IPv6 for future CCNAs – Part II

written by **Gerald C. Paciello**

February 18, 2016

In our IPv6 for future CCNAs – Part I, we ended the article talking, very lightly, about **IPv6 address types**. Let´s go ahead and dig deeper into the concepts.

To recap, the IPv6 address types are:

1. **Unicast.**
    - **One-to-one** communication. **Unique** address assigned to an interface, a packet sent to a Unicast address will be received by **one single interface**.
      There are several types of Unicast addresses:
        - **Global Unicast.**
        - **Link Local.**
        - **Unique Local.** (in place of Site-Local which was deprecated in 2004).
        - **Special Addresses**
            - **Unspecified.**
            - **Loopback.**

2. **Multicast.**
    - **One-to-many** communication. A **Multicast** address identifies a **group of interfaces**. A packets sent to a **Multicast** address are received by a group of interfaces that may be in different hosts.

3. **Anycast.**
    - **One-to-one of many** communication. An **Anycast** address represents a **group of interfaces**, but the packet sent to this address will be delivered **only** to the interface which is **closest**, in terms of the **routing protocol cost value**.
      Also, since **Anycast** addresses are allocated from the **Unicast address space**, they are syntactically indistinguishable from each other. So, an Anycast address is a Unicast address that was assigned to more than one interface.

Unicast Addresses: Global Unicast.

This address uniquely identifies an interface on a network, it is **globally routable** and it has a hierarchical structure. You can think of a ***Global Unicast*** address as the *Public* address on IPv4.

Configuration.

A Global Unicast addresses can be configured either automatically (**stateless**) using **Stateless Address Auto-Configuration - SLACC** or manually (**state-full**) through the **CLI.** Please note that when I say that a Global Unicast address can be "*automatically*" configured, it does not mean that its configuration will be 100% automatic, it means that you can either configure a specific value for the **Interface ID** manually using the command **"*ipv6 address xxxx::x/64*"** on the **CLI** or, you might prefer letting the **SLAAC** come up with the **Interface ID,** by using the command **"*ipv6 address 2001::/64 eui-64*"**. But, as you can see, in both cases you will need to go through the **CLI** and enter commands to configure either option. (More on SLAAC and EUI-64 later)

Here is a short video on Global Unicast address configuration:

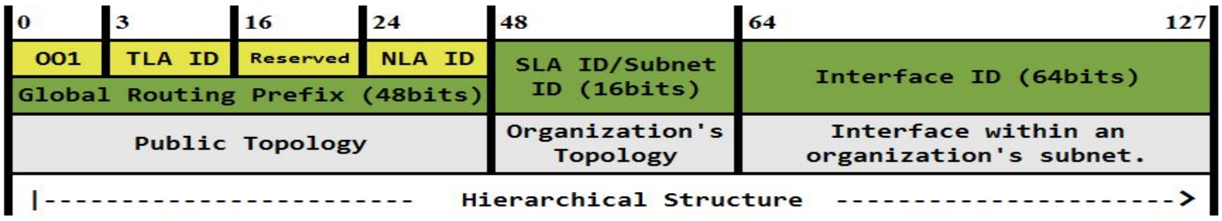
https://youtu.be/NTlWaNBnk4U

Global Unicast address structure.

When the IANA allocates an IPv6 address to you, you get an address that looks like this:

Global Routing Prefix	Subnet ID	Interdface ID	
2001:0AC8:1234:	0000:	0000:0000:0000:0000	/64

Let´s examine that structure in details:



The first **48** bits are referred to as the **Global Routing Prefix** and it is further divided into 4 fields:

***001*:** First 3 bits are fixed and they identify a **Global Unicast** address, so the available range goes from **2000::/3** to **3FFF::/3.** (The **IANA** is still assigning **Global Unicast** addresses in the 2001::/3 range so you might not see anything other than **2001::/3** for a while).

***TLA ID*:** As you can see on the picture above, a **Global Unicast** address has a hierarchical structure and this **Top Level Aggregation ID** (13bits) represents the **highest level** of the routing hierchy. It is allocated by the **IANA** to **RIR**s and in turn to the local **ISP**s and finally to the organizations.

***NLA ID*:** The **Next Level Aggregation ID** (24bits)**,** identifies the **second highest** level of routing hierarchy which is the organization that has been allocated this block of address.

***SLA ID*:** Also referred to as **Subnet ID,** the **Site Level Aggregation ID, i**dentifies a subnet within the **Organization's Topology**. It is used for **subnetting**, and since it is **16 bits** long, there are a total of **65,536** possible subnets (**2^{16}**).

***Interface ID*:** Identifies a single interface in a subnet within the organization`s topology.

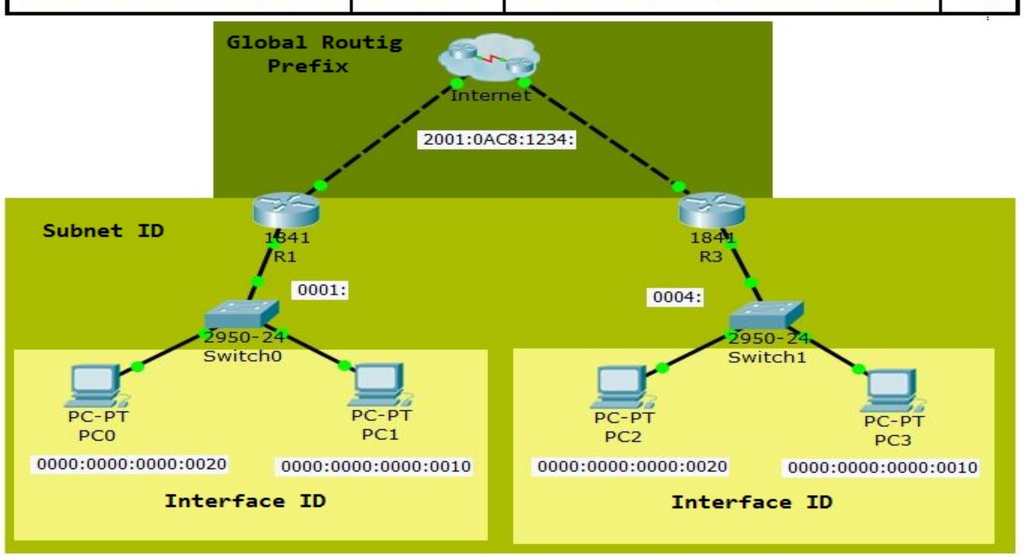
Hierarchical Structure.

Let´s take a closer look at the hierarchical Structure:

2001:0AC8:1234:0000:0000:0000:0000:0678	
2001:0AC8:1234:	**Global Routing Prefix** (001/TLA ID/RES./NLA ID)
0000:	**SLA ID** (Subnet ID)
0000:0000:0000:0678	**Interface ID**

Let´s see it in a topology:

Global Routing Prefix	Subnet ID	Interdface ID	
2001:0AC8:1234:	0000:	0000:0000:0000:0000	/64



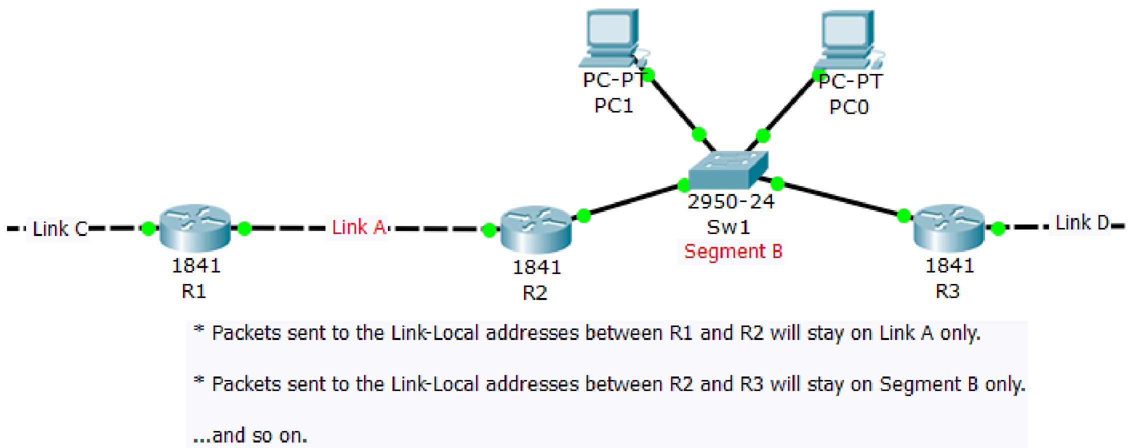
Unicast Addresses: Link-Local.

The most important concept that you need to understand about a **Link-Local** address is the following;

> **"It is only locally significant and only used by local protocol and services` overhead".**

Now, let´s see the details; an **IPv6** interface is required to have a **Link-Local** address, it may not have any other address type assigned, but it must have a **Link-Local** address. Hence, the **Link-Local** address is assigned automatically just by enabling the interface for **IPv6** (This behavior is a lot like the **APIPA** address in **IPv4)**.

The **Link-Local** address is, as its name implies, a local address used **only** inside a network **link/segment**, it will **not be routed** outside the broadcast domain in which it was originated.
The following picture will help you visualize where a **Link-Local** address is used:



Personal thought: In many places you will find the term "**segment**" referring to a "**link**" (a cable from device A to device B), **and that is correct**, technically it is a segment of the network. However, that really confused me a lot, so I (just me, I think...) started differentiating a link from a segment by calling the cable from **R1** to **R2** (in the picture above for example), a "**link**" and calling it a "**segment**" when ever there's something else in between devices, such as SW1 between **R2** and **R3** in our picture.

So, learning where a **Link-Local** address is used was fairly easy, but what about its purpose?... One of the best ways, I think, to understand what something is or what it does, is to know what it is used for, so here is a list of **Link-Local** address uses:

- *Router's communication on the same link/segment.*
- *Address Auto-Configuration (**SLAAC**).*
- *Neighbor Discovery Protocol (**NDP**).*
- *Routing protocol advertisements and next-hop address.*
- *Host-to-Host connection (crossover cable).*
- *"Host-Switch-Host" connection (no router)*.

Notice that a even though a Link-Local address is not routed, it can still be used to send user data (the last two uses listed above), in other words; if you have a Host-to-Host connection (using a crossover cable) or you have a bunch of hosts connecting through a switch, it is all just one segment, hence Link-Local addresses will be used to send user data because packets are NOT being routed outside the local segment.

Configuration.

Just like with a **Global Unicast** address, a **Link-Local** address can be configured either automatically (stateless) through **SLACC** and **EUI-64**, or manually (stateful) using the "*`ipv6 address FE80::x link-local`*" command, where "*x*" is the unique value (including 0) you want to use for the **Interface ID**.
However, the **Link-Local** address auto-configuration, as opposed to the **Global Unicast** auto-configuration, is 100% automatic, you do not need to enter any commands to configure it, it is auto-configured either when you use the command *"`ipv6 enable`"* on an interface or, when you assign a **Global Unicast** address to an interface.

Address Structure.

The address block **FE80::/10** has been reserved for **Link Local** addresses. This means that a **Link Local** address has the **10** (/10) most significant bits **set** to **1111 1110 10** so, according to this, a **Link Local** address will start with either **FE8, FE9, FEA or FEB** because of bits 11th and 12th can still be either a **1** or a **0**, providing for other possible values for hex **8**.

Address	F				E				8				::/64
1st 10 Bits	1	1	1	1	1	1	1	0	1	0	0	0	
|--------------------- 128 Bits -------------------|													

However, if you take a look at **RFC 4291 – IPv6 Addressing Architecture Sec. 2.5.6**, you will see this:

```
2.5.6.  Link-Local IPv6 Unicast Addresses

   Link-Local addresses are for use on a single link.  Link-Local
   addresses have the following format:

   |   10     |
   |  bits    |         54 bits         |          64 bits           |
   +----------+-------------------------+----------------------------+
   |1111111010|           0             |       interface ID         |
   +----------+-------------------------+----------------------------+
```

The picture above shows the first 10 bits set to **1111 1110 10** as it should, that is the rule, but it also says that the following **54bits** should also be **set to 0**s, it does not leave any other possibility but zeroes, thus, eliminating the **FE9**, **FEA** and **FEB** possibilities. (This is because bits **11th** and **12th** will always be set **0,** according to the graph above).
Yet, when you manually configure a **Link-Local** address on an interface, either **FE9**, **FEA** or **FEB** will be accepted as a valid **Link-Local** address!

OK, I want to give you a fair warning about the statement above; it does not work in Packet Tracer!
If you want to try assigning a **Link-Local** address to an interface, other than **FE80::**, like **FEA0::** for example, **Packet Tracer** does not accept it:

```
Router(config-if)#do sh ipv int br
FastEthernet0/0            [up/up]
FastEthernet0/1            [up/up]
Vlan1                      [administratively down/down]
Router(config-if)#ipv6 addre
Router(config-if)#ipv6 address fea0:: li
Router(config-if)#ipv6 address fea0:: link-local
% Invalid link-local address
Router(config-if)#
```

However, if you do the same on a real router:

```
GCP(config-if)#ipv add fea0:: link-local
GCP(config-if)#do sh ipv int br
FastEthernet0                 [up/up]
FastEthernet1                 [up/down]
FastEthernet2                 [up/down]
FastEthernet3                 [up/down]
FastEthernet4                 [up/up]
Vlan1                         [up/down]
NVI0                          [up/up]
Vlan10                        [up/up]
Loopback129                   [up/up]
    FEA0::
GCP(config-if)#
```

Please note that when **Link-Local** addresses are configured with **SLAAC**, the **RFC** statement coincides perfectly. In other words; when assigning a **Link-Local** address automatically, the address will always start with **FE8** followed by **54 zeroes** (**FE80::**)... hmm, maybe the statement is referring to this scenario only?...

Well, I guess we can summarize this by saying that if a **Link-Local** address was configured automatically (which is **99%** of the time), it will always be **FE80::**, but if you come across an **FE9, FEA** or **FEB**, you know that it was configured manually.

Quick video to check your understanding:

https://youtu.be/oAvcUn976As

Subnetting.

Subnetting... SUBNETTING... it was a big word on IPv4, lots of math involved... not so much on IPv6 though.
Even though at a CCNA level there is not much math in IPv6 subnetting, it can get pretty complicated, especially from an structural point a view (search for multi-level IPv6 subnetting and you'll see what I mean), but thankfully, we do not need to go that deep.

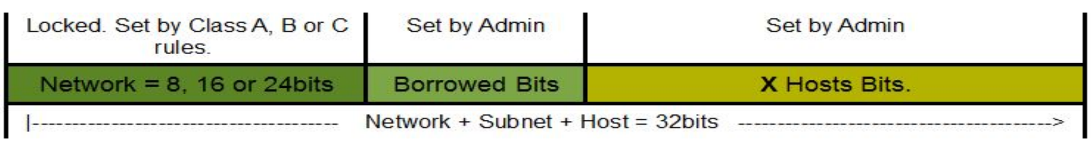
Here is what you need to know about IPv6 Subnetting for CCNAs; The scope for subnetting hasn't changed, it's 1 subnet per All subnetting occurs in the **Subnet ID** field and it's as easy as assigning each subnet needed, a unique hex value between **0000** and **FFFF.** That's a total of **65,536** possibilities!
Yes, that's right, with a **/64 IPv6** address, we can have **65,536** subnets... if you think that is wasteful for most, wait until we talk about the **Interface ID** next!

Now, there is no need at all for subnetting the interface bits, let´s see why that is; The **Interface ID** it´s **64bits**, this means that **each subnet** can have **2^{64} unique** addresses...and that is... wait for it... **18,446,774,073,709,551,616** unique host... sorry again... I mean interface addresses**!**
Now you're thinking; Wait a minute, there are **4,294,967,296 TOTAL IPv4** addresses, and with IPv6 we can have **18,446,... -**the big number we just mentioned- for **each** subnet!... man... **IPv6** is big!... Yeah... it is!
That's it; change the **Subnet ID** to uniquely represent each of your subnets and go home... or Moe's Tavern if it's Friday! :O)

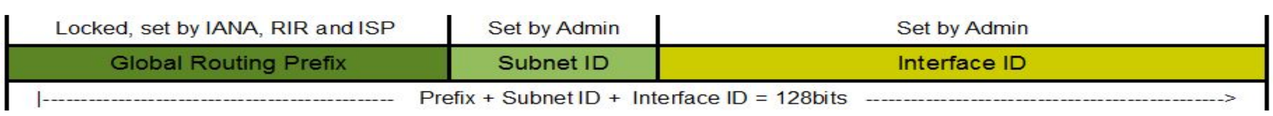
And before you ask, yes, it is possible to "borrow" interface bits to create more subnets, but there are a couple of very good reasons why you shouldn't; first and foremost, because you will never need more than **65,536** subnets, and second; because for **IPv6 auto-configuration** to work, it needs a **64bit Interface ID**. For example, a very common **IPv6** configuration method is **Stateless Address Auto-Configuration** - **SLAAC,** and this method uses the **Extended Unique Identifier** - **EUI-64** address, which takes the interface's 48bit MAC, sticks **FFFE** right in the middle and then flips the 7th bit, so it'll end up with a unique **64bit Interface ID** -we will talk about IPv6 auto-configuration and the EUI-64 address in more detail, later on in our discussion.

Here is a couple of pictures, to help you visualize where subnetting takes place for IPv4 and IPv6:

IPv4 Address:



IPv6 Address:



END OF PART II

References:

- Members of the Cisco Learning Network at https://learningnetwork.cisco.com/welcome -Thank you guys!
- https://tools.ietf.org/html/rfc4291
- https://docs.oracle.com/cd/E19683-01/817-0573/chapter1-26/index.html
- https://technet.microsoft.com/en-us/library/bb726995.aspx#EDAA
- http://www.iana.org/assignments/ipv6-unicast-address-assignments/ipv6-unicast-address-assignments.xhtml
- http://ipv6.com
- https://technet.microsoft.com/en-us/library/cc757359(v=ws.10).aspx
- http://www.firewall.cx/networking-topics/protocols/877-ipv6-subnetting-how-to-subnet-ipv6.html