

IP Addressing

IP Subnet Addressing Overview, Motivation, and Advantages

IP addressing was originally designed around the assumption of a strict [two-level hierarchy for internetworks](#). The first level was the *network*, and the second level the *host*. Each organization was usually represented by a single network identifier that indicated a [Class A, B or C block](#) dedicated to them. Within that network they had to put all of the devices they wanted to connect to the public IP network.

The Motivation for Subnet Addressing

It did not take long after the “classful” scheme was developed for serious inadequacies in it to be noticed, especially by larger organizations. The reason is that while dividing a large internetwork into networks that contain hosts is conceptually simple, it doesn't always match well the structure of each of the networks that comprises the internet. A big company with thousands of computers doesn't structure them as one big whopping physical network. Trying to assign and administer IP addresses to an organization's entire network without any form of internal logical structure is very difficult.

Unfortunately, under the original “classful” addressing scheme, there was no good solution to this problem. The most commonly-chosen alternative at the time was to trade a single large block of addresses such as a Class B for a bunch of Class Cs. However, this caused additional problems:

- It contributed to the explosion in size of IP routing tables.
- Every time more address space was needed, the administrator would have to apply for a new block of addresses.
- Any changes to the internal structure of a company's network would potentially affect devices and sites outside the organization.
- Keeping track of all those different Class C networks would be a bit of a headache in its own right.



Related Information: I fully explain [the problems with “classful” addressing](#) within [the section on “classful” addressing](#).

The Development of Subnet Addressing

In order to address this problem adequately, an enhancement was required to the “classful” addressing scheme. This change was outlined in RFC 950, which defined a new addressing procedure called *subnet addressing* or *subnetting*. (This RFC was published way back in 1985, which surprises some people!)

The basic idea behind subnet addressing is to add an additional hierarchical level in the way IP addresses are interpreted. The concept of a network remains unchanged, but instead of having just “hosts” within a network, a new two-level hierarchy is created: *subnets* and hosts. Each subnet is a subnetwork, and functions much the way a full network does in conventional classful addressing. A three-level hierarchy is thus created: networks, which contain subnets, each of which then has a number of hosts.

Thus, instead of an organization having to lump all of its hosts under that network in an unstructured manner, it can organize hosts into subnets that reflect the way internal networks are structured. These subnets fit within the network identifier assigned to the organization, just as all the “unorganized” hosts used to.

Advantages of Subnet Addressing

In essence, subnet addressing allows each organization to have its own “internet within the Internet”. Just as the real Internet looks only at networks and hosts, a two-level hierarchy, each organization can now also have subnets and hosts within their network. This change provides numerous advantages over the old system:

- **Better Match to Physical Network Structure:** Hosts can be grouped into subnets that reflect the way they are actually structured in the organization's physical network.
- **Flexibility:** The number of subnets and number of hosts per subnet can be customized for each organization. Each can decide on its own subnet structure and change it as required.
- **Invisibility To Public Internet:** Subnetting was implemented so that the internal division of a network into subnets is visible only within the organization; to the rest of the Internet the organization is still just one big, flat, “network”. This also means that any changes made to the internal structure are not visible outside the organization.
- **No Need To Request New IP Addresses:** Organizations don't have to constantly requisition more IP addresses, as they would in the workaround of using multiple small Class C blocks.
- **No Routing Table Entry Proliferation:** Since the subnet structure exists only within the organization, routers outside that organization know

nothing about it. The organization still maintains a single (or perhaps a few) routing table entries for all of its devices. Only routers inside the organization need to worry about routing between subnets.

The Impact of Subnetting on Addressing and Routing

The change to subnetting affects both addressing and routing in IP networks. Addressing changes of course, because instead of having just a network ID and host ID, we now also have a *subnet ID* to be concerned with. The size of the subnet ID can vary for each network, so an additional piece of information is needed to supplement the IP address to indicate what part of the address is the subnet ID and what part is the host ID. This is a 32-bit number commonly called a *subnet mask*. The mask is used both for calculating subnet and host addresses, and by routers for determining how to move IP datagrams around a subnetted network.

Routing changes because of the additional level of hierarchy. In regular “classful” addressing, when a router receives an IP datagram, it only needs to decide if the destination is on the same network or a different network. Under subnetting, it must also look at the subnet ID of the destination and make one of three choices: same subnet, different subnet on the same network, or different network. Again, this is done using the subnet mask. Changes are also required to routing protocols, such as the [Routing Information Protocol \(RIP\)](#), to deal with subnets and subnet masks.



Key Concept: Subnet addressing adds an additional hierarchical level to how IP addresses are interpreted, by dividing an organization’s IP network into *subnets*. This allows each organization to structure its address space to match its internal physical networks, rather than being forced to treat them a flat block. This solves a number of problems with the original “classful” addressing scheme, but requires changes to how addressing and routing work, as well as modifications to several TCP/IP protocols.

It’s funny, but the main drawbacks to subnetting, compared with the older addressing scheme, have more to do with understanding how subnetting works than the technology itself! More effort is required to deal with addressing and routing in a subnet environment, and administrators must learn how to subdivide their network into subnets and properly assign addresses. This can be a bit confusing to someone who is new to subnetting. However, the technology today is quite well-established so even this is not much of a problem. For the newcomer, having a handy reference guide like this one certainly helps. ☺

IP Subnetting: "Three-Level" Hierarchical IP Subnet Addressing

The simplest division of IP addresses is into a structure containing two elements: the *network ID* and the *host ID*. [In explaining this concept](#), I drew an analogy to the way North American phone numbers are ten digits long, but are broken down into a three-number area code and a seven-digit local number.

[As I mentioned in the preceding topic](#), subnetting adds an additional level to the hierarchy of structures used in IP addressing. To support this, IP addresses must be broken into three elements instead of two. This is done by leaving the network ID alone and dividing the host ID into a *subnet ID* and host ID. These subnet ID bits are used to identify each subnet within the network. Hosts are assigned to the subnets in whatever manner makes the most sense for that network.

Interestingly, the analogy to telephone numbers that we used before still holds in the world of subnetting, and shows how subnetting changes the way IP addresses are interpreted. A number like (401) 555-7777 has an area code ("401") and a local number ("555-7777") as I said before. The local number, however, can itself be broken down into two parts: the exchange ("555") and the local extension ("7777"). This means phone numbers really are comprised of three hierarchical components just as IP addresses are in subnetting.

Of course, the number of bits in an IP address is fixed at 32. This means that in splitting the host ID into subnet ID and host ID, we reduce the size of the host ID portion of the address. In essence, we are "stealing" bits from the host ID to use for the subnet ID. Class A networks have 24 bits to split between the subnet ID and host ID: class B networks have 16, and class C networks only 8.



Key Concept: A "classful" network is subnetted by dividing its host ID portion, leaving some of the bits for the host ID while allocating others to a new *subnet ID*. These bits are then used to identify individual subnets within the network, into which hosts are assigned.

Now, remember when [we looked at the sizes of each of the main classes](#), we saw that for each class, the number of networks and the number of hosts per network are a function of how many bits we use for each. The same applies to our splitting of the host ID. Since we are dealing with binary numbers, the number of subnets is two to the power of the size of the subnet ID field. Similarly, the number of hosts per subnet is two to the power of the size of the host ID field (less two for excluded special cases).

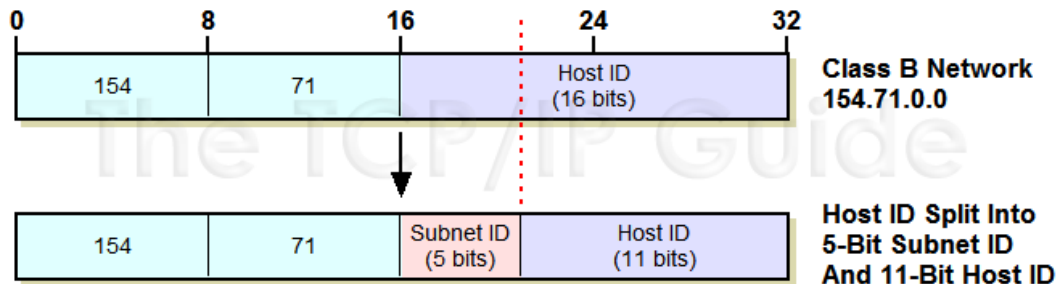


Figure 65: Subnetting A Class B Network

We begin with the Class B network 154.71.0.0, which has 16 bits in its host ID block as shown at top. We then subnet this network by dividing the host ID into a subnet ID and host ID. In this case, 5 bits have been allocated to the subnet ID, leaving 11 for the host ID.

Let's take a brief example to see how this works. Imagine that we start with Class B network 154.71.0.0. 16 bits are for the network ID (154.71) and 16 for the host ID. In regular “classful” addressing there are no subnets (well, one “subnet” that is the whole network, but never mind about that) and 65,534 hosts total. To subnet this network, we can decide to split those 16 bits however we feel best suits the needs of our network: 1 bit for the subnet ID and 15 for the host ID, or 2 and 14, 3 and 13, and so on. Most any combination will work, as long as the total is 16, such as 5 and 11, which I illustrate in Figure 65. The more bits we “steal” from the host ID for the subnet ID, the more subnets we can have—but the fewer hosts we can have for each subnet.

Deciding how to make this choice is one of the most important design considerations in setting up a subnetted IP network. The number of subnets is generally determined based on the number of physical subnetworks in the overall organizational network. The number of hosts per subnetwork must not exceed the maximum allowed for the particular subnetting choice we make. Choosing how to divide the original host ID bits into subnet ID bits and host ID bits is sometimes called *custom subnetting* and [is described in more detail later in this section](#).

IP Subnet Masks, Notation and Subnet Calculations

[Subnetting](#) divides an organization's network into a two-level structure of subnets and hosts. This division is entirely internal and hidden from all other organizations on the Internet. One of the many advantages of this is that each organization gets to make their own choice about how to divide the “classful” host ID into subnet ID and host ID.

In a non-subnetted “classful” environment, [routers use the first octet of the IP address](#) to determine what the class is of the address, and from this they know which bits are the network ID and which are the host ID. When we use subnetting, these routers also need to know how that host ID is divided into subnet ID and host ID. However, this division can be arbitrary for each network. Furthermore, there is no way to tell how many bits belong to each simply by looking at the IP address.

In a subnetting environment, the additional information about which bits are for the subnet ID and which for the host ID must be communicated to devices that interpret IP addresses. This information is given in the form of a 32-bit binary number called a *subnet mask*. The term “mask” comes from the binary mathematics concept called *bit masking*. This is a technique where a special pattern of ones and zeroes can be used in combination with boolean functions such as *AND* and *OR* to select or clear certain bits in a number. [I explain bit masking in the background section on binary numbers and mathematics.](#)

Function of the Subnet Mask

There's something about subnet masks that seems to set people's hair on end, especially if they aren't that familiar with binary numbers. However, the idea behind them is quite straight-forward. The mask is a 32-bit number, just as the IP address is a 32-bit number. Each of the 32 bits in the subnet mask corresponds to the bit in the IP address in the same location in the number. The bits of the mask in any given subnetted network are chosen so that the bits used for either the network ID or subnet ID are ones, while the bits used for the host ID are zeroes.



Key Concept: The *subnet mask* is a 32-bit binary number that accompanies an IP address. It is created so that it has a one bit for each corresponding bit of the IP address that is part of its network ID or subnet ID, and a zero for each bit of the IP address's host ID. The mask thus tells TCP/IP devices which bits in that IP address belong to the network ID and subnet ID, and which are part of the host ID.

Why bother doing this with a 32-bit binary number? The answer is the magic of [boolean logic](#). We use the subnet mask by applying the boolean *AND* function between it and the IP address. For each of the 32 “bit pairs” in the IP address and subnet mask we employ the *AND* function, the output of which is 1 only if both bits are 1. What this means in practical terms is the following, for each of the 32 bits:

- **Subnet Bit Is A One:** In this case, we are *ANDing* either a 0 or 1 in the IP address with a 1. If the IP address bit is a 0, the result of the AND will be

0, and if it is a 1, the AND will be 1. In other words, *where the subnet bit is a 1, the IP address is preserved unchanged.*

- **Subnet Bit Is A Zero:** Here, we are *ANDing* with a 0, so the result is always 0 regardless of what the IP address is. Thus, *when the subnet bit is a 0, the IP address bit is always cleared to 0.*

So, when we use the subnet mask on an IP address, the bits in the network ID and subnet ID are left intact, while the host ID bits are removed. Like a mask that blocks part of your face but lets other parts show, the subnet mask blocks some of the address bits (the host bits) and leaves others alone (the network and subnet bits). A router that performs this function is left with the address of the subnet. Since it knows from the class of the network what part is the network ID, it also knows what subnet the address is on.



Key Concept: To use a subnet mask, a device performs a boolean *AND* operation between each bit of the subnet mask and each corresponding bit of an IP address. The resulting 32-bit number contains only the network ID and subnet ID of the address, with the host ID cleared to zero.

Subnet Mask Notation

Like IP addresses, subnet masks are always used as a 32-bit binary number by computers. And like IP addresses, using them as 32-bit binary numbers is difficult for humans. Therefore, they are usually converted to dotted decimal notation for convenience, just like IP addresses are.

Let's take a quick example to show what this is all about. Suppose we have the Class B network 154.71.0.0. We decide to subnet this using 5 bits for the subnet ID and 11 bits for the host ID. In this case, the subnet mask will have 16 ones for the network portion (since this is Class B) followed by 5 ones for the subnet ID, and 11 zeroes for the host ID. That's "11111111 11111111 11110000 00000000" in binary, with the bits corresponding to the subnet ID highlighted. Converting to dotted decimal, the subnet mask would be 255.255.248.0. Figure 66 illustrates this process.

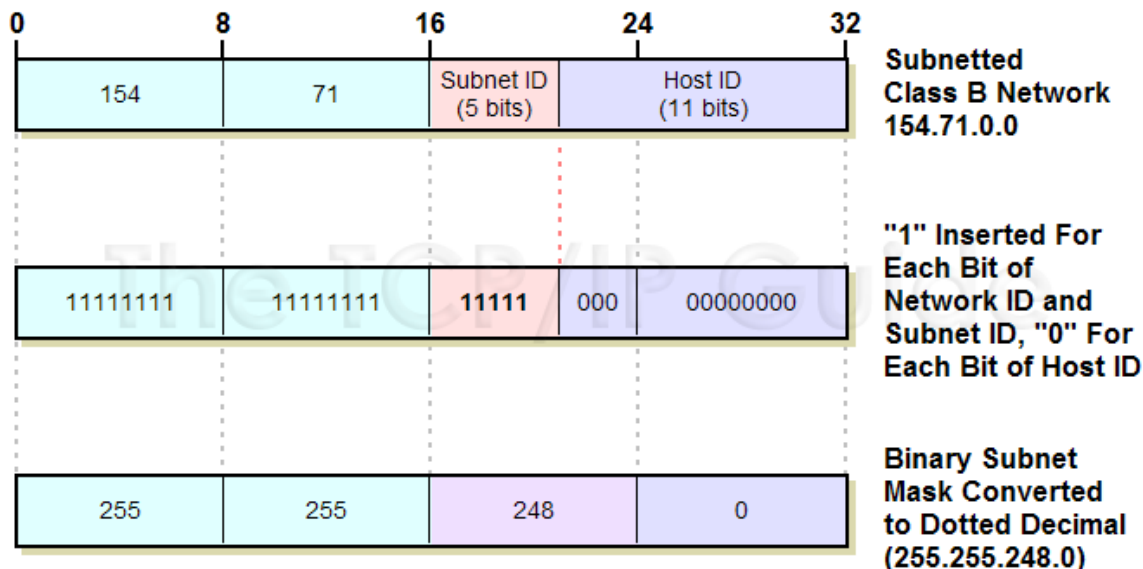


Figure 66: Determining the Subnet Mask of a Subnetted Network

The class B network from Figure 65 has been shown at top, with 5 bits assigned to the subnet ID and 11 bits left for the host ID. To create the subnet mask, we fill in a 32-bit number with "1" for each network ID and subnet ID bit, and "0" for each host ID bit. We can then convert this to dotted decimal.

Applying the Subnet Mask: An Example

Now, let's see how the subnet mask might be used. Suppose we have a host on this network with an IP of 154.71.150.42. A router needs to figure out which subnet this address is on. This is done by performing the masking operation shown in Table 50 and Figure 67.

Table 50: Determining the Subnet ID of an IP Address Through Subnet Masking				
Component	Octet 1	Octet 2	Octet 3	Octet 4
IP Address	10011010 (154)	01000111 (71)	10010110 (150)	00101010 (42)
Subnet Mask	11111111 (255)	11111111 (255)	11111000 (248)	00000000 (0)
Result of AND Masking	10011010 (154)	01000111 (71)	10010000 (144)	00000000 (0)

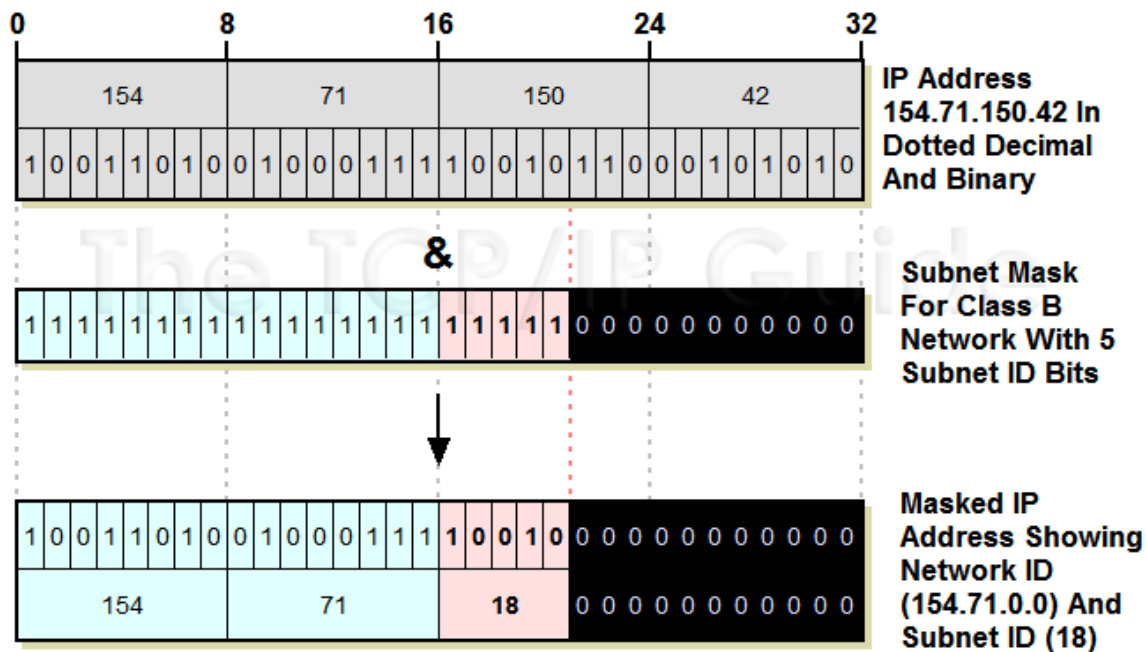



Figure 67: Determining the Subnet ID of an IP Address Through Subnet Masking

Subnet masking involves performing a boolean *AND* between each corresponding bit in the subnet mask and the IP address. The subnet mask can be likened to a [physical mask](#); each 1 in it lets the corresponding bit of the IP address “show through”, while each 0 blocks the corresponding IP address bit. In this way the host ID bits of the address are stripped so the device can determine the subnet to which the address belongs.

This result, 154.71.144.0, is the IP address of the subnet to which 154.71.150.42 belongs. There is no need to explicitly differentiate the network ID bits from the subnet ID bits, because we are still using “classful” addresses. Any router can see that since the first two bits of the address are “10”, this is a Class B address. So the network ID is 16 bits, and this means the subnet ID must be bits 17 to 21, counting from the left. Here, the subnet is the portion highlighted above: “10010”, or subnet #18. ([I'll explain this better in the section on custom subnetting.](#))

 **Key Concept:** The subnet mask is often expressed in dotted decimal notation for convenience, but is used by computers as a binary number, and usually must be expressed in binary to understand how the mask works and the number of subnet ID bits it represents.

Rationale for Subnet Mask Notation

So, in practical terms, the subnet mask actually conveys only a single piece of information: where the line is drawn between the subnet ID and host ID. You might wonder, why bother with a big 32-bit binary number in that case, instead of just specifying the bit number where the division occurs? Instead of carrying the subnet mask of 255.255.248.0 around, why not just say “divide the IP address after bit #21”? Even if devices want to perform a masking operation, could they not just create the mask as needed?

That's a very good question. There are two historical reasons: efficiency considerations and support for non-contiguous masks.

Efficiency

The subnet mask expression is efficient, in that it allows routers to perform a quick masking operation to determine the subnet address. This improves performance; remember that computers were much slower when this system was developed. Today this is not really that much of an issue.

Support For Non-Contiguous Masks

RFC 950 actually specified that when splitting the bits in the host ID for subnet ID and host ID, it was possible to split it in more than one place! In the example above, we could, instead of splitting the 16 bits into 5 bits for subnet ID and 11 for host ID, have done it as 2 bits for the subnet ID, then 4 bits for the host ID, then 3 more bits for the subnet ID and finally 7 more bits for host ID. This would be represented by the subnet mask pattern “11000011 10000000” for those sixteen bits (following the sixteen ones for the network ID).

Why do this instead of just dividing as 5 bits for subnet ID followed by 11 bits for host ID as we did before? I have no idea. ☺ In fact, most other people had no idea either. Trying to subnet this way makes assigning addresses **extremely** confusing. For this reason, while it was technically legal, the use of non-contiguous subnet masking was not recommended, and not done in practice.

Given that non-contiguous masks are not used, and today's computers are more efficient, the alternative method of expressing masks with just a single number is now often used. Instead of specifying “IP address of 154.71.150.42 with subnet mask of 255.255.248.0”, we can just say “154.71.150.42/**21**”. This is sometimes called *slash notation* or *CIDR notation*. It is more commonly used in [variable-length masking \(VLSM\)](#) environments, and as the second name implies, is also [the standard for specifying classless addresses](#) under [the CIDR addressing scheme](#). However, it is also sometimes seen in regular subnetting discussions.



Note: Since these weird masks were never really used, some resources say that the subnet mask always had to be contiguous, but this is not true—originally, it was legal but “advised against”. Later this practice became so out-of-favor that many hardware devices would not support it. Today, now that classless addressing and CIDR are standard, it is simply illegal.

If you've never subnetted before, this topic may have left you with your head spinning, despite our looking at an example. Don't worry. It will all become more clear as you become more familiar with subnetting by following the rest of this section. Remember also that [I have included a whole section that shows how to subnet step by step](#), including determining the subnet mask.

IP Default Subnet Masks For Address Classes A, B and C

[Subnetting](#) is the process of dividing a Class A, B or C network into subnets, as we've seen in the preceding topics. In order to better understand how this “division of the whole” is accomplished, it's worth starting with a look at how the “whole” class A, B and C networks are represented in a subnetted environment. This is also of value because there are situations where you may need to define an unsubnetted network using subnetting notation.

This might seem like a strange concept—if you aren't going to bother creating subnets, why do you need to consider how the old-fashioned classes are used under subnetting? The answer is that after subnetting became popular, most operating systems and networking hardware and software were designed under the assumption that subnetting would be used. Even if you decide not to subnet, you may need to express your unsubnetted network using a subnet mask.

In essence, a non-subnetted class A, B or C network can be considered the “default case” of the more general, custom-subnetted network. Specifically, it is the case where we choose to divide the host ID so that zero bits are used for the subnet ID and all the bits are used for the host ID. I realize that this seems like a bit of a semantic game. However, this default case is the basis for the more practical subnetting [we will examine in the next topic](#).

Just as is always the case, the subnet mask for a default, unsubnetted class A, B or C network has ones for each bit that is used for network ID or subnet ID, and zeroes for the host ID bits. Of course, we just said we aren't subnetting, so there **are** no subnet ID bits! Thus, the subnet mask for this default case has 1s for the network ID portion and 0s for the host ID portion. This is called the *default subnet mask* for each of the IP address classes.

Since classes A, B and C divide the network ID from the host ID on octet boundaries, the subnet mask will always have all ones or all zeroes in an octet. Therefore, the default subnet masks will always have 255s or 0s when expressed in decimal notation. Table 51 summarizes the default subnet masks for each of the classes; they are also shown graphically in Figure 68.

Table 51: Default Subnet Masks for Class A, Class B and Class C Networks					
IP Address Class	Total # Of Bits For Network ID / Host ID	Default Subnet Mask			
		First Octet	Second Octet	Third Octet	Fourth Octet
Class A	8 / 24	11111111 (255)	00000000 (0)	00000000 (0)	00000000 (0)
Class B	16 / 16	11111111 (255)	11111111 (255)	00000000 (0)	00000000 (0)
Class C	24 / 8	11111111 (255)	11111111 (255)	11111111 (255)	00000000 (0)

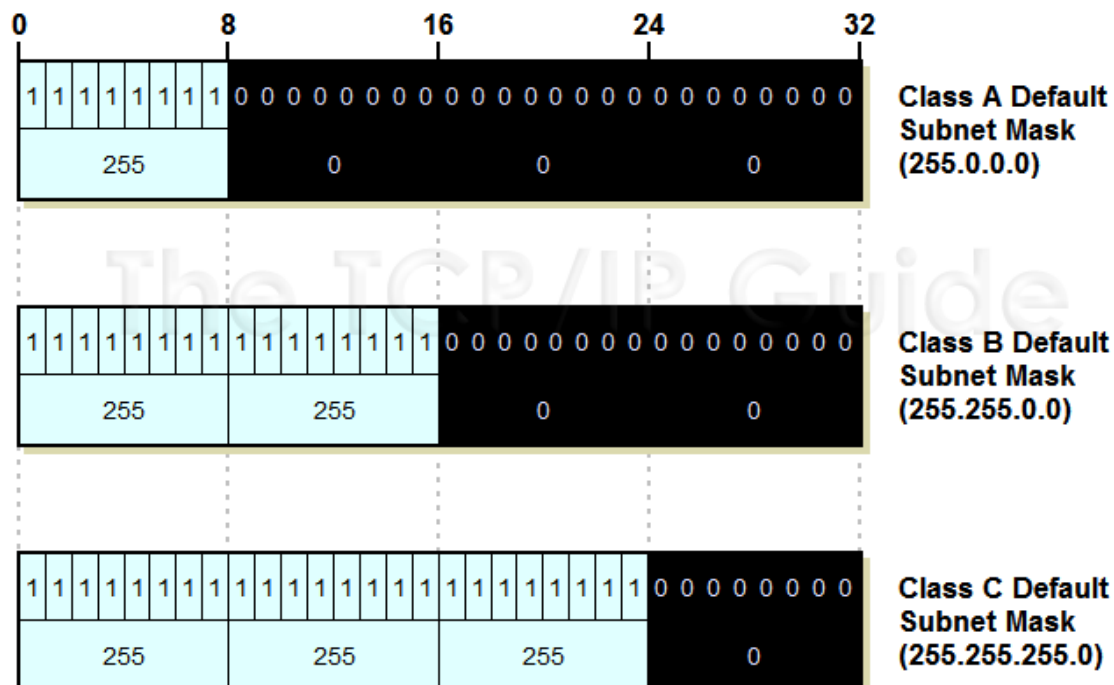


Figure 68: Default Subnet Masks for Class A, Class B and Class C Networks

So, the three default subnet masks are 255.0.0.0 for Class A, 255.255.0.0 for class B, and 255.255.255.0 for Class C. Note that while all default subnet masks use only “255” and “0”, not all subnet masks with “255” and “0” are defaults. There are a small number of custom subnets that divide on octet boundaries as well. These are:

- **255.255.0.0**: This is the default mask for Class B, but can also be the custom subnet mask for dividing a Class A network using 8 bits for the subnet ID (leaving 16 bits for the host ID).
- **255.255.255.0**: This is the default subnet mask for Class C, but can be a custom Class A with 16 bits for the subnet ID or a Class B with 8 bits for the subnet ID.



Key Concept: Each of the three IP unicast/broadcast address classes, A, B and C, has a *default subnet mask* defined that has a one for each bit of the class's network ID, a zero bit for each bit of its host ID, and no subnet ID bits. The three default subnet masks are 255.0.0.0 for Class A, 255.255.0.0 for class B, and 255.255.255.0 for Class C.

IP Custom Subnet Masks

It's important to understand what [default subnet masks](#) are and how they work. A default subnet mask doesn't really represent subnetting, however, since it is the case where we are assigning zero bits to the subnet ID. To do “real” subnetting we must dedicate at least one of the bits of the pre-subnetted host ID to the subnet ID, as shown in [the example in the topic that introduced subnet masks](#).

Since we have the ability to customize our choice of dividing point between subnet ID and host ID to suit the needs of our network, this is sometimes called *customized subnetting*. The subnet mask that we use when creating a customized subnet is, in turn, called a *custom subnet mask*. The custom subnet mask is used by network hardware to determine how we have decided to divide the subnet ID from the host ID in our network.

Deciding How Many Subnet Bits to Use

The key decision in customized subnetting is how many bits to take from the host ID portion of the IP address to put into the subnet ID. Recall that the number of subnets possible on our network is two to the power of the number of bits we use to express the subnet ID, and the number of hosts possible per subnet is two to the power of the number of bits left in the host ID (less two, which I will explain later in this topic).

Thus, the decision of how many bits to use for each of the subnet ID and host ID represents a fundamental trade-off in subnet addressing:

- Each bit taken from the host ID for the subnet ID doubles the number of subnets that are possible in the network.
- Each bit taken from the host ID for the subnet ID (approximately) halves the number of hosts that are possible within each subnet on the network.

Subnetting Bit Allocation Options

Let's take a brief example or two to see how this works. Imagine that we start with a Class B network with the network address 154.71.0.0. Since this is Class B, 16 bits are for the network ID (154.71) and 16 are for the host ID. In the default case there are no subnets (well, one "subnet" that is the whole network) and 65,534 hosts total. To subnet this network, we have a number of choices:

1. We can decide to use 1 bit for the subnet ID and 15 bits for the host ID. If we do this, then the total number of subnets is 2^1 or 2: the first subnet is 0 and the second is 1. The number of hosts available for each subnet is $2^{15}-2$ or 32,766.
2. We can use 2 bits for the subnet ID and 14 for the host ID. In this case, we double the number of subnets: we now have 2^2 or 4 subnets: 00, 01, 10 and 11 (subnets 0, 1, 2 and 3). But the number of hosts is now only $2^{14}-2$ or 16,382.
3. We can use any other combination of bits that add up to 16, as long as they allow us at least 2 hosts per subnet: 4 and 12, 5 and 11, and so on.

Trading Off Bit Allocations To Meet Subnetting Requirements

How do we decide how to divide the "classful" host ID into subnet ID and host ID bits? This is the key design decision in subnetting. We must make this choice based on our requirements for the number of subnets that exist in the network, and also on the maximum number of hosts that need to be assigned to each subnet in the network. For example, suppose we have 10 total subnets for our Class B network. We need 4 bits to represent this, because 2^4 is 16 while 2^3 is only 8. This leaves 12 bits for the host ID, for a maximum of 4,094 hosts per subnet.

However, suppose instead that we have 20 subnets. If so, 4 bits for subnet ID won't suffice: we need 5 bits ($2^5=32$). This means in turn that we now have only 11 bits for the host ID, for a maximum of 2,046 hosts per subnet. [Step #2 of the practical subnetting example discusses these decisions in more detail.](#)

Now, what happens if we have 20 subnets and also need a maximum of 3,000 hosts per subnet? Well, we have a problem. We need 5 bits to express 20 different subnets. However, we need 12 bits to express the number 3,000 for the host ID. That's 17 bits—too many. The solution? We might be able to shuffle our physical networks so that we only have 16. If not, we need a second Class B network.

It's also important to realize that in regular subnetting, the choice of how many bits to use for the subnet ID is fixed for the entire network. You can't have subnets of different sizes—they must all be the same. Thus, the number of hosts in **the largest subnet** will dictate how many bits you need for the host ID. This means that in the case above, if you had a strange configuration where 19 subnets had only 100 hosts each but the 20th had 3,000, you'd have a problem. If this were the case, you could solve the problem easily by dividing that one oversized subnet into two or more smaller ones. An enhancement to subnetting called [Variable Length Subnet Masking \(VLSM\)](#) was created in large part to remove this restriction.



Note: I have included [summary tables](#) that show the trade-off in subnetting each of Classes A, B and C, and the subnet mask for each of the choices.



Key Concept: The fundamental trade-off in subnetting: each addition of a bit to the subnet ID (and thus, subtraction of that bit from the host ID) doubles the number of subnets, and approximately halves the number of hosts in each subnet. Each subtraction of a bit from the subnet ID (and addition of that bit to the host ID) does the opposite.

Determining the Custom Subnet Mask

Once we have determined how many bits we are going to devote to the subnet ID and the host ID, we can figure out the subnet mask. This is fairly easy to do, now that we understand how subnetting works. We begin with the default subnet mask in binary for the appropriate class of our network. We then start with the left-most zero in that mask and change as many bits to 1 as we have dedicated to the subnet ID. We can then express the subnet mask in dotted decimal form. Figure 69 shows how the custom subnet mask can be determined for each of the subnetting options of a Class C network, in both binary and decimal.

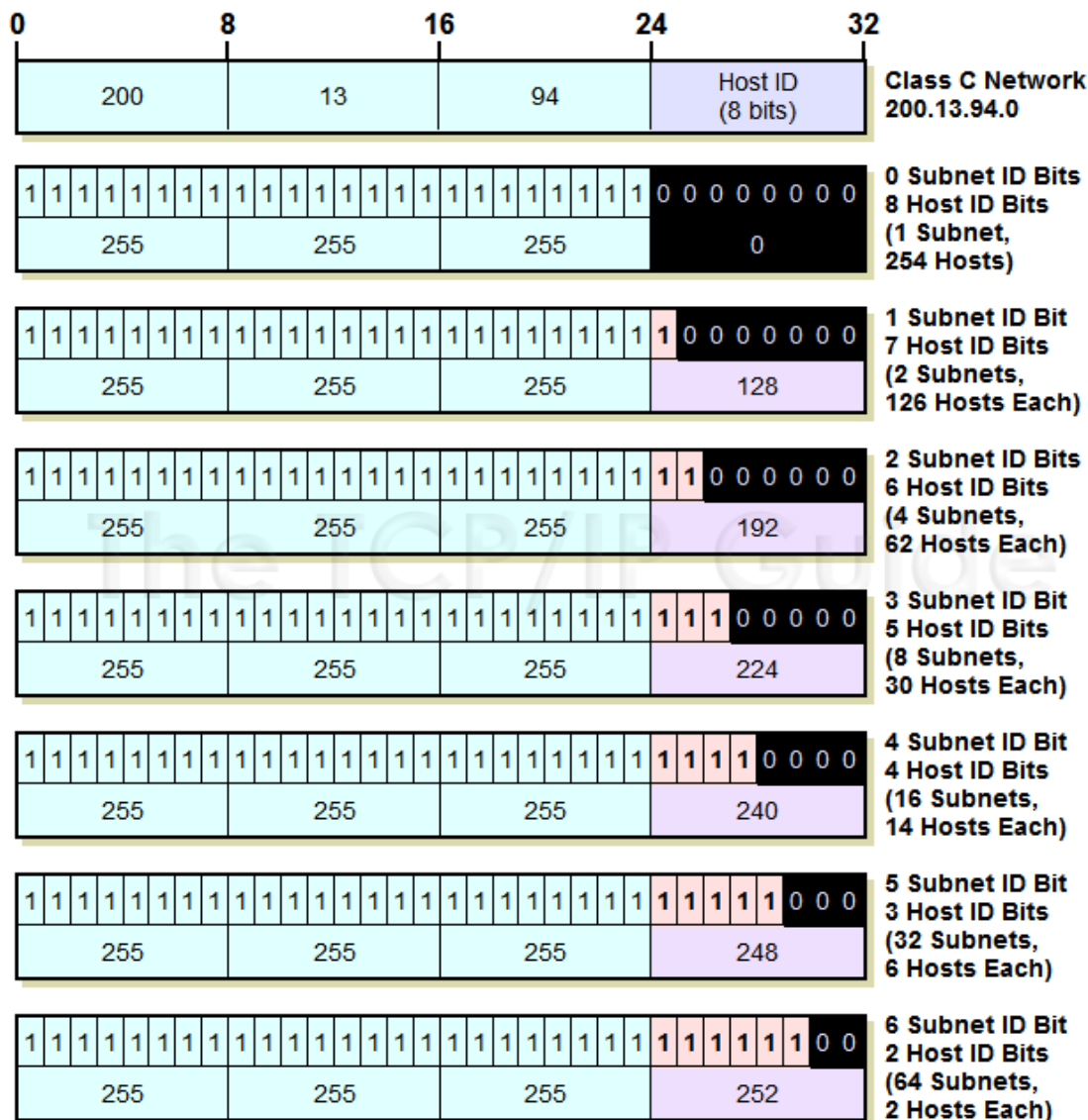


Figure 69: Custom Subnet Masks for Class C Networks

Since there are 8 host ID bits in a Class C network address, there are six different ways that the network can be subnetted. Each corresponds to a different custom subnet mask, which is created by changing the allocated subnet ID bits from zero to one.

So, to take the example in that figure, consider the Class C network 200.13.94.0. There are 8 bits in the original host ID, which gives us six different subnetting options (we can't use 7 or 8 bits for the subnet ID, for reasons we will discuss shortly.) Suppose we use three of these for the subnet ID and five are left for the

host ID. To determine the custom subnet mask, we start with the Class C default subnet mask:

11111111 11111111 11111111 00000000

We then change the first three zeroes to ones, to get the custom subnet mask:

11111111 11111111 11111111 11100000

In dotted decimal format, this is 255.255.255.224.



Key Concept: Once the choice of how to subnet has been made, the custom subnet mask is determined simply, by starting with the default subnet mask for the network and changing each subnet ID bit from a 0 to a 1.

Subtracting Two From the Number of Hosts Per Subnet and (Possibly) Subnets Per Network

There's one more issue that needs to be explained regarding the split into subnet ID and host ID. We've already seen how in regular "classful" addressing, we must subtract 2 from the number of hosts allowed in each network. This is necessary because two host IDs in each subnet have "special meanings": the all-zeroes host ID meaning "this network", and the all-ones host ID which is a broadcast to "all hosts on the network". These restrictions apply also to each subnet under subnetting too, which is why we must continue to subtract 2 from the number of hosts per subnet. (This is also why dividing the 8 host ID bits of a Class C network into 7 bits for subnet ID and 1 bit for host ID is not just silly, but in fact meaningless: it leaves $2^1 - 2 = 0$ hosts per subnet. Not particularly useful.)

There is a similar issue that occurs with the subnet ID as well. When subnetting was originally defined in RFC 950, the standard specifically excluded the use of the all-zeroes and all-ones subnets. This was due to concern that routers might become confused by these cases. A later standard, RFC 1812 (Requirements for IP Version 4 Routers) removed this restriction in 1995. Thus, modern hardware now has no problem with the all-zeroes or all-ones subnets, but some very old hardware may still balk at it.



Key Concept: The number of hosts allowed in each subnet is the binary power of the number of host ID bits remaining after subnetting, **less two**.

The reduction by two occurs because the all-zeroes and all-ones host IDs within each subnet are reserved for two "special meaning" addresses: to refer to the subnetwork itself and its local broadcast address. In some implementations, the number of subnets is also reduced by two because the all-zeroes and all-

ones subnet IDs were originally not allowed to be used.

For this reason, you will sometimes see discussions of subnetting that exclude these cases. When that is done, you lose 2 potential subnets: the all-zeroes and all-ones subnets. If you do this, then choosing 1 bit for subnet ID is no longer valid, as it yields $2^1 - 2 = 0$ subnets. You must choose 2 bits if you need 2 subnets.



Note: In this Guide I assume we are dealing with modern hardware and do not exclude the all-zeroes and all-ones subnets, but I do try to make explicit note of this fact wherever relevant.