Subnetting Examples

There are three types of subnetting examples I will show in this document:

1) Subnetting when given a required number of networks

2) Subnetting when given a required number of clients

3) Given an IP address & Subnet Mask, finding original network range (reverse engineering a subnet problem)

Subnetting Examples document created by Jeremy D. Cioara. Please do not change this document without first contacting me! Thank you!

If you have any questions, please email me at Jeremy@CBTNuggets.com

Subnetting, Style 1 - ***Subnetting when given a required number of networks***

Example 1: A service provider has given you the Class C network range 209.50.1.0. Your company must break the network into 20 separate subnets.

Step 1) **Determine the number of subnets and convert to binary**

- In this example, the binary representation of 20 = 00010100.

Step 2) **Reserve required bits in subnet mask and find incremental value**

- The binary value of 20 subnets tells us that we need at least 5 network bits to satisfy this requirement (since you cannot get the number 20 with any less than 5 bits – 10100)
- Our original subnet mask is 255.255.255.0 (Class C subnet)
- The full binary representation of the subnet mask is as follows:

  255.255.255.0 = 11111111.11111111.11111111.00000000

- We must "convert" 5 of the client bits (0) to network bits (1) in order to satisfy the requirements:

  New Mask = 11111111.11111111.11111111.11111000

- If we convert the mask back to decimal, we now have the subnet mask that will be used on all the new networks – 255.255.255.248
- Our increment bit is the last possible network bit, converted back to a binary number:

  New Mask = 11111111.11111111.11111111.1111(1)000 – bit with the parenthesis is your increment bit. If you convert this bit to a decimal number, it becomes the number '8'

Step 3) **Use increment to find network ranges**

- Start with your given network address and add your increment to the subnetted octet:

  209.50.1.0
  209.50.1.8
  209.50.1.16
  …etc

- You can now fill in your end ranges, which is the last possible IP address before you start the next range

  209.50.1.0 – 209.50.1.7
  209.50.1.8 – 209.50.1.15
  209.50.1.16 – 209.50.1.23
  …etc

- You can then assign these ranges to your networks! *Remember the first and last address from each range (network / broadcast IP) are unusable*

Example 2: Your company would like to break the Class B private IP address range 172.16.0.0 into 60 different subnets

Step 1) **Determine the number of subnets and convert to binary**

- In this example, the binary representation of 60 = 00111100

Step 2) **Reserve required bits in subnet mask and find incremental value**

- The binary value of 60 subnets tells us that we need at least 6 network bits to satisfy this requirement (since you cannot get the number 60 with any less than 6 bits – 111100)
- Our original subnet mask is 255.255.0.0 (Class B subnet)
- The full binary representation of the subnet mask is as follows:

  255.255.0.0 = 11111111.11111111.00000000.00000000

- We must "convert" 6 of the client bits (0) to network bits (1) in order to satisfy the requirements:

  New Mask = 11111111.11111111.11111100.00000000

- If we convert the mask back to decimal, we now have the subnet mask that will be used on all the new networks – 255.255.252.0
- Our increment bit is the last possible network bit, converted back to a binary number:

  New Mask = 11111111.11111111.11111(1)00.00000000 – bit with the parenthesis is your increment bit. If you convert this bit to a decimal number, it becomes the number '4'

Step 3) **Use increment to find network ranges**

- Start with your given network address and add your increment to the subnetted octet:

  172.16.0.0
  172.16.4.0
  172.16.8.0
  …etc

- You can now fill in your end ranges, which is the last possible IP address before you start the next range

  172.16.0.0 – 172.16.3.255
  172.16.4.0 – 172.16.7.255
  172.16.8.0 – 172.16.11.255
  …etc

- You can then assign these ranges to your networks! *Remember the first and last address from each range (network / broadcast IP) are unusable*

Subnetting, Style 2 - ***Subnetting when given a required number of clients***

Example 1: A service provider has given you the Class C network range 209.50.1.0. Your company must break the network into as many subnets as possible as long as there are *at least* 50 clients per network.

Step 1) **Determine the number of clients and convert to binary**

- In this example, the binary representation of 50 = 00110010

Step 2) **Reserve required bits in subnet mask and find incremental value**

- The binary value of 50 clients tells us that we need at least 6 client bits to satisfy this requirement (since you cannot get the number 50 with any less than 6 bits – 110010)
- Our original subnet mask is 255.255.255.0 (Class C subnet)
- The full binary representation of the subnet mask is as follows:

  255.255.255.0 = 11111111.11111111.11111111.00000000

- We must ensure 6 of the client bits (0) *remain* client bits (save the clients!) in order to satisfy the requirements. All other bits can become network bits:

  New Mask = 11111111.11111111.11111111.11 000000 ← note the 6 client bits that we have saved

- If we convert the mask back to decimal, we now have the subnet mask that will be used on all the new networks – 255.255.255.192
- Our increment bit is the last possible network bit, converted back to a binary number:

  New Mask = 11111111.11111111.11111111.1(1)000000 – bit with the parenthesis is your increment bit. If you convert this bit to a decimal number, it becomes the number '64'

Step 3) **Use increment to find network ranges**

- Start with your given network address and add your increment to the subnetted octet:

  209.50.1.0
  209.50.1.64
  209.50.1.128
  209.50.1.192

- You can now fill in your end ranges, which is the last possible IP address before you start the next range

  209.50.1.0 – 209.50.1.63
  209.50.1.64 – 209.50.1.127
  209.50.1.128 – 209.50.1.191
  209.50.1.192 – 209.50.1.255

- You can then assign these ranges to your networks! *Remember the first and last address from each range (network / broadcast IP) are unusable*

Example 2: Your company would like to break the Class B private IP address range 172.16.0.0 into as many subnets as possible, provided that they can get *at least* 300 clients per subnet

Step 1) **Determine the number of clients and convert to binary**

- Remember, the binary representations of 8 bits (a single octet of an IP address) can only reach 255, but that does not mean we cannot cross octet boundaries when working with Class A or B examples!
- In this example, the binary representation of 300 = 100101100

Step 2) **Reserve required bits in subnet mask and find incremental value**

- The binary value of 300 clients tells us that we need at least 9 client bits to satisfy this requirement (since you cannot get the number 300 with any less than 9 bits – 100101100)
- Our original subnet mask is 255.255.0.0 (Class B subnet)
- The full binary representation of the subnet mask is as follows:

  255.255.0.0 = 11111111.11111111.00000000.00000000

- We must ensure 9 of the client bits (0) *remain* client bits (save the clients!) in order to satisfy the requirements. All other bits can become network bits:

  New Mask = 11111111.11111111.1111111 0.00000000 ← note the 9 client bits that we have saved

- If we convert the mask back to decimal, we now have the subnet mask that will be used on all the new networks – 255.255.254.0
- Our increment bit is the last possible network bit, converted back to a binary number:

  New Mask = 11111111.11111111.111111(1)0.0000000 – bit with the parenthesis is your increment bit. If you convert this bit to a decimal number, it becomes the number '2'

Step 3) **Use increment to find network ranges**

- Start with your given network address and add your increment to the subnetted octet:

  172.16.0.0
  172.16.2.0
  172.16.4.0
  etc…

- You can now fill in your end ranges, which is the last possible IP address before you start the next range

  172.16.0.0 – 172.16.1.255
  172.16.2.0 – 172.16.3.255
  172.16.4.0 – 172.16.5.255
  etc…

- You can then assign these ranges to your networks! *Remember the first and last address from each range (network / broadcast IP) are unusable*

Subnetting, Style 3 - **Given an IP address & Subnet Mask, find original network range** (reverse engineering a subnet problem)

You are given the following IP address and subnet mask:

192.168.1.58
255.255.255.240

Identify the original range of addresses (the subnet) that this IP address belongs to

- When reverse engineering a problem, all you need to do is break the subnet mask back into binary and find the increment that was used

    255.255.255.240 = 11111111.11111111.11111111.11110000

- As before, the last possible network bit is your increment. In this case, the increment is 16
- Use this increment to find the network ranges until you pass the given IP address:

    192.168.1.0
    192.168.1.16
    192.168.1.32
    192.168.1.48
    192.168.1.64 (*passed given IP address 192.168.1.58*)

- Now, fill in the end ranges to find the answer to the scenario:

    192.168.1.0 – 192.168.1.15
    192.168.1.16 – 192.168.1.31
    192.168.1.32 – 192.168.1.47
    192.168.1.48 – 192.168.1.63 (*IP address 192.168.1.58 belongs to this range*)

The Great Exception

This subnetting process works by finding the number of bits that are required to meet a given requirement. For example, if I wanted to break a range into 25 networks, I know that I'll need to convert 5 bits to network bits to satisfy the requirement. However, because binary numbering counts from zero, there are exceptions to this rule. For example, if I were asked to break a network into 8 subnets, you would assume it would take four bits since 8 in binary is:

0 0 0 0 1 0 0 0

However, you can achieve this requirement with only three bits since 0-7 is really 8 numbers (0, 1, 2, 3, 4, 5, 6, 7). If you work out the subnetting problem by reserving only three bits, you will get exactly eight subnets. The same thing happens when breaking into networks for every "precise" binary number: 2, 4, 8, 16, 32, 64, 128. So how do you avoid this issue? You can always subtract 1 from the number of networks required. For example, if you're asked to break a range into 30 subnets, figure it out for 29. If you're asked to break a range into 16 subnets, figure it out for 15…and so on.

There is a similar rule for finding the number of bits to reserve for hosts. For example, if I wanted to break a range into 25 hosts per network, I know that I'll need to save 5 bits as host bits to satisfy the requirement. This rule works fine, but does not account for the two hosts that are unusable from every range: the network IP address and the broadcast IP address. The fact that binary counting starts from zero helps us with one of those IP addresses, but we can still come up one short in certain cases. For example, if we were asked to break a network into subnets that can hold up to 31 hosts, you would assume it would take 5 bits since 31 in binary is:

0 0 0 1 1 1 1 1

However, when you work out the problem you'll find that you only get 30 hosts per subnet (1 IP address short). The same thing happens with every "full" binary number: 3, 7, 15, 31, 63, 127. So how do you avoid this issue? You can always add 1 to the number of hosts required. For example, if you're asked to break a range into 25 hosts per subnet, figure it out for 26. If you're asked to break a range into 63 subnets, figure it out for 64…and so on.

I realize this exception gets somewhat technical… so to summarize:

When subnetting based on the number of networks, SUBTRACT 1 from the number

When subnetting based on the number of hosts per network, ADD 1 to the number

Follow these rules and you'll always be safe.