>
Published on *NetworkWorld.com Community* (http://www.networkworld.com/community)

IPv6 Address Design

By *jdoyle*
Created *Jan 28 2011 - 2:20am*

There are a few culprits that regularly contribute to delayed or failed IPv6 deployment projects, such as poor DNS planning, insufficient testing, unanticipated application behavior, and poor IPv6 support in peripheral support, management, or security systems. Many deployment projects suffer temporary halts when the original IPv6 address design is found to be inadequate – in a few cases, the address design has had to be reworked more than once.

Even worse that an IPv6 address design that halts a project is a design that is just good enough to allow the project to proceed, but bad enough that it will saddle the network with expensive operational difficulties for years to come.

The address design should be considered from the very beginning of the planning phase for IPv6 deployment. This article discusses some of the factors you should consider when creating an IPv6 address design.

This is Not IPv4

Most network architects are too young to remember when IPv4 address space was not considered a limited resource. We are all sensitive to any sign of waste in an IPv4 address design, and approach new designs expecting to carefully analyze each network segment and provide just enough addresses to meet requirements. Variable Length Subnet Masking (VLSM) is essential, carefully carving out enough subnet space while trying to insure enough host addresses in each subnet.

Abandoning many of those conservative design ethics is one of the hardest parts of performing an IPv6 address design. Take, for example, the commonplace issue of making every subnet in an IPv6 network a /64, including point-to-point links. The advantage of doing this is the one-size-fits-all consistency in your subnetting, doing away with the headaches of VLSM. But it invariably makes people itch: The idea of having a subnet with $1.8446744074 \times 10^{19}$ available addresses and only using two of them – and knowing that you will never, ever need more than two of them – seems mind-bogglingly wasteful.

And it *is* wasteful. But the IPv6 address space assigned to most networks is so vast, you can afford a shocking level of waste in exchange for consistency, simplicity, scale, and efficiency. If you have a /32 prefix, you can have 4.3 *billion* /64 subnets – as many subnets as there are addresses in the entirety of IPv4. Applying traditional IPv4 address conservation practices to an IPv6 design can be a bit like spending $200 to find a way to save $2.

What to Look For

Although you can discard much of your concern for address waste, existing IPv4 address management should be assessed. There are two sides to this consideration. On the one hand, the more you can integrate IPv6 addressing into familiar practices, the easier you make life for your operations personnel. On the other hand, if your existing address management practices have been inadequate, this is the perfect time to make changes.

My own experience has been that many if not most of my clients are happy to abandon what they see as an inadequate, inefficient, or barely existent IPv4 address management workflow.

In addition to address management practices, any polices that filter on an IP address must also be analyzed. These include routing, summarization, and security policies. Again, your analysis should balance what is familiar and useful with what has long needed improvement.

And if you are one of the many network operators who still manages your IP addresses with spreadsheets, this is a good time to consider purchasing IP Address Management software. Beyond a simple formula that converts decimal to hexadecimal within a single cell, Excel has no hex support. Hex numbers must be entered as text, and adding long sequences of hex numbers is tedious and mistake-prone. I have spent many a long night mindlessly typing out thousands of hex numbers into cells in developing IPv6 address designs for my customers.

A good IPAM package not only makes IPv4 and IPv6 address management easier and safer, it integrates DNS and DHCP management too.

Designing for Simplicity

Even with a nice IPAM application, engineering and operations personnel are still going to be looking at and trying to make sense of those ugly IPv6 addresses every day. The easier your addresses are to interpret, the more you can reduce routine mistakes.

Start by mapping out the bits of the address space that are available to you to work with. You will have some assigned prefix – usually 29, 32, 48, or 56 bits, depending on the size of your network – that are always the same. The last 64 bits, the Interface-ID, should usually be left alone except for certain classes of address such as loopbacks. The space between the assigned prefix and the Interface-ID are the bits you have to work with.

Obviously the 8 bits between a 56-bit prefix and the Interface-ID give you far less flexibility than the 32 bits between a 32-bit prefix and the Interface-ID. But if you have a 56-bit prefix you have a small network anyway, and there is little need for much meaning other than simple subnet numbers.

As you map out the bits of your workable space, group them by hex digit (four bits per digit). Then decide what “meanings” you need to have in your addresses for easy interpretation. Meanings might include geographic locations (such as region, city, POP, office), a logical topology (such as OSPF area or simple subnet number), a type designation, or a customer / user ID.

Then try to match your defined meanings to hex digits rather than to individual bits. This is the key to keeping your design simple: If your personnel can interpret the meaning of one or more hex digits without having to decipher the address to the bit level, time is saved and risk is reduced.

For example, suppose your network is constructed into nine regions, the largest region has 100 offices, and the largest office has 75 subnets. You might designate one hex digit as a Region ID (one hex digit is 4 bits, giving you 16 Region IDs), two subsequent digits as an Office ID (two hex digits is 8 bits, giving you 256 Office IDs per region), and two digits as a Subnet ID (256 Subnet IDs per office). Such a design will require a /44 prefix at minimum; a /40 will give you room to spare.

Your engineers will quickly learn to focus on just the hex digits that are meaningful to the task at hand – the digits between the assigned prefix and the Interface-ID. So instead of trying to take in 32 hex digits (128 bits), they are only looking at, for instance, the 8 digits following a /32 prefix, 6 digits after a /40, 4 digits after a /48, or 2 digits after a /56.

In larger networks requiring more design complexity, the format of lower-order bits might vary according to the "meaning" defined in higher-order bits. In the example above suppose you have, in addition to the nine regions, two data centers each of which have 2000 subnets. Region IDs E and F could identify the data centers, and behind those IDs instead of a two-digit Office ID and two-digit Subnet ID you could have a four-digit Subnet-ID (for 65,536 subnets per data center).

But don't get carried away with adding too many "meanings" in your design. There is no need to add ten layers of hierarchy to your address design if four layers are enough to tell your engineers everything they need to know about an address. This leads to another way to keep things simple: Use strings of zeroes in your addresses as much as possible. 2001:DB8:2405:C::27 is a much easier address to read and record than 2001:DB8:2405:83FC:72A6:3452:19ED:4727. If the first address gives you enough information to manage your network, why use the second?

Designing for Scale

"Scale" and "scalability" might be two of the most overused words in the network design vocabulary; I've certainly contributed to turning those perfectly good words into creaky buzzwords. Nevertheless, the concept of scaling is crucial to good design, whether we're talking about protocols, devices, logical topologies, or addresses. You don't want something that cannot accommodate growth.

The amazing thing about IPv6 is that you can often afford to be wasteful in some respects, creating subnets with a vastly larger number of addresses than will ever be used, and at the same time allowing room for growth at the subnet level and above.

I mentioned that a /40 prefix will give our simple (and admittedly simplistic) example design room to spare. I also said that you should use zero space as much as you can to shorten the overall address. The zero space should be designated as "Reserved," and distributed in your design in such a way that multiple fields can expand into it if needed.

The example design requires 5 hex digits (20 bits) to provide the Region, Office, and Subnet IDs. If you have a /40 prefix, you have 6 hex digits (24 bits) to work with. Rather than making the 11th hex digit the Region ID, the 12th and 13th digit the Office ID, the 14th and 15th digit the Subnet ID, and the 16th digit Reserved, place the Reserved digit between the Region ID and the Office ID. In addition to making the Office and Subnet IDs fall on more tidy boundaries, the placement of the 4-bit Reserved digit between the Region and Office IDs gives you flexibility should you ever need it: Either the Region or Office IDs can grow into that space if you underestimated future growth, or you could use the space to add another layer of hierarchy should future requirements dictate it.

Designing for the Future

A challenge for any network design is attempting to anticipate future needs, when you don't clearly know what the future holds. "Anticipating the unanticipated" seems to be an exercise in futility, and sometimes it is. But if you make good use of reserved space, well distributed, you stand a better chance of being able to accommodate future requirements with a simple expansion of definitions within your existing design rather than having to do a complete redesign.

The large working space of most IPv6 allocations also help you to “future-proof” your address design. Frequently you can reserve an entire space behind your assigned prefix, giving you the room to add a different format without abandoning your existing format.

Another factor to consider is that at some point in the future – no one can say for sure when, but I think it will be sooner than most people expect – IPv4 will become obsolete. The catalyst for accelerated IPv4 obsolescence will be the expense, risk, and difficulty of running two versions of IP in a network. The only direction forward is to IPv6, so at some point network operators will make a deliberate effort to push IPv4 out of their network.

Given that assumption, do not make IPv4 addresses an element in your IPv6 design unless there is a strong reason for doing so. For example, some engineers will designate the last 32 bits of a device’s IPv6 loopback address as the same as the device’s IPv4 loopback address. But does this really have any benefit? After all, the IPv4 bits will be encoded in hex, and not readily recognizable by operations staff: Given the IPv6 address 2001:DB8:1305:7C::C0A8:53E5, is it readily apparent that the last 32 bits are the IPv4 address 192.168.83.229?

An effort to “integrate” IPv4 addresses into your IPv6 addresses looks backward to IPv4 rather than looking to the IPv4-free future. If IPv4 is eventually removed from your network, a bad design could lock you into accommodating an obsolete addressing system.

Designing for Efficiency

Your preliminary analyses of routing, summarization, and security policies pays off in helping you to create an address design that maximizes the efficiency of any device that must filter on an IPv6 packet’s source or destination address. If a filter must scan well into the address to find the bits that trigger a “hit,” your list of filter rules can grow long. Take into account the number of potential addresses within your IPv6 prefix, and that list could become unmanageably huge.

So as much as possible, try to design your addresses so that any “bits of interest” to a filter appear early in the address (either the prefix part or the Interface-ID part), so that they can represent as many individual addresses as possible.

What About the Interface-ID?

Most Interface-IDs in your network will be 64 bits, with the possible (and arguable) exception of point-to-point interface addresses. If stateless address autoconfiguration (SLAAC) is to be used anywhere in your network, it is particularly important for the Interface-IDs in those segments to remain 64 bits in order for SLAAC to work. But if you are assigning addresses either via DHCPv6 or statically, this is another opportunity to use plenty of reserved zero bits and simplify your addresses. You might, for example, make only the last 12 bits assignable and set all the preceding 52 bits to zero. That gives you 4,096 addresses per subnet – and isn’t that enough?

12 or 16 bits also gives you the room to include some randomization within your Interface-ID assignments, reducing your exposure to port scans that try to find working devices on a subnet by beginning at the lowest Interface-ID and working sequentially up.

At the same time, you might have need of some identifier digits within the Interface-ID, either for filtering at the subnet level or just for easy identification of device types within a subnet. In this case, use some of the leading bits in the Interface-ID, at the other end from the interface-specific bits, and again leave reserved zero space between. If you do use identifiers within the Interface-ID, they should be type identifiers. All location meanings should reside in the first 64 bits of the address preceding the Interface-ID.

The enormous 64-bit capacity of the Interface-ID leaves you plenty of room for identifiers, more than enough addresses per subnet, and the ability to keep your overall address manageably small.

Fun and Games with Hex

When I taught networking basics classes many years ago, I emphasized to my students that they should be careful about trying to interpret IPv4 addresses at the dotted decimal level; they should practice converting each of the four decimal numbers into their binary equivalent, until they became proficient at just looking at a number, say 240, and automatically seeing 11110000. The patterns of the bits tell you how a router interprets an IP address, not the decimal value representing the bits.

With a good design and no VLSM to obscure things, IPv6 is a bit easier to interpret at the hex level. But there are still plenty of good reasons to sometimes delve down to the individual bit values to look for a pattern. So the ability to quickly convert between hex and binary is just as important for working with IPv6 as converting between decimal and binary is for working with IPv4.

There are a few simple tricks that make these conversions fast and easy. (And no, you cannot count on always having a scientific calculator handy to do the conversions for you.) In the next post I’ll show you how.

IPv6 LANs / WANs address design IPv6

Source URL: http://www.networkworld.com/community/blog/ipv6-address-design