# *MET CS CS625 Business Data Communications and Networks*
# *Lab 3 – Subnet Masks*

## OBJECTIVES

The objective of this lab is to learn to work with subnet masks in the IPv4 protocol. Previously we learned how to represent IPv4 addresses in dotted decimal notation, and how to specify the length of the network identifier in CIDR notation. We also learned how to determine if two IPv4 addresses are on the same network. In this lab, we are going to learn the specific operations that a computer uses to determine if two IPv4 addresses are on the same network, which involves subnet masks.

## PREREQUISITES

Before attempting this lab, it is best to read the textbook and lecture material for the week. While this lab shows you how to work with IPv4, the lab does not explain in full the framework and theory behind subnet masks, with the assumption that you will first read the lecture and textbook material.

## LAB SUBMISSION

Use the submission template provided in the assignment inbox to perform the steps requested by this lab. Return to the assignment inbox to submit your lab.

## OVERVIEW

As we learned previously, computers and computer networks deal with addresses in raw bits, while we as human beings find it easier to represent the addresses in other notations. It follows that we find it easier to determine if two IPv4 addresses are on the same network using a different method than computers. In particular, we have learned to determine if two IPv4 addresses are on the same network by calculating the network addresses for both, represent them in decimal notation, then comparing the two. Computers do not have the luxury or the need to represent the addresses in decimal notation for this comparison, and instead use a *subnet mask* and the *bitwise AND* operator to calculate the network addresses for comparison. You will learn more about both of these the steps below.

## STEPS

1. A subnet mask is a 32-bit binary number that has the 1 bit set for each bit that corresponds to the network identifier for its associated IPv4 address, and the 0 bit set for each bit that corresponds to the host identifier. For example, if the length of the network identifier for an IPv4 address is 20 bits, then the corresponding subnet mask would be a 32-bit binary number consisting of 20 1s followed by 12 0s, like so:

   11111111111111111111000000000000

   The number of bits for the network identifier directly determines the number of 1s and 0s that appear in the subnet mask. More specifically, when configuring a network adapter, the subnet mask itself is the only way the computer knows how to calculate the network address corresponding to an IPv4 address.

   If we are given a CIDR entry, say, 198.255.255.32/24, what would the subnet mask be? Simple! We just list out 24 1s followed by 8 0s.

   11111111111111111111111100000000

2. For each CIDR entry below, list its subnet mask as a binary number.

   10.2.9.33/17
   192.168.5.3/16

3. Just as IPv4 addresses are commonly represented in dotted decimal notation, so are subnet masks. We can follow all of the same steps as we did when converting IPv4 addresses from binary to dotted decimal. You are already well familiar with this type of conversion, though you should also be aware of a shortcut we can use. If an entire octet consists of all 1 bits, we do not need to manually convert that octet, because we know its decimal representation is 255.

   A subnet mask will always consist of at least three octets that are fully spanned by 1 bits, or fully spanned by 0 bits, and we know the equivalent decimal numbers are 255 and 0, respectively. In the event that the length of the network identifier not a power of 2 (8, 16, or 24), then there will be one octet that is partially spanned with some 1s and some 0s. And it is only the partially spanned octet that we need to convert to decimal to correctly represent the subnet mask in dotted decimal notation.

   Let’s take an example, 208.64.121.161/14. From this CIDR entry, we know that the network identifier spans 14 bits, and the corresponding subnet mask can be represented as follows in binary.

11111111 11111100 00000000 00000000

The four octets have been separated in the above example.

Notice that the 1s fully span the first octet and partially span the second, and that the third and fourth octets consists of all 0s. We can now represent the full subnet mask simply by following the shortcut mentioned above, and converting the second octet to decimal. The shortcut tells us that the mask is 255.X.0.0, so we need only determine X to obtain the full dotted decimal notation.

11111100 ➔ 252

| Power of 2 | | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| Bit | | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| Cumulative Amount | 0 | 128 | 192 | 224 | 240 | 248 | 252 | 252 | 252 |

Replacing X with 252, the subnet mask is 255.252.0.0.

You may have noticed something interesting when converting the partial octet. Since the 1 bits must appear consecutively followed by consecutive 0 bits, there are actually only nine possibilities for any given octet in a subnet mask, as defined in the table below.

| | | | |
|---|---|---|---|
| 00000000 = 0 | 10000000 = 128 | 11000000 = 192 | 11100000 = 224 |
| 11110000 = 240 | 11111000 = 248 | 11111100 = 252 | 11111110 = 254 |
| 11111111 = 255 | | | |

If we remember these nine conversions, we do not even need to perform a manual conversion of the partial octet.

4. You give it a try with the following CIDR entries. Represent the subnet mask for each in dotted decimal notation following the methodology used in step 3. Use the lookup table below to convert the partial octet for each mask (if any).

| | | | |
|---|---|---|---|
| 00000000 = 0 | 10000000 = 128 | 11000000 = 192 | 11100000 = 224 |
| 11110000 = 240 | 11111000 = 248 | 11111100 = 252 | 11111110 = 254 |
| 11111111 = 255 | | | |

23.3.96.50/19
157.166.226.25/16

5. Now that we understand subnet masks, we can now learn about the bitwise AND operator, which is what computers use to calculate the network address corresponding to an IPv4 address. So that we understand the phrase better, let us

break the phrase “bitwise AND operator” down piece by piece. First, an *operator* transforms one or more items into a new item. For example, the plus operator in mathematics, represented by the “+” sign, can be used transform two numbers into a new number by adding them together. A *bitwise* operator, as its name suggest, operates on individual bits of binary numbers. A bitwise *AND* operator compares each bit of the first binary number to the corresponding bit in the second binary number, and yields a 1 if both bits are 1, and a 0 otherwise. The bitwise AND operator is often represented mathematically with the “&” symbol.

You may find useful the following table of all possible bitwise AND operations between two bits:

| Bit A | Bit B | A & B |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

Let us view a simple example, followed by a more complex example. If we were to apply the bitwise AND operator to two binary numbers – 1101 and 0101 – the result would be computed as follows.

```
  1101
& 0101
----------
  0101
```

Did you catch what just happened? Starting from left to right:

1 & 0 = 0
1 & 1 = 1
0 & 0 = 0
1 & 1 = 1

The result is thus 0101 from this bitwise AND operation.

6. Now you give it try with the following two simple examples.

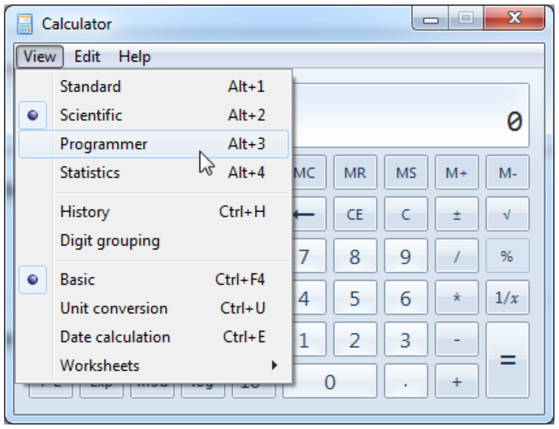
```
  1010
& 0101

  1111
& 0111
```

7. Why does any of this matter? It matters because computers and devices perform the bitwise AND operation each and every time they need to send an IP packet, in order to determine if the destination node is on its same network. In the world today, there are literally billions of networked computers and devices that continuously perform the bitwise AND operation in order to communicate!

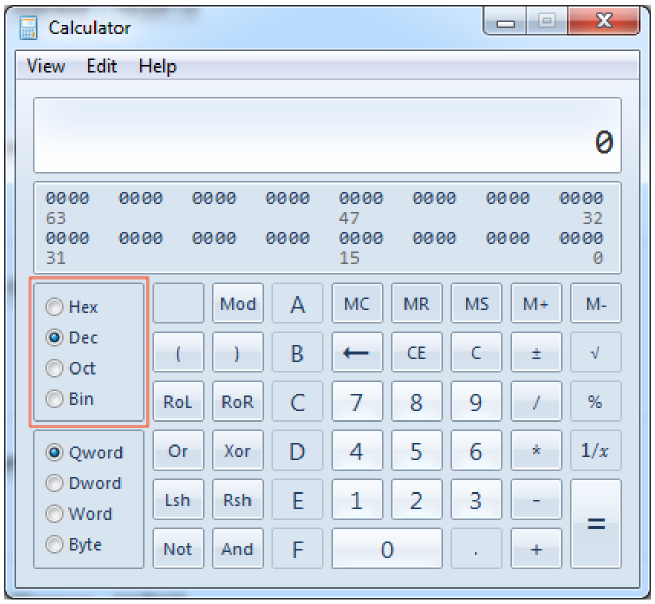
   Let's start with the way a computer computes a single network address. We can use the example CIDR entry from step 3, 208.64.121.161/14. We already expressed the subnet mask in binary, but we will also need to represent 208.64.121.161 in binary as well.

   In the first two labs, you are required to manually convert each decimal number into a binary number and vice versa. It is very important that you know how to perform these calculations by hand. However, when working in the real world and in academia, we often use tools, most commonly a calculator, to do many conversions for us. Think of this as akin to learning how to perform basic addition, subtraction, multiplication, and division by hand so that you understand the concepts and can do so when no tool is available, but after you understand how to do so you often use a calculator to save yourself some time.

   For example, in modern versions of Windows, one can switch the built-in calculator into "Programmer" mode like so.



   After doing so, you will notice that there are options to switch the calculator to a base other than base 10, including Hexadecimal (Hex), Octal (Oct), and Binary (Bin).

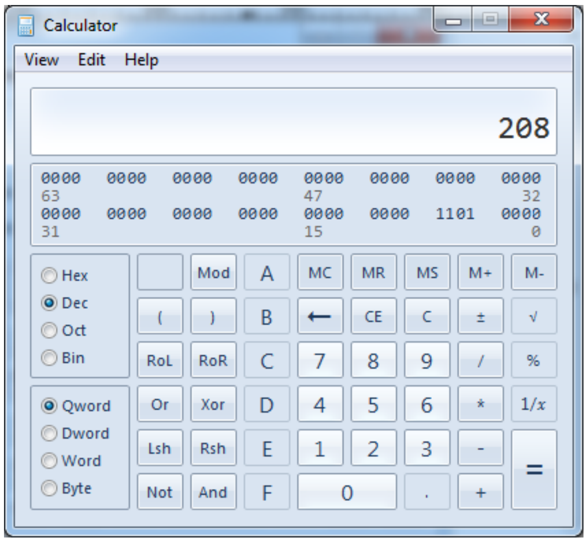


```
                                                0

0000   0000   0000   0000   0000   0000   0000   0000
63                          47                     32
0000   0000   0000   0000   0000   0000   0000   0000
31                          15                      0
```

Hex  [✓ Dec]  Oct  Bin

[✓ Qword]  Dword  Word  Byte

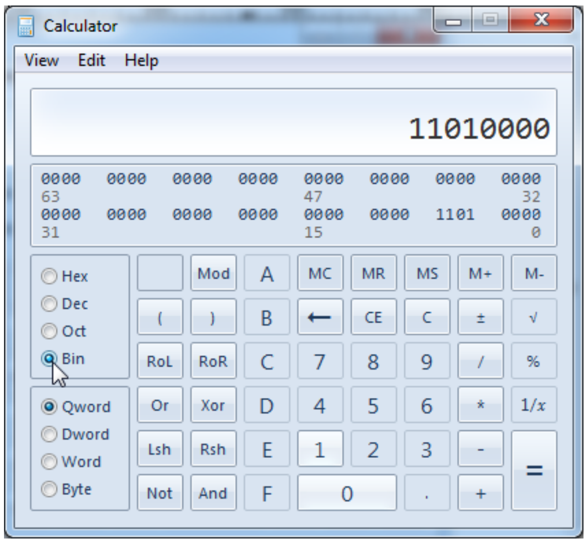
If you enter a number in one base, then switch to the other base, it will automatically convert the number for you. For example, to convert the first octet of the IPv4 address 208.64.121.161, we would enter 208 as decimal, then click “Bin”, to show its binary representation.



```
                                              208

0000   0000   0000   0000   0000   0000   0000   0000
63                          47                     32
0000   0000   0000   0000   0000   0000   1101   0000
31                          15                      0
```

Hex  [✓ Dec]  Oct  Bin

[✓ Qword]  Dword  Word  Byte

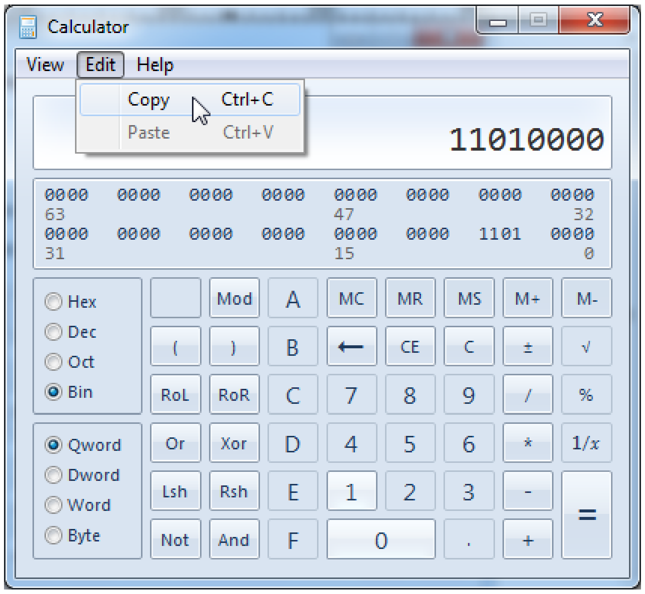


```
                                         11010000

0000   0000   0000   0000   0000   0000   0000   0000
63                          47                     32
0000   0000   0000   0000   0000   0000   1101   0000
31                          15                      0
```

Hex  Dec  Oct  [✓ Bin]

[✓ Qword]  Dword  Word  Byte

We can then simply use the Edit/Copy feature to paste it into our document.



```
0000  0000  0000  0000  0000  0000  0000  0000
63                      47                  32
0000  0000  0000  0000  0000  0000  1101  0000
31                      15                   0
```

Keep in mind while doing so that *the calculator removes any leading 0 bits*, so you will need to manually add leading 0 bits yourself. In other words, the calculator does not know we are specifically working with octets, and so will not bother put any leading 0 bits since lead 0 bits to not change the number's numeric value.

From this step forward and in future labs, you are free to use a tool to perform these binary to decimal or decimal to binary conversions for you, unless you are specifically asked to perform the conversion by hand. By-hand conversions will be sprinkled throughout the remaining labs to help ensure you remember how to do so.

The IPv4 address 208.64.121.161 is 11010000010000000111100110100001 in binary. We can now perform the bitwise AND operation between the IPv4 address and it subnet mask, as follows. Note that the example below separates the octets for readability.

```
  11010000 01000000 01111001 10100001
& 11111111 11111100 00000000 00000000
-------------------------------------
  11010000 01000000 00000000 00000000
```

Did you catch what just happened? By the nature of the bitwise AND, anywhere the subnet mask has a 1 bit, the corresponding bit from the IPv4 address is copied into the result. Anywhere the subnet mask has a 0 bit, it does not matter what the corresponding bit in the IPv4 address is, because the resulting bit is always 0. Thus, *the subnet mask is the computer's tool to copy the network identifier from the IPv4 address, and zero out the host portion of the IPv4 address, the result being the* *network address of the IPv4 address*. Stop and think about the last sentence until you are sure you understand it, as the concept it expresses is integral to understanding the value and purpose of a subnet mask.

8. Now, give it a try on each of the following CIDR entries, by performing the same steps illustrated in step 7 to calculate the network address corresponding to its IPv4 address.

   137.254.120.47/25
   204.79.197.200/12

9. Now that we have tried out calculating the network address using a subnet mask, mirroring what computers do, we can take the next step of comparing the network addresses to each other to see if they are on the same network.

   What is the significance of being on the same network versus being on a different network? If the sender and receiver are on the same layer-3 network, the sending computer knows that, at the data-link layer, it can directly address the recipient. Otherwise, the sending computer will address the data-link layer frame to a router who knows how to route the packet to the recipient's network. This behavior illustrates and emphasizes that *all computers and devices must utilize the data-link layer to send all communications*. An IPv4 packet cannot be sent without first being embedded in a data-link layer frame.

   The data-link layer knows nothing about networks, and is only capable of sending frames to directly reachable network adapters. It is up to the sending node to determine to whom the data-link layer frame must be addressed. It is for this reason that each time a computer needs to transmit an IPv4 packet, it must first determine if it will transmit the packet directly to the recipient on the same network, or if it will transmit it to a router who will route the packet to another network. The data-link layer will not perform this logic on behalf of the sending computer.

   If a computer's IPv4 address and network identifier length were given as 192.168.254.5/23, and it intended to send to another node with IPv4 address and length 192.168.255.39/23, how would the computer know if the two are on the same network? Simple! It would calculate the network addresses for both and compare them. If the network addresses are equal, they are on the same network; otherwise, they are not.

So we start with the first entry, 192.168.254.5/23.

```
  11000000 10101000 11111110 00000101
& 11111111 11111111 11111110 00000000
---------------------------------------------------------
  11000000 10101000 11111110 00000000
```

And now the second entry, 192.168.255.39/23.

```
  11000000 10101000 11111111 00100111
& 11111111 11111111 11111110 00000000
---------------------------------------------------------
  11000000 10101000 11111110 00000000
```

Now we simply compare both network addresses.

```
11000000 10101000 11111110 00000000
11000000 10101000 11111110 00000000
```

When we look through them bit by bit, we see that they are in fact equal. Thus, 192.168.254.5/23 and 192.168.255.39/23 are on the same network.

10. A sending computer will often compare its own network address to hundreds or even thousands of other network addresses, because computer users and applications are oftentimes used to communicate to hundreds or thousands of other nodes. For example, how many different websites do you visit in a week? Probably plenty! As well, the IP address and subnet mask on a network adapter does not typically change often, so a sending node usually finds itself comparing the same network address (its own) to many other network addresses.

    Now you have a chance to try out the methodology used in step 9 with a similar scenario. Imagine that that the only network adapter on a computer has an IP address of 63.240.110.201 and that its network identifier spans 21 bits – 63.240.110.201/21. Further imagine that the computer wants to send a message to some other computers with the following CIDR entries.

    63.240.110.219/21
    63.240.104.12/21
    63.240.102.35/21
    67.119.61.37/21

    First calculate the sending computer's subnet mask and network address by using the methodology listed in step 9. Then, for each of the CIDR entries above, use the methodology listed in step 9 to determine if each entry is on the same network or a different network than the sending computer. You may use a tool to perform the decimal to binary conversions as needed, but you must otherwise show your work for full credit, and should use the computer's methodology (illustrated in step 9) by using subnet masks.

11. In a paragraph or two, explain in your own words how the use of a subnet mask and the bitwise AND operation enables a computer to determine if another IPv4 address is on the same network.

Congratulations! You have learned the way that billions of computers and devices determine where to send a data-link layer frame when it intends to communicate with another node at the network layer.