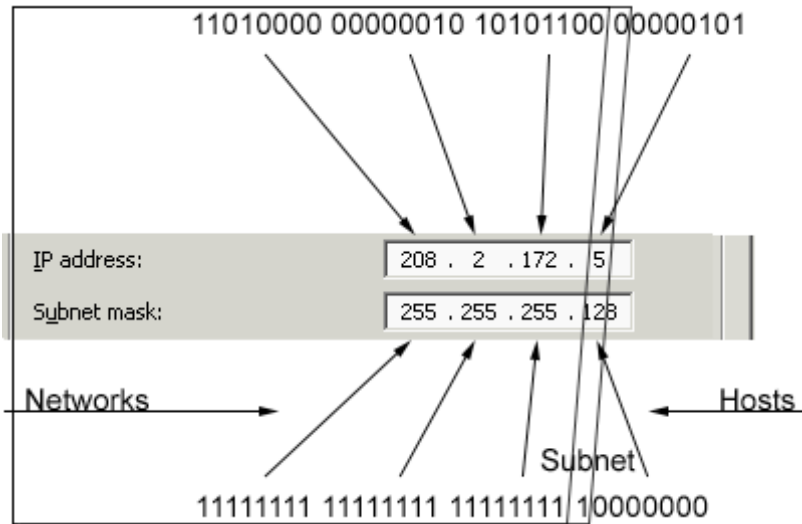
(Figure 4)

Most people at this point begin to ask how we came up with a mask of 128, as you will see later all we did was put up a virtual wall in the middle of the room. Much of the confusion surrounding subnetting lye's with classfull IP addressing and dotted decimal notation. In a perfect world classfull IP addressing is simple and elegant, we know to type 255 to define the network and 0 to define the host to define the subnet mask. Then the Internet ran out of IP addresses and every bit suddenly counted. A computer doesn't care about convenient little spaces we use to type in an IP address. It simply processes the entire 32-bit IP address against the subnet mask.



(Figure 5)

As shown in (Figure 5) dotted decimal notation is for our viewing pleasure, it's easier to remember and work with four small numbers such as 208.2.172.5 vs. trying to figure out or remember the binary equivalent of 3,489,836,037 which is what the computer does. This also graphically shows how the subnet mask is used to determine where the network stops and the hosts start. To truly understand how subnetting works you need to see binary the same way a computer does. To complete our example we will split our 172 network by pushing the mask to the right by 1-bit creating two new subnetworks.



(Figure 6)

Don't worry about the math for now, simply notice how the mask moved right by one bit and intrudes on our once pristine untainted octet of hosts changing our network mask of 0 to 128. Here lies the challenge of subnetting; we're left grappling with the idea that the 4th octet partly describes the network and the host. Look at the subnetted mask broken down into binary at the bottom of (Figure 6); as you can see we gained network real estate at the expense of losing space for our hosts. In the real world we would also lose a little space for a wall, it too consumes some space. Now that we have the concept down and know what to expect from a binary perspective let's do some subnetting.

1) How Many Sub-Networks Do You Need?

This is a simple decision made based on your company needs. By using the following six-step approach **we can use the answers from each question to help answer the next, so all you have to do is remember the 6 questions.** For our purpose here we can pull any number we want out of thin air, let's first start with our current example then we'll move onto a different one.

We needed two networks, question 1 answered!

2) How Many Bits Did You Have To Use?

Once you know how many subnets you need, then you just need to figure out how many bits it will take to accomplish the mission. Remember every binary bit is either on or off it has two possible values 0 or 1, this is where the powers of two come into play. No doubt you've read about the dreaded $2^n - 2$ formula for determining how many hosts you get ⁿ
($2^n - 2$ is no longer applied to networks, see NOTE: in step 6a)*

Most people (myself once included) don't realize you can use the same formula for determining your subnetworks; 2^n but remove the -2 from the formula. I take no credit for this approach it's been around for sometime, many writers have just convoluted this point. I've simply organized the process into six easy to remember steps. For the sake of those who don't understand how powers work and hate to admit they didn't pay attention in math class (again myself included) let's take a moment to reveal how powers actually work; don't worry we'll recap before we move on so we stay on track.

Of all the powers calculations powers of 2 is the easiest to understand. Breaking down 2^3 can be arrived at by doing the long math $2 \times 2 \times 2 = 8$ but it can also be arrived by simply doubling each resulting answer 2, 4, 8, the two's just have it like that. If you've ever had that classic conversation about how much money you'd have if you doubled a penny each day for 30 days. At the end of 30 days you would have \$5, 368,709.12 because each day the total amount you had from the day before doubles again growing exponentially. The same is true for binary except instead of starting with one penny we start with two pennies or to stay on topic, two bit values 0 and 1. So 1 binary bit has two possible unique values 0 and 1, to answer our second question it takes 1 bit to give us two subnets.

# needed	2	subnets						208.2.172.???
	128	64	32	16	8	4	2	1
Networks								Hosts
0 =	0							
128 =	1							

(Figure 7)

Look at the above example, we get a 0 subnet and a 128 subnet each capable of holding 128 values or hosts. Don't worry about the hosts for now just enjoy the simplicity of this step. By doubling the first available bit once we come up two possible values or subnets (remember networks push right). Then we simply ask ourselves how many times did I have to double? In this case only once, you could literally count out subnets on your fingers. You might be thinking this is too simple so let's do a few more. Your network might need four subnets so you simply start doubling from 2 and keep doubling until you reach four.

# needed	2	4	subnets						208.2.172.???
	128	64	32	16	8	4	2	1	
Networks								Hosts	
0 =	0	0							
64 =	0	1							
128 =	1	0							
192 =	1	1							

(Figure 8)

2, 4, there you have it! We needed four subnets and we doubled two times to get the desired result, so we need two bits. The above example shows this as well as all four unique subnets. We'll do one more before moving on to question three. If you needed six subnets we would double as follows: 2, 4, 8, done! We only needed six subnets but got some breathing room with two extra so you can expand if you need to, always a good thing.

# needed	2	4	8	subnets					208.2.172.???
	128	64	32	16	8	4	2	1	
Networks									Hosts
0 =	0	0	0						
32 =	0	0	1						
64 =	0	1	0						
96 =	0	1	1						
128 =	1	0	0						
160 =	1	0	1						
192 =	1	1	0						
224 =	1	1	1						

(Figure 9)

This brings us to important point, which is there are only 8 possible subnets in an octet before we're forced into the next octet. Once you get to the last subnet of 256 as shown below you're basically back to classfull IP addressing, if we used all eight bits then each subnet would have a value of one and your potential range is 0 through 255 as expected in a classfull IP addressing scheme.

# needed	2	4	8	16	32	64	128	256
	128	64	32	16	8	4	2	1
Networks	→							

(Figure 10)

The great part about this method to subnetting is if we needed more subnets for example one thousand (not that many would) we can still easily figure out how many bits are required to accomplish the mission.

# needed	2	4	8	16	32	64	128	256	512	1024		
	128	64	32	16	8	4	2	1	128	64	32	16
Networks	→											

(Figure 11)

As you can see we just invade the next octet. Powers of two just keep on working, remember octets are for our viewing pleasure, the computer however works with all thirty-two bits, it doesn't care about the period. By now you're probably asking...

3) What Is Your Subnet Mask?

You may have already started to figure out how to come up with the answer; by simply adding up the binary placeholders we used in the second step we learn what the mask is. In our first example from step two we needed four subnets and determined we had to use two bits (2, 4). Looking at the below figure you can see we just add up the placeholder values for both bits used to extend our 255.255.255.0 network, $128 + 64 = 192$, so 255.255.255.192 will be the new mask for all four subnets.

# needed	2	4	subnets		208.2.172.???				
	128	64	32	16	8	4	2	1	
Networks	→								← Hosts
0 =	0	0							
64 =	0	1							
128 =	1	0							
192 =	1	1							

$128 + 64 = 192$

(Figure 12)

Our second example from step two we needed six subnets but got eight it's the same process (2, 4, 8), we add the three bits used to get a mask of 255.255.255.224 for all eight subnets.

	# needed			subnets					208.2.172.???
	2	4	8	16	8	4	2	1	
Networks	128	64	32	16	8	4	2	1	
0 =	0	0	0						
32 =	0	0	1						
64 =	0	1	0						
96 =	0	1	1						
128 =	1	0	0						
160 =	1	0	1						
192 =	1	1	0						
224 =	1	1	1						

$128 + 64 + 32 = 224$

(Figure 13)

As said earlier there are only eight possible subnets on any octet before you are forced into the next octet.

128	=	128	
128 + 64	=	192	
128 + 64 + 32	=	224	
128 + 64 + 32 + 16	=	240	
128 + 64 + 32 + 16 + 8	=	248	(mental hint: 8 always equals 248)
128 + 64 + 32 + 16 + 8 + 4	=	252	
128 + 64 + 32 + 16 + 8 + 4 + 2	=	254	
128 + 64 + 32 + 16 + 8 + 4 + 2 + 1	=	255	

(Figure 14)

This brings us to another important point since there are only 8 possible network masks it doesn't matter what your starting IP address is the answer is the same regardless. If you needed 60 subnets we could do the math with just that piece of information alone.

1) How Many Sub-Networks Do You Need?

(2, 4, 8, 16, 32 64, Stop! You needed 60 but you got 64)

# needed	2	4	8	16	32	64
Networks	128	64	32	16	8	4

2) How Many Bits Did You Have To Use?

(We doubled 6 times so we used 6 bits)

bits used	1	2	3	4	5	6
# needed	2	4	8	16	32	64
Networks	128	64	32	16	8	4

3) What Is Your Subnet Mask?

(128 + 64 + 32 + 16 + 8 + 4 = 252 mask)

# needed	2	4	8	16	32	64
Networks	128	64	32	16	8	4
	= 252 Mask					

(Figure 15)

We got all that just from knowing we needed 60 subnets. Now we can apply that mask to any IP address range we want such as:

IP Range	172.31.0.0	(172.31/16)
Mask	255.255.0.0	

Changing it to:

IP Range	172.31.252.0	(172.31/22)
Mask	255.255.252.0	

Or we could apply the mask to:

IP Range	10.0.0.0	(10/8)
Mask	255.0.0.0	

Changing it to:

IP Range	10.252.0.0	(10/14)
Mask	255.252.0.0	

You might be wondering what the /16, /22, /8, and /14 are, these are common expressions used in the profession to show how many total bits are used to represent the network side of an IP address range. For example look at the masks for the following address ranges in binary counting the 1's.

Class A	11111111.00000000.00000000.00000000	/8
Class B	11111111.11111111.00000000.00000000	/16
Class C	11111111.11111111.11111111.00000000	/24
Classless IP	11111111.11111111.11111100.00000000	/22

(Figure 16)

Using the slash expression is an abbreviated method of expressing the complete network mask.

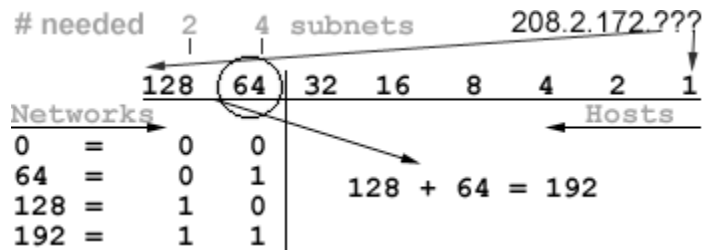
Class A	255.0.0.0	/8
Class B	255.255.0.0	/16
Class C	255.255.255.0	/24
Classless IP	255.255.252.0	/22

(Figure 17)

Most of the time though you will see this expression applied to actual IP address ranges such as 172.16/22 so you get both the IP range and mask together, if your just generically referring just the mask then we can simply refer to it as a /22 as you can see it's short and accurate. Now on to question four..

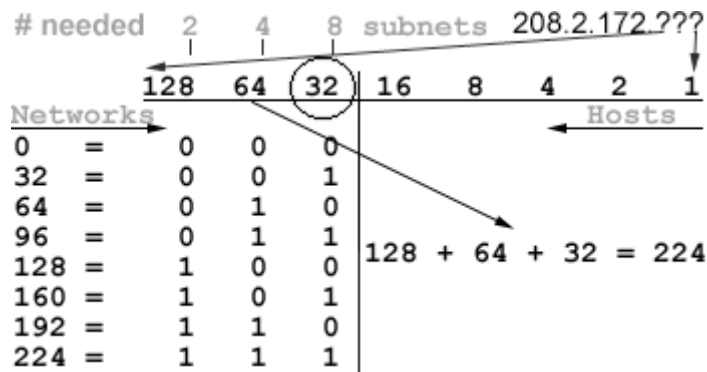
4) What Is Your Block-Size?

This too is easy and again we can look to the answer of the previous question to satisfy our need here. The block-size is the lowest number found in the mask. Going back to our first example from question two, the lowest number is 64 and you may have guessed by now it is used help determine the answer to question five, "What Are Your Subnets?" Each subnet increments by the value of our block-size 64.



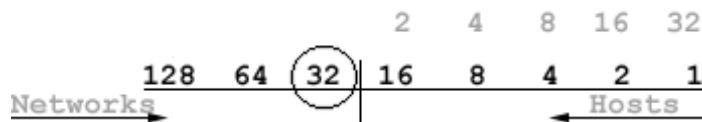
(Figure 18)

In the next figure you can see our block-size is 32 for our second example question two and the subnets unravel by increments of 32.



(Figure 19)

Before we move on to the next question lets take a moment to recognize one other interesting shortcut. Look below and notice how the block size also happens to match the remaining number of possible hosts **in the octet**. So if you ever want to know how many hosts are available in a subnetted octet just look at the block-size.



(Figure 20)

5) What Are Your Subnets?

The cat's out of the bag already but for the sake of being thorough here's the math. Our block size is simply used to calculate each subnet itself. Our first example has a block-size of 64 below so the math becomes simple addition $0 + 64 + 64 + 64$ until we reach our maximum value of 192, the value of the subnet mask itself.

# needed	2	4	subnets	208.2.172.???
	128	64	32 16 8 4 2 1	1
Networks				
0 =	0	0		
64 =	0	1	+ 64 = 64	
128 =	1	0	+ 64 = 128	
192 =	1	1	+ 64 = 192	

(Figure 21)

What we're really asking is how many times will 64 go into 256-bits or put another way I want to slice my 256-bits (an octet) into 4 equal pieces? It's division spelled out. We don't have the luxury of just accepting the answer of 4 because we need to know what the hosts are in each subnet.

Let's do one more example but start from the beginning for practice. This time let's decide we need at least 12 subnets and for the sake of conversation our starting network is 192.168.0/24.

1) How Many Sub-Networks Do You Need?

# needed	2	4	8	16					
	128	64	32	16	8	4	2	1	
Networks									

(2, 4, 8, 16, Stop! You needed 12 but you got 16)

2) How Many Bits Did You Have To Use?

bits used	1	2	3	4					
# needed	2	4	8	16					
	128	64	32	16	8	4	2	1	
Networks									

(We doubled 4 times so we used 4 bits)

3) What Is Your Subnet Mask?

($128 + 64 + 32 + 16 = 240$ mask making our mask for all subnets created 255.255.255.240 or /28 network.)

Networks	128 + 64 + 32 + 16	8	4	2	1	
						= 240 mask

4) What Is Your Block-Size?

(The smallest number in your mask is 16 that's your block-size)

Networks	128 + 64 + 32 + 16	8	4	2	1	
						block-size = 16

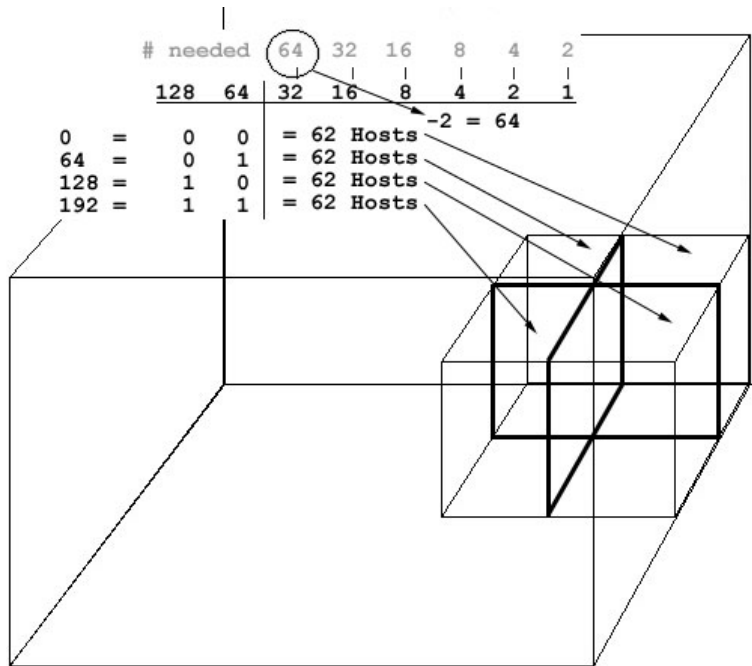
we move on though lets complete two earlier examples we worked on.

In the network where we needed only 4 subnets from the 208.2.172/24 we can now determine the number of hosts. We have 6 bits left over 2, 4, 8, 16, 32, 64 done! Now subtract -2, this gives us 62 hosts per subnet or 62 hosts x 4 subnets = 248 total hosts.

	# needed	64	32	16	8	4	2	
	128	64	32	16	8	4	2	1
0	=	0	0	=	62 Hosts	-2 = 64		
64	=	0	1	=	62 Hosts			
128	=	1	0	=	62 Hosts			
192	=	1	1	=	62 Hosts	← Hosts		

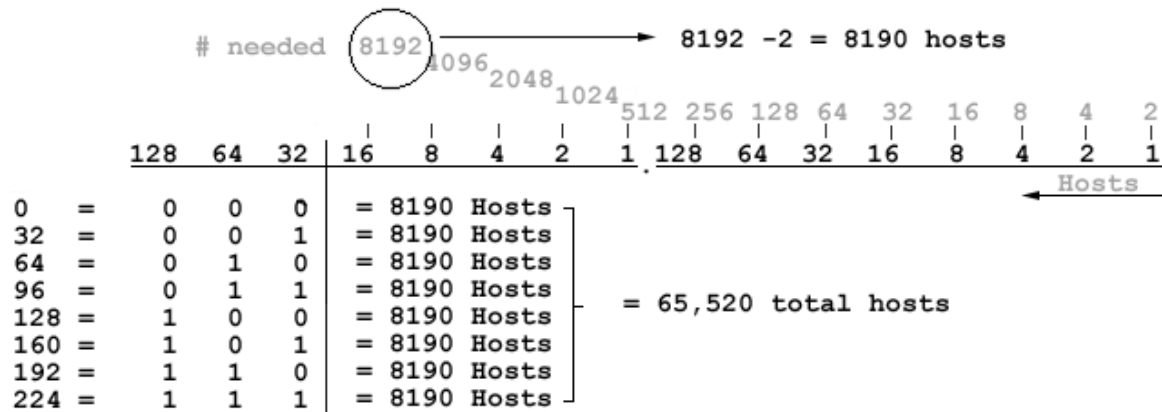
(Figure 24)

If we correlate this to our office suite example at the beginning of this paper to this example, effectively so far we have created 4 office spaces (subnets) for people to work at. We determined 62 cubes (hosts) go into each office space for a total of 248 cubes (hosts) inside the Suite #172 just so we can send mail to Alfred E. Newman.



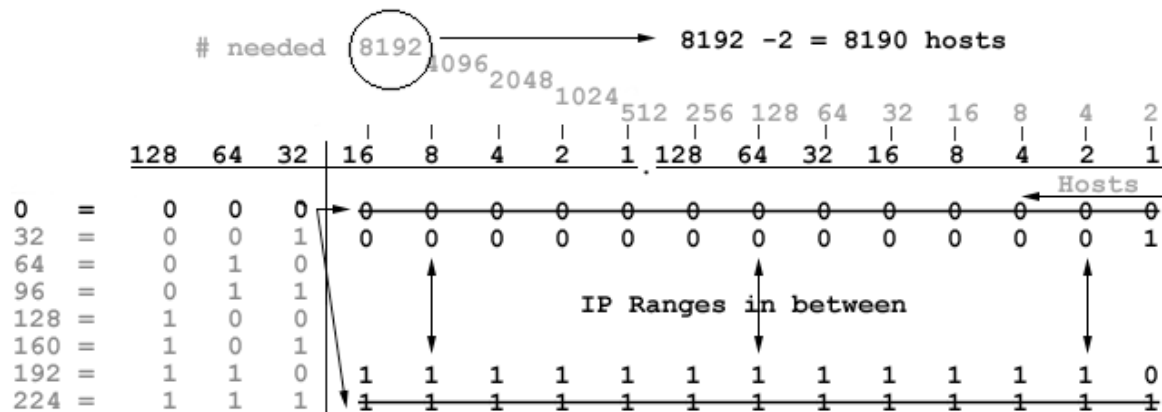
(Figure 25)

We also had an example where we needed six subnets but got 8 giving us 224 mask, let's apply this subnet mask to a 172.31/16 network to demonstrate how the $2^n - 2$ formula takes care of itself when subnetting the second or third octet. Looking below you can see how we determine the number of hosts 2, 4, 8 etc. until we get $8192 - 2 = 8190$ hosts per subnet. We would never use that many in a real subnet but this does show how a computer again ignores the period and looks at all the hosts at once.



(Figure 26)

Looking at the next image shows this also and how the $2^n - 2$ formula applies. We can't use the first or last host (all 0's and 1's) however notice the second line from the top and second line from the bottom and look at only the third octet. It uses all zeros and ones! When you enter your IP in the third octet you don't need to concern yourself with subtracting two because the first and last values for the hosts in each subnet is always defined by the one bit in the fourth octet. The value in the fourth octet will just be 1 through 254 just like in classfull IP addressing.



(Figure 27)

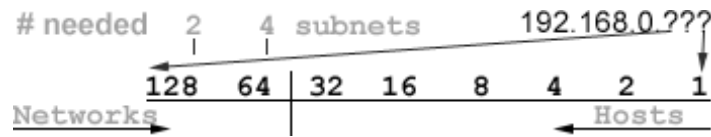
This brings us to the last part of step six...

6b) What Are The IP Ranges For Each Subnet?

Now that you understand the how $2^n - 2$ applies to both the third octet and fourth octet lets do all six steps. We will start by subnetting a class C address 192.168.0/24 and we need at least 3 subnets.

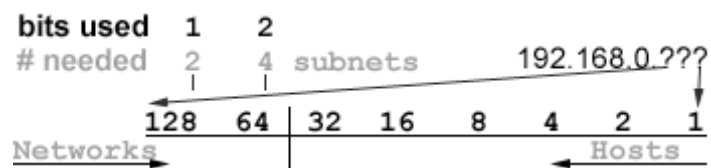
1) How Many Sub-Networks Do You Need?

(You needed 3 but you got 4)



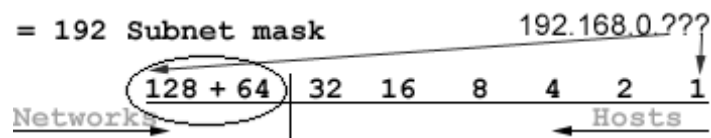
2) How Many Bits Did You Have To Use?

(We doubled 2 times so we used 2 bits)



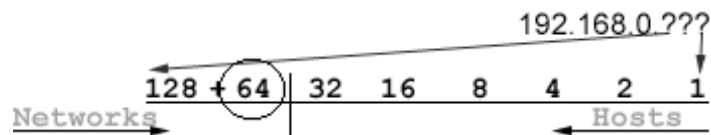
3) What Is Your Subnet Mask?

(128 + 64 = 192 mask)



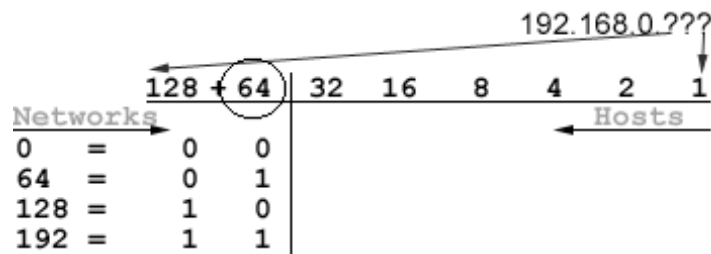
4) What Is Your Block-Size?

(The smallest number in your mask is 64 that's your block-size)



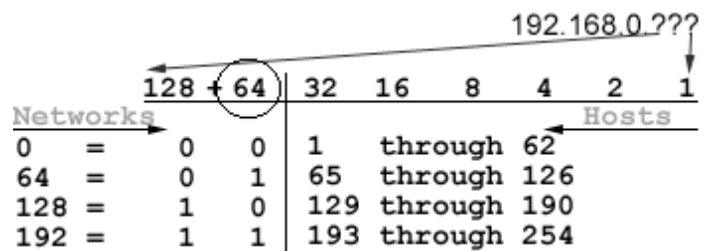
5) What Are Your Subnets?

(Add from 0 by the block-size value, 0 + 64 + 64 + 64 etc. 4 times gives us our 4 subnets)



6) What Are The Number Of Hosts And IP Ranges For Each Subnet?

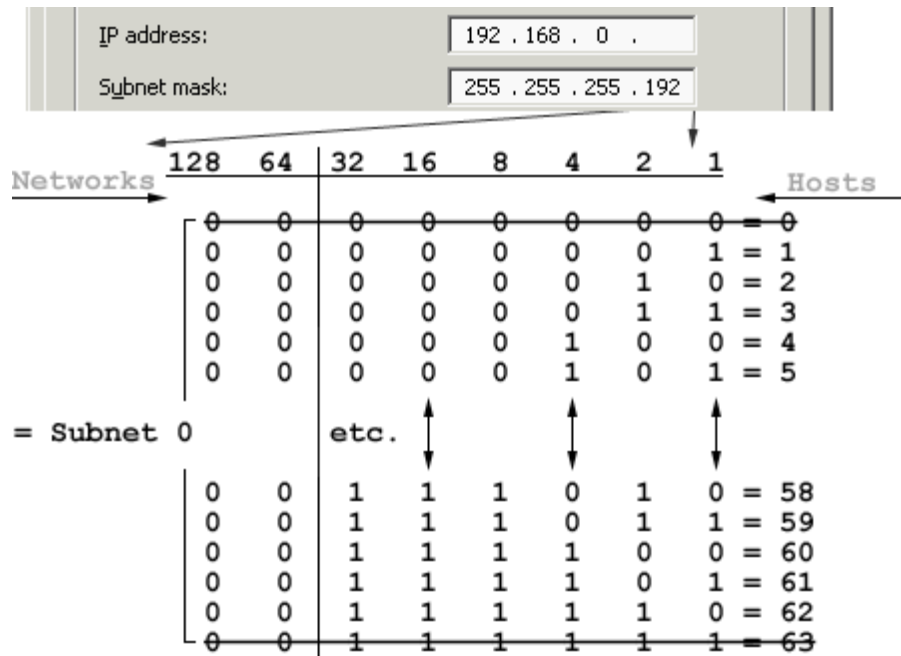
(Because were subnetting the fourth octet we concern ourselves with $2^n - 2$ so we have to remove the first host record and the last host record from our host range for each subnet)



NOTE: The first host address is 0 and the last is 63 in subnet 0, we can't use them so we shave them off and use 1 and 62 and just repeat this for each subnet.

(Figure 28)

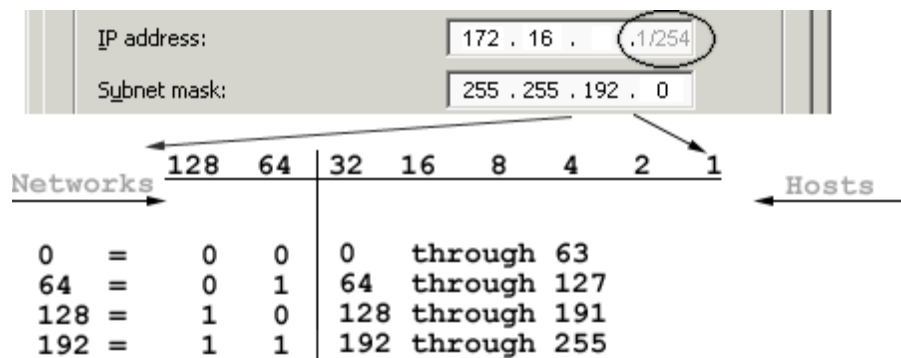
Let's look at subnet 0 in binary, as you can see visually 0 and 63 can't be used. You can think of these values as being the separating walls between subnetworks just as a wall in an office building separates rooms, in both cases the wall itself requires a certain amount of space.



(Figure 29)

Also (Figure 29) demonstrates the abstraction between binary and decimal numbers, even though you must enter a decimal value on the network interface of your computer, the network mask 192 makes your IP address represent partially both network and host. If you were to enter the decimal value of 5 on your NIC from the host list above, what you are actually telling the computer is your computer is in the 0 subnet and your computer is host number 5, your adding the network and host together for a combined value of 5 defining both the computer's place in the network and its host identity.

If we were applying the same 192-mask to a class B network such as 172.31/16 then we would need to define the hosts for 3rd octet. The process is identical up until we get to step 6 determining the number of hosts and their address ranges. At that point our job is even easier, we simply apply all the hosts within the subnet and the fourth octet will take care of the $2^n - 2$ formula, its values will be 1 through 254.



(Figure 30)