12
Understanding IPv6

Internet Protocol version 6 (**IPv6**) is becoming more and more prominent. Despite its slow take-up initially, the ever-increasing demand for IP addresses will no doubt make organizations that have been hesitant to implement IPv6 reconsider their stance. I think with the abundance of the **Internet of Things** (**IoT**) devices that are now available, this demand will continue to increase dramatically and having knowledge of IPv6 will become more important.

This chapter introduces you to IPv6, giving you an overview of how IPv6 addresses are represented in general, before moving on to discuss the various address types. Then, we will cover assigning IPv6 addresses in a Windows environment. We will finish this chapter by briefly talking about intercompatibility between IPv4 and IPv6, which, until we are solely using IPv6, will continue to be an important consideration.

The following topics will be covered in this chapter:

- Overview of IPv6
- Understanding address types
- Understanding assigning IPv6 addresses
- Understanding interoperability with IPv4

Technical requirements

To complete the exercises in this chapter, you will need an internet-connected PC running Windows 7 or above, preferably Windows 10.

Overview of IPv6

Let me start this chapter with a question. If IPv4 is so widespread and works well, why has IPv6 been released? Quite simply, we've run out (or are running out) of IPv4 addresses. Yes, you read that right. We have run out of 4.2 billion addresses. You might be wondering how this is possible, so let's have a look at some figures.

As of June 2019, Internet World Stats published figures that stated the world population was 7.7 billion (sourced from the United Nations Population Division), and of those, 4.4 billion people were internet users (sourced from the **International Telecommunications Union** (**ITU**)). Admittedly, quite a number of those 4.4 billion will be sharing an internet-connected device, such as a home PC, and that will be their only means of connection. But if we flip that, there will be a large number of people who have multiple devices that connect directly to the internet. As an example, I'm currently sitting at the train station typing this up on my internet-connected tablet, with my internet-connected phone next to me. That's two IP addresses gone from me.

We can also add that IPv4 uses IP addresses inefficiently into the mix. Remember, there are certain IPv4 addresses or address ranges that we cannot issue to hosts. This eats into those 4.2 billion addresses. For example, no hosts can be allocated from the following:

- Networks starting with `0`: 16,777,216 hosts
- Networks starting with `127`: 16,777,216 hosts
- Networks starting with `169.254`: 65,536 hosts
- Any Class D network: 268,435,456 hosts
- Any Class E network: 268,435,456 hosts

To combat this, IPv6 was introduced. IPv6 is a 128-bit hexadecimal address that offers a staggering 340 undecillion addresses (2^128), or to write it in full, 340,282,366,920,938,463,463,374,607,431,768,211,456 addresses.

If you want to know how to read that out loud, I have included a link in the *Further reading* section.

Although the extra addresses that IPv6 provides us are, of course, of massive benefit to us, this is not the only benefit that was brought about by the introduction of this new addressing scheme. When IPv6 was being developed, the developers looked at what was wrong with or missing from IPv4 and, where possible, rectified or included it in IPv6. For example, they changed the header format, not only to support the new address system but also to make it as efficient as possible. They achieved this by creating an IPv6 header that was 40 bytes long, compared to IPv4's header, which was 20 bytes long. This isn't bad when you consider that, by just looking at the address sizes, an IPv4 address is 4 bytes long (32 bits) and an IPv6 address is 16 bytes long (128 bits). Here, the header has to have a source and destination address, and you can see that an IPv6 header has 24 bytes more being used for addressing compared to its IPv4 counterpart. How have they squeezed it in, then? Well, an IPv4 header uses 12 bytes for non-addressing information, whereas an IPv6 header has stripped that back to only 8 bytes of non-addressing information.

I mentioned previously that an IPv6 address is a hexadecimal number, so before we talk about the format of the address, it would be worthwhile to understand what hexadecimal actually is.

Hexadecimal numbering

The hexadecimal numbering system is also known as base-16. What confuses a lot of people is that it hexadecimal numbers also include letters. Yes, you read that correctly. There are letters in there, but we actually refer to them as **symbols**. Hexadecimal uses the numbers 0-9 and symbols A-F to represent any decimal number that we wish to throw at it. Counting up to and including 15 is relatively straightforward; 0-9 is represented by 0-9 as usual, but 10-15 are represented by symbols, so we would get the following:

Decimal	Hexadecimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B

12	C
13	D
14	E
15	F

When we want to count over 16, we need to combine the characters to continue. Have a look at the following table, where I have continued this:

Decimal	**Hexadecimal**
16	10
17	11
18	12
19	13
20	14
21	15
22	16
23	17
24	18
25	19
26	1A
27	1B
28	1C
29	1D
30	1E
31	1F
32	20

While not as simple to read as the decimal and binary tables we have used previously in this book, we could create a similar one for hexadecimal:

65536	**4096**	**256**	**16**	**Units**

With just the units column, we cover the range 0-255; by adding the 16 column, we can cover 0-255; the 256 column gives us 0-4,095; the 4,096 column gives us 0-65,535; and finally (in this example, anyway), the 65,536 column gives us 0-1,048,575. The pattern here between the column headings is that each column value is 16 times the column value to the right of it. You'll be pleased to know that, for networking purposes, generally, just having the two right-most columns is enough for us.

I want to finish off this section on hexadecimal with just a quick explanation of how we convert a hexadecimal value into binary. Although it is not part of the exam, having this knowledge forms a good foundation that you can build upon when you continue to upskill your networking knowledge.

Converting hexadecimal into binary

Each hexadecimal character is made up of 4 bits, so if we have a single character, we can quite simply convert it into its decimal equivalent and then convert that into binary, like we did in the previous chapter. The following table provides a mapping of single hexadecimal characters and their binary and decimal equivalents:

Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

That is pretty straightforward, but what if you have multiple hexadecimal characters? Would you believe it is as simple as doing each character separately and merging the results? Let's look at a couple of quick examples:

- Hex = A9
- A = 1010
- 9 = 1001

Merging the two gives us `10101001`:

- Hex = AC4
- A = 1010
- C = 1100
- 4 = 0100

Merging the three gives us `101011000100`.

Notice I have kept the leading zero in the binary representation of the hexadecimal value 4. This is important: do not remove leading zeros when converting.

Now, we know how to convert hexadecimal into binary, let's reverse the process and convert binary into hexadecimal. When reversing the process, we need to split the binary into groups of 4 bits. Always split from the right:

- *Binary = 110111001011*
- *Splitting = 1101 1100 1011*
- *1101 = D*
- *1100 = C*
- *1011 = B*

Merging these gives us a hexadecimal value of DCB.

I'm going to give you one last example. In this case, the binary does not split up evenly into groups of 4. We're going to convert the binary value of `1010010111`. If we split that into groups of 4, remembering to start from the right, we get `10 1001 0111`.

Notice that we have a group of only two at the start? All we need to do is pad it out with leading zeroes to make it up to a 4 character number, like this: `0010`.

Once you get familiar with converting between the two, you will most likely stop padding because you know how to interpret it. Let's finish off the conversion:

- *Binary = 1010010111*
- *Splitting = 10 1001 0111*
- *Padding = 0010 1001 0111*
- *0010 = 2*
- *1001 = 9*
- *0111 = 7*

Merging the three gives us a hexadecimal value of *297*.

If you want to practice converting between hexadecimal and binary a bit more, I have included some links to online converters in the *Further reading* section.

I'd like to finish this section by briefly mentioning how we can differentiate between each numbering system. It is quite obvious when we see a number containing symbols, for example, 1C, that this is a hexadecimal number. However, what if we see the number 11? Is this a hexadecimal, binary, or decimal value? We cannot tell just from the number alone. Unfortunately, there is no hard and fast method of doing this, and it can vary depending on what underlying programming language is being used. However, hexadecimal is usually prefixed with `0x`. You may see binary prefixed with `0b` but this is unusual, and decimal does not have a prefix.

The format of an IPv6 address

We have already ascertained that IPv6 addresses are 128 bits long and are made up of hexadecimal numbers. In this section, we will look at the format of the address.

An IPv6 address is formed out of 8 groups of 4 hexadecimal numbers, with each group separated by a colon, that is, a `:` character. Each character in the address is worth 4 bits, so each group of hexadecimal numbers is worth 16 bits or 2 bytes. To round off the math here, 8 groups of 16 bits equals 128 bits.

Note that, unofficially, these groups are referred to as hextets because they are 16 bits, but this is unlikely to be mentioned in the exam. Also, I have heard people refer to the groups as octets as a throwback to IPv4, but this is 1) not a correct term and 2) will not be in the exam.

Let's look at an example IPv6 address:

```
2001:0034:09FA:F3B2:20E4:1030:0001:4BC2
```

There we go—it just rattles off the tongue, doesn't it? OK, I have to admit, it is quite laborious writing these down. Fortunately, there are some rules we can follow, and we will look at them now.

Dropping leading zeroes

The one rule that everyone seems to remember is, in each group of numbers, we can drop the leading zeroes. Using our preceding example, I have emboldened the leading zeroes:

2001:**00**34:**0**9FA:F3B2:20E4:1030:**000**1:4BC2

Removing them would give us the following legitimate IPv6 address:

```
2001:34:9FA:F3B2:20E4:1030:1:4BC2
```

The important thing to note is that it is only leading zeroes, not intermediate nor trailing zeroes. I'll give you a couple of examples to highlight why we specify only leading. Look at the sixth group of numbers:

```
1030
```

If we could remove leading AND trailing zeroes, we would be left with the following:

```
103
```

We would have no way to find out whether the original number was `0103` or `1030`. This would be even worse if were allowed to removed intermediate zeroes also. We would be left with the following number:

```
13
```

Wow; this is much shorter, but what was the original number? All the device would know is there are two zeroes to fit in somewhere. Was it `0013`, `0103`, `0130`, `1300`, or `1003`?

Dropping contiguous zeroes once

The first time I heard the term *contiguous*, I thought I had misheard the instructor and they had said *continuous*, but no, contiguous it was. With that in mind, I think it is important to briefly define what the word actually means, although you may still think it the same as continuous. Contiguous means adjacent or being in contact with. This will become clearer when I go through an example.

Take the following IPv6 address:

```
2001:0034:09FA:0000:0000:1030:0001:4BC2
```

In the center of it, we can see that it has two groups of numbers made up solely of zeroes, and these two groups are adjacent to each other. They are contiguous, so we can drop them:

```
2001:0034:09FA:0000:0000:1030:0001:4BC2
```

This leaves us with the following:

```
2001:0034:09FA::1030:0001:4BC2
```

Notice that, in their place, there is double colon `::`; which indicates where in the address the zeroes originally were. Once we have done that, we can also drop the leading zeroes:

```
2001:0034:09FA::1030:0001:4BC2
```

This leaves us with `2001:34:9FA::1030:1:4BC2`.

Ahh, that's better. This is much shorter.

I would like to give a further example, to expand on this a little. Let's look at the following address:

```
2001:0034:0000:0000:0000:1030:0001:4BC2
```

In this example, we now have three contiguous groups of zeroes, and we can drop all of these:

```
2001:0034:0000:0000:0000:1030:0001:4BC2
```

This leaves us with `2001:0034::1030:0001:4BC2`.

How many colons did I replace the contiguous zeroes with? Yes, just the two colons again, and this is why I wanted to provide a further example. It does not matter how many groups of contiguous groups of zeroes are removed – they are only replaced with the one set of double colons.

Of course, we can then drop the leading zeroes that remain:

```
2001:0034::1030:0001:4BC2
```

This would leave us with `2001:34::1030:1:4BC2`.

Let's look at one final example. Consider the following IPv6 address: `2001:0034:0000:0000:AB76:0000:0000:4BC2`. What do you notice about it?

Hopefully, you will have spotted that there are two sets of contiguous groups of zeroes:

`2001:0034`**`:0000:0000:`**`AB76`**`:0000:0000:`**`4BC2`

You might be tempted to shorten it to `2001:0034::AB76::4BC2`.

Unfortunately, you cannot do this, as the device does not know how many sets of zeroes each double colon represents. Was it a group of three and a single group, two groups of two, or a single group and a group of three? Because of this, you can only drop one set of contiguous zeroes. So, either of these would be allowed:

`2001:0034::AB76:0000:0000:4BC2` or `2001:0034:0000:0000:AB76::4BC2`.

OK, this doesn't look too amazing, but remember we can drop leading zeroes, including the leading zeroes in the groups that consist solely of zeroes: `2001:`**`00`**`34::AB76:`**`000`**`0:`**`000`**`0:4BC2`.

This gives us `2001:34::AB76:0:0:4BC2` or `2001:`**`00`**`34:`**`000`**`0:`**`000`**`0:AB76::4BC2`, and finally, this gives us `2001:34:0:0:AB76::4BC2`.

We have just seen the format an IPv6 address can take and have looked at the various methods of shortening the addresses to make them a little more user-friendly. As you can see, the address format is vastly different to that of IPv4. Likewise, the method of displaying subnets is different, as you will see in the next section.

Subnets and prefixes

Just like IPv4, IPv6 addresses use a similar concept of subnetting. However, we do not use the dotted decimal notation. Instead, we use the slash notation we used in CIDR, and this is referred to as a prefix. Like CIDR, the notation refers to how many bits of an IPv6 address the network element uses. The remainder, like IPv4, is set aside for the host element.

The term prefix is used here as it refers to the bits at the start of the IP address indicating the network element. Fortunately, the MTA exam does not interrogate you much on IPv6 subnets and prefixes—at least not to the same degree as it does for IPv4. As we go through the various addresses, it will be sufficient just to know what prefixes go with what address type.

Transmission types

With IPv4, we had three types of transmissions: unicast, multicast, and broadcast. IPv6 has similar transmission types, but drops broadcast and introduces anycast transmissions.

The easiest way to understand anycast is with an analogy. Imagine you have seen a crime being committed. Do you have to tell every police officer (broadcast)? Do you have to tell a particular police officer (unicast)? Or do you have to tell the first police officer that you see (anycast)? Anycast transmissions are sent to the nearest device from a predefined group of devices. Usually, anycast addresses are allocated to routers only.

Before we move on to the next section, I would just like to reiterate how important it is for you to understand the format of an IPv6 address. Given the length of the addresses, it is very easy for a *typo* to creep in, so knowing how the address is written in its various formats, and which characters are valid, will be beneficial not only in the exam but also in real life. While each address follows the same format, just like IPv4, there are various address types. We will discuss them in the next section.

Understanding address types

For the exam, it is important to be able to identify each of the various address types and understand their purpose. In this section, we will discuss each of them in turn.

Global unicast address

An IPv6 global unicast address is similar to a public IP address in IPv4, in that it is routable across the internet. Because there are so many available IPv6 addresses, it is feasible that every device can have its own global unicast address. This means that **Network Address Translation** (**NAT**) is not required, although this has not been well-received by all network administrators who appreciate the added security NAT provides.

Global unicast addresses all begin with `2000::/3`, and the address is broken down into parts:

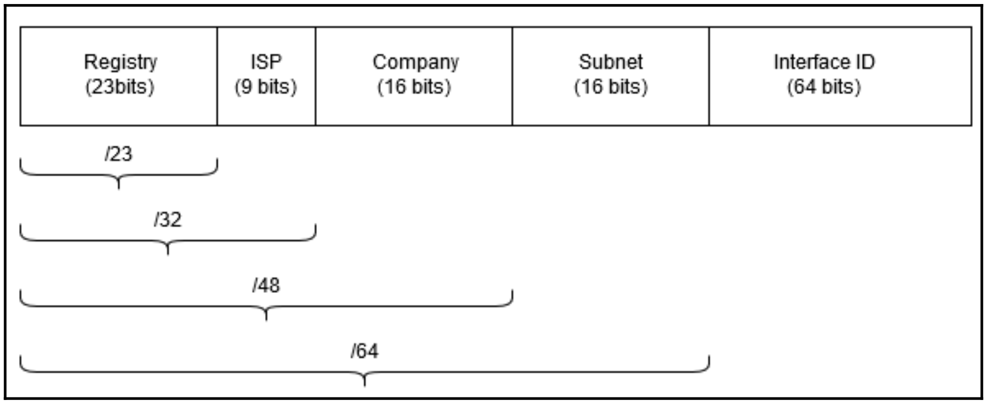
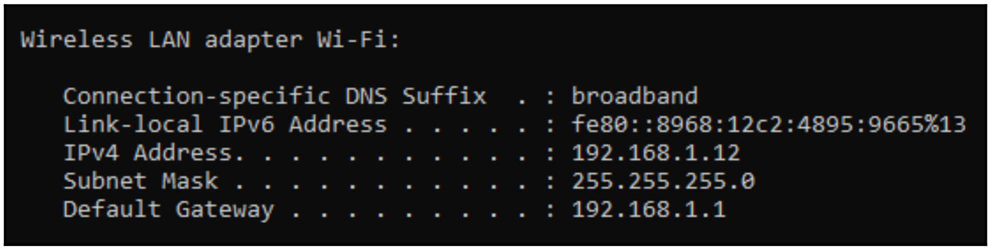


Figure 12.1: Breakdown of a global unicast address

The first 23 bits identify the registry that controls the IPv6 address; the next 9 bits identify the ISP that has been granted the IP address. The ISPs issue an IPv6 address to an organization, and this organization is identified in the next 16 bits. Finally, the organization subnet this with the next 16 bits. This leaves the final 64 bits for the interface ID. An interface ID is the same as the host element we have seen in IPv4.

Link-local addresses

A link-local address is similar to an IPv4 APIPA address. They are not designed to be routable across the internet. A Windows device is automatically assigned one of these addresses, but unlike IPv4, a Windows device can have a link-local address, as well as a unicast address. In *Figure 12.2*, we can see a link-local address in use:



```
Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . : broadband
   Link-local IPv6 Address . . . . . : fe80::8968:12c2:4895:9665%13
   IPv4 Address. . . . . . . . . . . : 192.168.1.12
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.1
```

Figure 12.2: Link-local address

Link-local addresses all begin with `FE80::/10`. While it is not Windows-specific, a lot of network administrators will allocate the same link-local address to each interface on a router. As you may recall, each interface on a router connects to a different network. Therefore, allocating the same link-local address to each interface is permitted because they are all on different interfaces.

The presence of a `%` number at the end of the line, for example, `%13`, as seen in the preceding screenshot, is another indication that this is a link-local address. This number is an identifier of the reachability scope of the address. Reachability scopes are beyond the exam objectives, but as the number is showing in the screenshot, I felt it important to just mention it here. There is a link to a Microsoft article in the *Further reading* section that goes into a little more detail about these if you wish to read it.

One last point I would like to make about link-local addresses is that a network card can have a link-local address AND one of the other address types simultaneously.

Unique local addresses

Unique local addresses are the IPv6 equivalent of private IPv4 addresses and begin with `FC00::/7`. Like their IPv4 counterparts, these are not routable across the internet.

Multicast addresses

Like their IPv4 counterpart, IPv6 multicast addresses are used to transmit data to devices within a specified group. Multicast IPv6 addresses begin with `FF00::/8`.

As a sort of replacement for broadcast, there is a multicast group called **all nodes**, which sends data to all the devices on the network. The all nodes multicast address is `FF02::1`.

Loopback address

Despite having so many IP addresses available, the IPv6 developers realized that the loopback address range used in IPv4 wasted so many IP addresses. Therefore, in IPv6, there is only one loopback IP address and that is `::1`. This is obviously the shortened version of the address, and all the preceding hexadecimal characters are zeroes.

By understanding the various address types, you will be able to not only identify what an address is used for, but you will also be able to identify whether it has been applied correctly. For example, you may see a link-local address in use when you were anticipating a global unicast address.

Now, we have discussed the address formats and types, let's move on to actually assigning an IPv6 address to a host.

Assigning IPv6 addresses

Like IPv4, IPv6 addresses can be assigned manually or dynamically. However, to obtain an IP address automatically, we can use either stateless auto-configuration or DHCPv6.

Manual configuration

Manually assigning an IP address to a Windows computer involves adjusting the IPv6 properties of the NIC itself. As a device can have more than one NIC, ensure you are configuring the right one. Let's quickly walk through this process:

1. From the Control Panel, select **Network and Internet**:

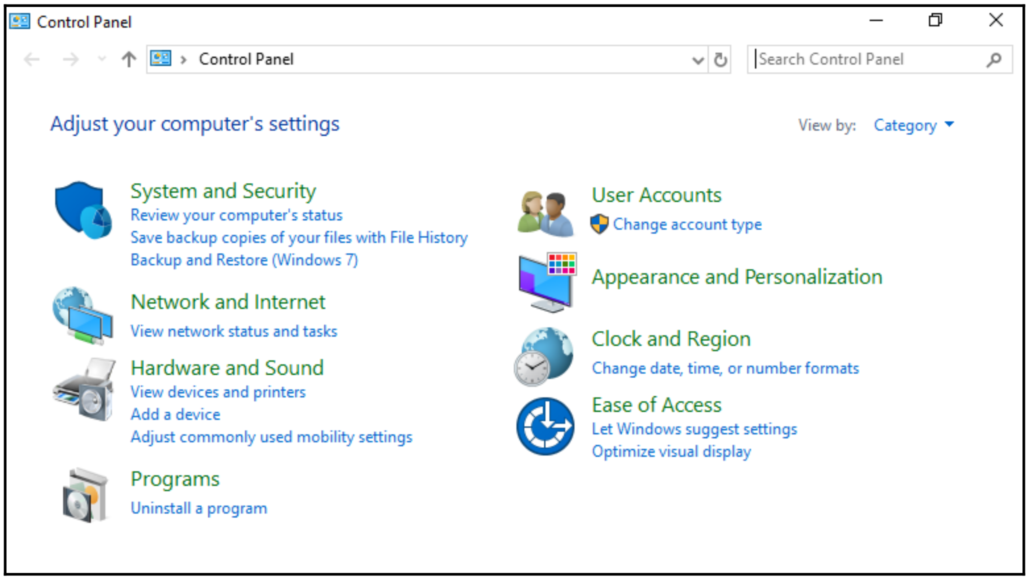

Control Panel
Control Panel
Search Control Panel
Adjust your computer's settings
View by: Category
System and Security
Review your computer's status
Save backup copies of your files with File History
Backup and Restore (Windows 7)
Network and Internet
View network status and tasks
Hardware and Sound
View devices and printers
Add a device
Adjust commonly used mobility settings
Programs
Uninstall a program
User Accounts
Change account type
Appearance and Personalization
Clock and Region
Change date, time, or number formats
Ease of Access
Let Windows suggest settings
Optimize visual display

Figure 12.3: Control Panel