SUBNETTING SUBNETS WITHOUT CONSTRAINTS

Larry R. Newcomer
The Pennsylvania State University
York Pennsylvania 17403

Abstract

Most available treatments of subnetting are constrained by assumptions that classful addressing is being used, that the number of bits borrowed from the Host ID to create the Subnet ID does not exceed eight, that the bits borrowed from the Host ID to create the Subnet ID do not cross an octet boundary, and that subnet zero is not being used. This paper presents procedures and formulas for use in the many common situations not subject to the above constraints. The procedures and formulas below solve the subnetting problem using decimal rather than binary arithmetic, making them more accessible to students and faster and more reliable in professional environments or when sitting for certification exams. The methodology gives faculty, networking professionals, and students an effective tool for understanding and quickly solving generalized subnetting problems.

Procedures for Subnetting Any Subnet

The following discussion assumes that the reader is already familiar with basic subnetting operations, their binary underpinnings, and their uses and applications. This material can be found in Ref. [1].

Most academic discussions of subnetting are bound by at least some of the following assumptions [2 – 8]:

- That the network being subnetted uses a standard class A, B, or C classful address and uses a standard default class A, B, or C subnet mask
- That the number of bits borrowed from the Host ID to create the Subnet ID does not exceed eight bits (one octet)
- That the bits borrowed from the Host ID to create the Subnet ID do not cross an octet boundary
- That the all 0's (subnet zero) and all 1's subnets are prohibited

This paper presents procedures and formulas for use in situations not subject to the above constraints. Examples include subnetting problems where classless (CIDR) addressing is being used, where the number of bits borrowed from the Host ID to create the Subnet ID exceeds eight or crosses an octet boundary, and where subnet zero is deployed. In modern networking these situations are common and arise frequently when one needs to subnet a network address that itself has been derived by subnetting a higher-level address space. Real-life examples include the following situations:

- When subnetting a private classless (CIDR) or classful network address (which may be a subnet of an encompassing private network address)
- When subnetting a public classless (CIDR) or classful network address obtained from an ISP (which is likely to be a subnet of the ISP's public address space)
- When subnetting an existing subnet of any type (for example, using Variable Length Subnet Masks [VLSM])

In addition, the procedures and formulas below solve the subnetting problem using decimal rather than binary arithmetic, making them more accessible to students and faster and more reliable in professional environments or when sitting for certification exams. The methodology gives faculty, networking professionals, and students an effective tool for understanding and quickly solving generalized subnetting problems.

Preliminary Operations

Given a dotted decimal subnet mask to be further subnetted, carry out the following steps:

Step 1

Let L = the count of the number of leading (leftmost) octets equal to 255 in the original dotted decimal subnet mask

Step 2

Let V = the value of the first (leftmost) octet NOT equal to 255 in the original dotted decimal subnet mask (note that this may be a zero octet, in which case V is zero)

Now perform the following calculations:

Step 3

$b = 8 - \log_2 (256 - V)$, the number of network/subnet bits (i.e., binary 1 bits) in the "boundary" octet (where 1's change to 0's in the subnet mask). Note that b is guaranteed to be an integer value between 0 and 7 (see Lemmas 1 and 2 below for a proof of this assertion)

Step 4

$p = 8 * L + b$, the total number of network/subnet bits in the custom subnet mask (this is equal to the prefix in prefix notation)

Now carry out the following estimates:

Step 5

Estimate S = number of subnets needed. See Ref. [1] for more information on estimating S

Step 6

Estimate H = maximum number of hosts per subnet (i.e., number of hosts on the largest subnet for this round of subnetting. If implementing VLSM, H typically will decrease with each round of subnetting). See Ref. [1] for more information on estimating H

Now calculate:

Step 7

s = smallest integer x such that $2^x - 2 >= S$, the number of additional subnet bits that must be borrowed from the host ID

Note that initially subnet IDs consisting of all 0 bits (called the "all 0's subnet" or "subnet zero") or all 1 bits (called the "all 1's subnet") were prohibited. Modern software, when properly configured, permits the use of these subnet IDs. If your network is configured to use subnet zero, then find the smallest integer x such that $2^x >= S$

Step 8

h = smallest integer y such that $2^y - 2 >= H$, the number of host ID bits that must be kept. Note that since a valid host ID may not be either all 0's or all 1's (in binary), h must be >= 2

Step 9

Finally, ensure that $s + h <= 32 - p$

If $s + h > 32 - p$, the problem is not solvable. You must either revise your estimates for S and H and/or obtain a different address block and subnet mask from your ISP or from private IP address space. After successfully passing step 9, you are ready to produce the three main deliverables of the subnetting process: i) the custom subnet mask for each subnet, ii) the subnet address for each subnet, and iii) the starting and ending IP addresses for each subnet [2]. Each of these three deliverables is discussed in turn below.

Produce the New Custom Subnet Mask

If the inequality in step 9 holds (in which case the problem is solvable), calculate:

$p' = p + s$, the new total of network/subnet bits in the Custom Subnet Mask (note that p' is also the new prefix value when prefix notation is used)

L' = *floor* (p' / 8), where *floor* (x) is the largest integer <= x

r = *remainder* (p' / 8), where *remainder* (x/y) is the remainder upon dividing the integer x by the integer y. "r" is the remaining number of 1 bits to be borrowed for the subnet ID in the "boundary" octet

If r = 0, then form the new custom subnet mask as L' octets consisting of all binary 1's (written as 255 in dotted-decimal) followed by enough octets consisting of all binary 0's (written as 0 in dotted-decimal) to make up a total of 4 octets

If r <> 0, then form the new custom subnet mask as

- L' octets consisting of all binary 1's (written as 255 in dotted-decimal), followed by
- An octet equal to $256 - 2^{(8-r)}$, followed by
- Enough octets consisting of all binary 0's (written as 0 in dotted-decimal), if any, to make up a total of 4 octets

Produce the Subnet Address for Each New Subnet

1. Start with the original network/subnet address, which is the address of subnet-zero when the new custom subnet mask is applied.
2. Work in the L' + 1 octet from the left.
3. Add $2^{(8-r)}$ to the working octet to get the address of subnet 1.
4. Add $2^{(8-r)}$ to the working octet a second time to get the address of subnet 2.
5. All adding must be modulo 256: when the "working" octet sum equals 256, set the working octet to 0 and carry 1 into the next octet to the left. If the carry causes the octet to become 256, set it to 0 and carry 1 into the next octet to its left, etc.
6. Continue adding $2^{(8-r)}$ to the working octet until you reach the subnet address that, if incremented, would produce the address of the subnet immediately following the original subnet address that you began with. This is the "all ones" subnet – the last possible subnet.
7. You should have a total of 2^s subnets, counting the all-zeros and all-ones subnets (or $2^s - 2$ subnets if you are not using subnet zero). See Ref. [1] for more examples and discussion on calculating subnet addresses.

Produce the Starting and Ending IP Addresses for Each New Subnet

To generate the valid IP addresses for a given subnetwork, start with that subnetwork's subnet address. Add 1 to the rightmost octet in the subnet address to obtain the first valid IP address on that subnet. For example, if the Network ID of the first subnet is 152.201.16.0, then the first valid IP address on that subnet would be 152.201.16.1.

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 152.201.16.254 to obtain 152.201.16.255. The next step would not be to add 1 again to obtain 152.201.16.256 (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 152.201.17.0. 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. See Ref. [1] for more examples and discussion on calculating IP addresses for a subnet.

Lemmas

Lemma 1

$\log_2 (256 - V)$ is an integer between 1 and 8 (thus making $b = 8 - \log_2 (256 - V)$ an integer value between 0 and 7), where V is the leftmost non-255 octet in a subnet mask. This means V is a properly formed octet in a subnet mask that is not equal to 255, i.e., V is an 8-bit string consisting of an uninterrupted substring of one or more 1 bits followed by an uninterrupted substring of one or more 0 bits, with the total number of combined bits being 8.

Proof:

If $V = 0$, then $\log_2 (256 - V) = \log_2 (256 - 0) = 8$, as claimed

If V is not zero, then in order to be the leftmost non-255 octet in a properly formed subnet mask, V must be of the form $V = 2^7 + 2^6 + \ldots + 2^d + 0 + \ldots + 0$, where d is the binary digit position in the octet corresponding to the last 1 before the first 0 (positions being numbered according to the powers of 2 represented, i.e., 76543210)

Note that d must be an integer between 1 and 7 (if d falls in position 0, the octet value would be 255 and therefore it would not be the leftmost non-255 octet, a contradiction). Hence $8 - d$ must be an integer between 1 and 7, which is greater than zero (as required for Lemma 2 below)

Thus $256 - V$

$= 2^8 - V$, since $2^8 = 256$
$= 2^8 - (2^7 + 2^6 + \ldots + 2^d)$, by substituting the value of V
$= 2^d * [2^{(8-d)} - (2^{(7-d)} + 2^{(6-d)} + \ldots + 2(^{d-d})]$, by factoring 2^d out of the above expression

$= 2^d * [2^{(8-d)} - (2^{(7-d)} + 2^{(6-d)} + \ldots + 2^0)]$
$= 2^d * [2^{(8-d)} - (2^{(8-d)} - 1)]$, by substitution for the inner parenthetical expression per Lemma 2 below
$= 2^d * 1$, since the expression in square brackets evaluates to 1
$= 2^d$,which is a power of two

Since $256 - V = 2^d$, $\log_2 (256 - V) = \log_2 (2^d) = d$, an integer between 1 and 7.

Thus in all cases, $\log_2 (256 - V)$ is between 1 and 8, which was to be proved.

Lemma 2

For any integer $n > 0$, $2^n - 1 = 2^{(n-1)} + 2^{(n-2)} + ... + 2^1 + 2^0$

Proof (by induction):

Since n must be greater than 0, we start with the case $n = 1$:
$2^1 - 1 = 1 = 2^0 = 2^{(1-1)}$, which was to be shown
Now assume that for any $k > 0$, $2^k - 1 = 2^{(k-1)} + 2^{(k-2)} + ... + 2^1 + 2^0$. We must then show that $2^{(k+1)} - 1 = 2^{(k+1-1)} + 2^{(k+1-2)} + \ldots + 2^1 + 2^0$. We do this as follows:
$2^{(k+1)} - 1$
$= 2^{(k+1)} - 2 + 1$, since $-2 + 1 = -1$
$= 2 * (2^k - 1) + 1$, by factoring out 2
$= 2 * (2^{(k-1)} + 2^{(k-2)} + ... + 2^1 + 2^0) + 1$, by substitution based on the inductive assumption
$= (2^k + 2^{(k-1)} + ... + 2^2 + 2^1) + 1$, by multiplying by 2
$= 2^k + 2^{(k-1)} + ... + 2^2 + 2^1 + 2^0$, by rewriting the trailing 1 as 2^0
$= 2^{(k+1-1)} + 2^{(k+1-2)} + \ldots + 2^1 + 2^0$, which was to be proved

References

1. L. R. Newcomer, "Subnetting Made Simple: IP Subnetting without Tables, Tools or Tribulations," *Computers in Education Journal*, vol. XII, no. 3, July – September 2002.

2. K. Caudle and K. Cannon, *CCNA Guide to Cisco Networking*, 3rd ed., Thomson/Course Technology, Boston, 2004.

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

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

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

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

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

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

Biographical Information

Larry Newcomer is an Associate Professor of Information Sciences & Technology at The Pennsylvania State University. He has published textbooks and papers in the general areas of networking, databases, and programming. In addition, he is currently engaged in a long-term project to become a decent drummer.