Inessential IPv6: A primer for those with a technical background

Author: Erik Nygren <ipv6-doc@erik.nygren.org> (and likely other collaborators)
Date: Aug 18, 2022 ; last updated: Jul 25, 2023
Status: Work-in-Progress Draft

Copyright 2023 Akamai Technologies under a **CC BY** 4.0 license.

Note: this will be published someplace else and converted to a better format at some point.

Foreword

(Feel free to skip this and jump straight to the Introduction…)

Many of us who have been using the Internet for decades grew up in a world that was Legacy IPv4-centric. In the intervening decades, IPv6 has gone from a concept (originally "IPng") to getting implemented in operating systems and network devices and applications around the world, to now being used on a daily basis by over half the end-users in many parts of the world. This introductory guide is for those of you who have a deep knowledge of IPv4, TCP, DNS, and the like, but only a peripheral knowledge of IPv6. This intro assumes a fairly deep level of technical background so is not the best general starting point. However, when I started working on IPv6 back in 2009 I first started with a few-pager that my colleague James "Kretch" Kretchmar wrote in LaTeX and it helped me bootstrap myself.

This introduction is not meant as a replacement for more formal documentation such as RFCs or books on the topic. Many things are hand-waved over here or not covered in full detail. For comprehensive answers, see RFC 8200 and other RFCs on IPv6.

For more context on how far the world has come in the past decade (from 2012 to 2022), see my Akamai blog post "10 Years Since World IPv6 Launch". Akamai observes end-user adoption levels in the approximately 44% to 62% range for 6 of the top 10 global economies, as measured by looking at the percentage of requests to a subset of IPv4+IPv6 dual-stacked content on the Akamai CDN. Of the top 100 networks, approximately half have IPv6 adoption of over 40% (as measured by Akamai). What this means is that many people in the world are using IPv6 all the time without even knowing it, which is a success. Even some people who worked on IPv6 in the IETF back in the 1990's and early 2000's (and who would toast to "the universal adoption of IPv6") have lost track of how far IPv6 has come and are surprised by its success.

I still recall the IPv4 subnet for my college dorm at MIT (18.239.0.0/16) from before MIT sold that IP space off to Amazon for use in AWS. In today's dollars, those 65k IPv4 addresses is worth over $3M USD on the secondary market. Giving this much IPv4 space to a single college dorm is indicative of the problems with IPv4: the easy way to allocate address space is to over-allocate it. Trying to be

conservative about address space by packing it densely, and then reassigning it later rapidly becomes messy. As IPv4 has scaled and become inherently resource-limited, the need for careful address planning, resource allocation, and shuffling has resulted in fragmented routing tables, lots of NAT, and related messes. Even with RFC1918 Private Address space and NAT, many companies are struggling to manage their addresses. All it takes is a few mergers/acquisitions and some easy-but-poor-in-retrospect decisions long ago for large organizations to even have their RFC1918 space become an unmanageable mess.

Many of the design decisions in IPv6 are influenced by a strong desire to avoid some of these challenges and to create lots of flexibility for the future to come. Even if we botch IPv6 address space allocation for the next few decades, there is still vast swaths of space to allow us to change our approaches but without changing the underlying technology. But some of this flexibility in IPv6 also results in many of its challenges. IPv6 is far from perfect and with any complex system, many things can fail in surprising and frustrating ways. Many IPv6 implementations still lack some of the usability of their IPv4 counterparts. But at this point we are far enough along with our journey into IPv6 that we are all best served by improving implementations and deploying in an IPv6-centric approach. I'd much rather be spending my time architecting solutions that are IPv6-centric than negotiating how we shuffle things around to avoid conflicts in RFC1918 space.

If you are looking for a more comprehensive book on IPv6, please check out Brian Carpenter's open source IPv6 Book.

(Meta-comment: I'm starting this as in google doc as it has good collaborative editing functionality. I could have started in Markdown-in-git or LaTeX or DOCBOOK or an LMS or something else, but then I'd still be worrying about format rather than writing content. If this is successful I may convert it to something else.)

Table of Contents

Introduction

Today's Internet was initially built on IPv4 which is inherently limited to 32 bits (or four billion possible addresses), and even within this many blocks are reserved for purposes such as multicast. In a world with almost twice that many people, this just doesn't scale adequately. Even the use of large private blocks such as 10.0.0.0/8 only provide ~16.7 million addresses with perfectly dense allocation, and many large networks have many more subscribers than that. NAT44 can help, but quickly becomes hard to manage.

The primary purpose of IPv6 is scale. It also brings many more changes and different design decisions and complexity in some areas and potential for unlocking new use-cases. However, if it wasn't for the scaling limitations of IPv4 and the need to overcome them, IPv6 would likely never have been deployed.

IPv6 addresses are 128 bits so provide for 10^{38} addresses. This is enough to give 50 million addresses to every bacteria on Earth. Some current IPv6 address schemes may seem wasteful (eg, some ISPs giving a /56 per household) but even if we get this wrong, the current IPv6 allocation models mean we have many opportunities to retry before getting anywhere close to having issues.

Credit: xkcd (http://xkcd.com/865/), (License: CC BY-NC 2.5)

IPv6 started its development in the 1990's as part of an "IPng" effort in the IETF, with many of the core specifications such as RFC 1883 being published in 1995 and RFC 2460 updating it in 1998. The IPv6 page on Wikipedia has more summary details. Early deployments started in academia and industry, often using private networks, overlay networks, or tunnels. Implementations in most major operating systems and common client and server software happened over the 2000's. By 2011 these implementations had reached a point where some large ISPs and content providers were ready to turn IPv6 on in production at-scale for a "World IPv6 Day", followed by a "World IPv6 Launch" event in 2012 where more joined to turn native IPv6 on for real. This was just in time, as most regional address registries exhausted their pools of freely available IPv4 addresses and an IPv4 secondary market developed. In some cases, new deployments can still get small IPv4 allocations, but anyone needing large IPv4 blocks to build or grow a business have had to purchase them on a secondary market. For the past decade, more and more ISPs, content providers, cloud providers, and others have been native deploying IPv6 in production, often alongside IPv4.

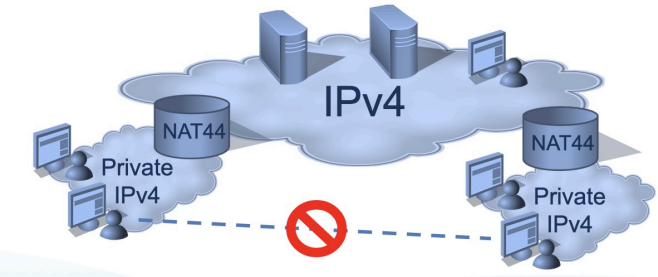
Deploying IPv6 alongside IPv4 is called “Dual Stack” and has been a highly successful model during this transitional stage. However, IPv6 adoption is now adequate in many areas to enable switching to various IPv6-centric model. In IPv6-centric models, networks and clients are provisioned as IPv6-only and access to legacy IPv4 resources are made available through a range of “IPv4-as-a-Service” (IPv4aaS) technologies. For example, many mobile ISPs are only provisioning IPv6 addresses to their handsets and then are using a mix of DNS64, NAT64 and 464XLAT technologies to provide access to legacy IPv4-only resources.

The typical mobile user in the US or India has IPv6 (and perhaps only native IPv6) on their phone and uses it much of the time without realizing it. Many fixed-line residential users in the US have Dual Stack IPv4+IPv6 and it just works. If you have a VM on a service such as Linode, there’s a good chance your VM has an IPv6 address in addition to its IPv4 address (or that you can enable this in its provisioning console).

If you’re a technical user, getting residential dual stack IPv4+IPv6 working in your house is often a good starting point (but may require switching ISPs if your ISP is one of the long tail of laggards). If you’re building out a new service (eg, in the cloud or a new data center or even if you are building our your own ISP), it is worth considering starting with an IPv6-centric model. We’re not yet at the point where IPv6-centric deployments “just work” as there are plenty of issues still being worked through, but I’m optimistic that this situation will be much better in a few years.

The ugly alternative to IPv6: more and more IPv4 NAT44/CGN

With constrained IPv4 address space, most deployments of IPv4 now heavily rely on NAT44. (NAT is Network Address Translation. NAT44 means translating from IPv4 to IPv4, typically in a way that also remaps TCP and UDP port numbers and requires state. CGN, also called CGNAT, is Carrier Grade NAT, which are large-scale NAT devices. NAT gateways may be performance bottlenecks and can run out of IPv4 addresses and/or TCP/UDP port space.
They are also not an option for server deployments. Pockets of machines can’t directly communicate without the help of services such as STUN (which also doesn’t work in many NAT environments). Client addresses are also “translated” so servers lose visibility which can be challenging for debugging.

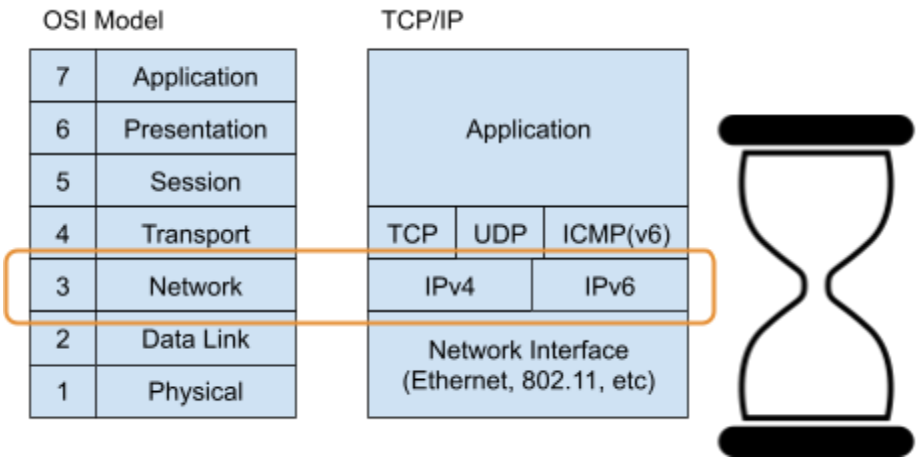


Large enterprises and large ISPs may also not have enough private IPv4 space so multi-layer NAT44 is sometimes needed, where different pockets of a corporate network have overlapping RFC1918 private address pools with NAT44 between them (yuck!), or ISP networks where packets get NAT’ed multiple times (eg, leaving the home and before reaching the Internet, and sometimes even in additional places). NAT has been successful in delaying the IPv6 transition, especially for smaller networks that aren’t as pressured by scale, but this also adds complexity.

Note that NAT is **not** a security technology by itself as there are attacks that allow external attackers to circumvent NATs. They can provide a false sense of security.

How does IPv6 relate to IPv4?

IPv6 and IPv4 both live at the Network Layer (Layer 3) in the OSI model. They are below the Transport Layer (which contains TCP and UDP) but above the Data Long layer (eg, Ethernet). This is at the neck of the "hourglass", as this Network Layer is the common denominator for the Internet which everything builds upon and which also provides an internetworking abstraction across all of the underlying layers. Changing this fundamental Network Layer of the Internet is *hard* and takes a long time no matter how we do it.



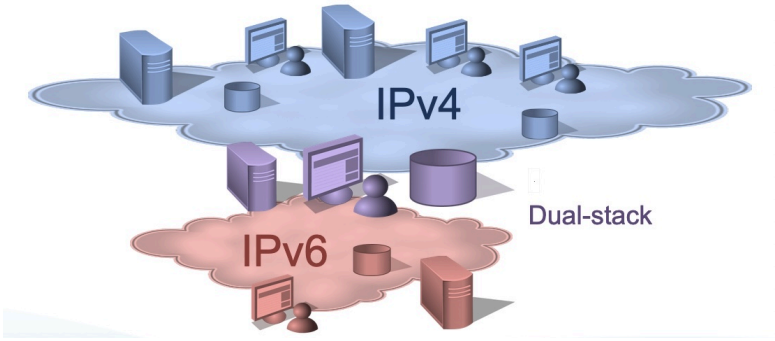
Changes resulting from the transition from IPv4 to IPv6 also have ramifications further up the stack. As an example, applications that interact with IP addresses have all needed changes to support both IPv4 and IPv6. Supporting both in-parallel is also often more complex than just supporting only one or the other.

There is no direct compatibility or connectivity between IPv4 and IPv6, so they are effectively two internets that live side-by-side. A given IP packet is either an "IPv4 packet" or an "IPv6 packet" on-the-wire. (As an example, somehow embedded an IPv4 address in an IPv6 address and putting that in an IPv6 packet is still IPv6 and not interoperable with IPv4. There are IPv4aaS transitions that do this, in-fact, but they rely on technologies such as "NAT64" to do this transformation between IPv4 and IPv6.

A fundamental difference between IPv4 and IPv6 is that while most hosts in IPv4 have just a single IPv4 address, the expectation in the IPv6 world is that most hosts will have multiple addresses simultaneously. This is discussed in more detail later, but it is important to make a shift from thinking about a host's "IPv4 address" to thinking about its "IPv6 addresses and IPv4 address".

The IPv4 to IPv6 transition

Many hosts and devices and servers will live on both IPv6 and IPv4, and this is called "IPv4+IPv6 dual stack" (or just "dual stack"). Dual-stack devices with have both IPv4 and IPv6 addresses.

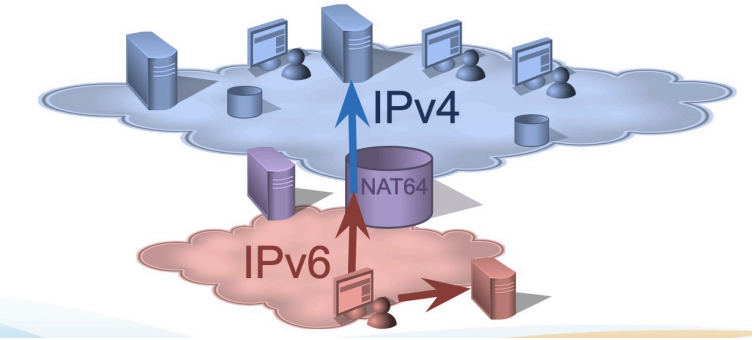


Other devices are IPv4-only or IPv6-only. While devices that are IPv4-only generally don't have access to IPv6-only resources, IPv4aaS transition technologies can provide access to IPv4-only resources from IPv6-only servers.

The general approach we are taking with the IPv6 transition has been following a path of:

1. Acknowledge we're starting from an IPv4-mostly world.
2. Start with having limited bubbles of IPv6 connectivity, but connected using tunnels over IPv4. Teredo, 6to4, and 6RD were examples of these. With limited exceptions (eg, Microsoft XBox using Teredo tunnels when it only has IPv4, or some early-mover ISPs still using 6RD) we are well past this phase and these approaches are being deprecated. Some advanced users may still use tunnels (such as via Hurricane Electric) to get IPv6 connectivity on IPv4-only networks. (I used a tunnel provider called SixXS for many years and am thankful for it, and they were able to sunset themselves in 2017.)
3. Broadly add dual stack support to software and network devices, preferably in a way that defaults to using IPv6 when it is available. Deploy dual stack connectivity in networks. Enable dual stack for services and content. Prefer using IPv6 where possible when both endpoints support it, and fall back to IPv4 as necessary. Over time, more and more will use IPv6.
4. Once IPv6 has enough momentum (and we are getting there), switch to IPv6-centric and IPv6-only-but-with-IPv4aaS models. Only give clients IPv6 connectivity, but provide ways for them to get at remaining IPv4-only resources. Only give servers IPv6 connectivity (and perhaps a limited way to get at IPv4-only resources) but enable access to them from legacy IPv4-only clients through dual-stacked load balancers, CDNs, or other ingress gateways.
5. Start phasing out the remaining ways to get at legacy IPv4-only resources. This will likely happen in limited domains first (eg, internal communications within a cloud or enterprise environment or IoT), but over the years may start applying elsewhere as well. As an example, the new Matter IoT standard is IPv6-only.

Note that through this entire transition, the average end-user should be blissfully unaware as to what mixture of IPv4 and IPv6 they are using, when everything is going well. Most end-users should ideally never have to interact directly with IP addresses, although they may see them in-passing in diagnostic interfaces.



Many of the IPv4aaS transition technologies are variations on NAT64, where IPv4 packets are embedded in IPv6 (either through encapsulation or with IPv4 addresses embedded in larger IPv6 addresses). Clients using this model have direct access to IPv6 resources (such as dual stack web sites) while IPv4 resources are accessed via the NAT64, still having many of the downsides of NAT44. In networks using these transition technologies (which include many 4G and 5G mobile networks), IPv6 will often have better performance.

Major differences between IPv4 and IPv6

It is worth giving a brief comparison of some major differences between IPv4 and IPv6:

- There is no broadcast in IPv6. Instead there is a heavier use of multicast.
- ARP is replaced by Neighbor Discovery (ND) which uses ICMPv6
- Address assignment often uses Stateless Address AutoConfig (SLAAC) which uses ICMPv6 for hosts to pick their own address within a /64. This replaces DHCP in many cases.
- There is no fragmentation at routers! If PMTU Discovery is broken, hosts can't communicate. It is important to make sure ICMPv6 PTB messages can make it through.
- There are a range of Header and Option changes. There is a somewhat longer packet header. While addresses are longer (128 vs 32 bits), there are fewer default fields (8 vs. 13).
    - There are no header checksums for IPv6
- IPv6 supports jumbograms for high-MTU links.

*(As a side-note, ipsec is **not** required in IPv6. This is a common misperception. Nothing on the ipsec front makes IPv6 magically more secure than IPv4.)*

	IPv4	IPv6
Example Address	198.51.100.74	2001:db8:23::23a:fd3:34cd
Localhost Address	127.0.0.1	::1
Address Length	32 bits (4 bytes)	128 bits (16 bytes)

Number of Addresses	4,294,967,296	340,282,366,920,938, 463,463,374,607,431, 768,211,456
DNS Records	A	AAAA
Finding Hosts on LAN	ARP	ICMPv6 ND
Minimum MTU	512 bytes	1280 bytes
IP Fragmentation in the network	Allowed	Not Supported
# Packet Header Fields	13 (+ options)	8 (+ options)
Header Checksum	Yes	No

IPv6 Addresses

Given that the primary motivator driving the adoption of IPv6 is to get more addresses, this seems like a good place to start. While IPv4 addresses are 32 bits, IPv6 addresses are 128 bits. In both of them, IP addresses act as identifiers (although not always unique), names, and locators.

In both of them, the higher-order bits identify the “network” and the lower-order bits identify the “host”. There is inherent hierarchy in the both types of addresses, allowing prefixes (sometimes called “CIDRs”) to be used to apply structure. In both IP versions, prefixes are used to route traffic, apply firewall rules, and more.

Representations of IPv6 addresses

As a reminder, IPv4 addresses are represented as a series of 4 decimal octets (8-bit values), such as 198.51.100.232, where each octet is a base-10 number between 0 and 255.

In IPv6, the 128-bit addresses is written as a colon-separated sequence of eight lowercase 16-bit hexidecimal values (ie, between 0x0000 and 0xffff). When represented this way, there are also a variety of notational short-cuts to help with readability and usage. A side-effect of this is that there are many ways to represent a given IPv6 address so care must be taken in performing string comparisons.

As an example, “2001:0db8:cccc:dddd:1111:2222:3333:4444” and “2001:0db8:000c:000d:0000:0000:0000:0004” are both fully qualified IPv6 addresses in expanded form. The latter is also the same as “2001:db8:c:d::4” in a canonical presentation form (which is much easier to write and takes up less space in logs!).

Some notational rules for the canonical presentation form:

- Written in lower-case

- Omit leading zeros, so:
  - “000a:000b:000c:000d:1000:0202:0001:0004” becomes “a:b:c:d:1000:202:1:4”
- Replace ***first*** longest series of “0:0:0:[…]:0” with “::”. For example:
  - “a:b:c:0:0:0:0:4” becomes “a:b:c::4”
  - “a:b:0:0:c:0:0:4” becomes “a:b::c:0:0:4”
  - “a:b:0:0:0:0:0:0” becomes “a:b::”
  - Note that you can never have more than one “::” in an address as this would be ambiguous.

For more details see RFC 5952: A Recommendation for IPv6 Address Text Representation.

In addition to the above, it is also possible to replace the bottom two 16-bit fields with an 32-bit IPv4 address in dotted-octet notation. For example:

- “2001:db8::a:b:198.51.100.7” is the same as “2001:db8::a:b:c633:6407”

Again, because of the various ways to represent IPv6 addresses, be very careful about performing string comparisons.

Converting IPv6 addresses in languages

In libc, the inet_ntop() function can be used to convert from a binary representation (the in6_addr structure in network byte order) into the presentation format. You can go the other way with inet_pton().

In python, the “ipaddress” class can be used to interoperate with IPv6 (and IPv4) addresses can be useful for conversions. For example, in Python 3:

```
>>> str(ipaddress.ip_address("000a:000b:000c:000d:1000:0202:0001:0004"))
'a:b:c:d:1000:202:1:4'
```

Representing port numbers along with IPv6 addresses

While the URI specification and many tools have long used a “:” to separate IP addresses from port numbers, this is ambiguous in IPv6. As such, when the two need to be represented together the IPv6 address is generally surrounded in square brackets. For example, “https://[2001:db8::abc]:8443/” is an unambiguous way to specify port 8443 on 2001:db8::abc.

IPv6 Network Addressing

In IPv6 you specify the networks in a ${NETWORK}/${PREFIX_LENGTH} format, similar to the “IPv4 CIDR” format. The same “replace the first longest string of zeros with ::” also applies. For example:

- 2001:db8::/16 = 2001:db8:0:0:0:0:0:0/16
- 2001:a:b:c::/64
- 2001:a:b:c::2:3/128

A /128 also specifies just a single IPv6 address.

In almost all cases involving hosts, the network prefix for a local subnet is 64 bits. This provides 64 bits for the network and 64 bits for the host identifier (the “IID”). While this may seem wasteful, much discussion has been had on this topic and many specifications and implementations require this model.

This also simplifies things. For example, in SLAAC network auto-configuration, hosts may select the bottom 64 bits themselves, as long as they do duplicate address detection (DAD).

Network Allocation and Usage

Because of the large amount of IPv6 address space, there is less of a need to be conservative about how much space is allocated (within reason). Example prefix lengths for a range of purposes:

Length	Typical Usage	# Addrs	# /64s
/128	Individual host (and hosts will often have multiple)	1	
/127	Inter-router point-to-point links (see RFC 6164)	2	
/64	Local network / LAN. Also the minimum prefix delegated to a residential network. In many cases (such as around trust and identity) it makes sense to treat an IPv6 /64 similar to an IPv4 /32. While a single /64 can have many clients (including ones that move around), this is similar to an IPv4 /32 which often has many clients behind it due to NAT.	1.8e19	1
/60 to /56	Typical assignment for residential networks via DHCPv6 PD (although this can vary widely). Using a /56 might also be common for a building on a campus.	2.9e20 to 4.7e21	16 to 256
/48	Longest prefix that is generally accepted in the global BGP routing table (similar to /24 in IPv4). Typical assignment for a site that may need independent routing (such as an enterprise office site, or an ISP POP).[1]	1.2e24	65,536
/40 to /22 (or more)	Common ranges given to entities (LIRs and ISPs) by RIRs such as ARIN.[2] The amount that can be requested depends on the expected utilization.	3e26 to 8e31	2^{24} (16.8M) to 2^{42} (4.4T)
/12	Unicast assignment to an RIR[3] (such as ARIN, LACNIC, RIPE, and APNIC).	8.8e34	2^{52} (4.5e15)
/7 to /10	Special purpose IANA assignments[4] (Link-scoped unicast, Multicast, ULA).		
/3	Global unicast space (only 2000::/3 is assigned so far, which includes 2000::/4 and 3000::/4).	4.2e37	2.3e18

[1] RFC 3177: IAB/IESG Recommendations on IPv6 Address Allocations to Sites
[2] https://www.arin.net/resources/guide/quickguide.pdf
[3] https://www.iana.org/assignments/ipv6-unicast-address-assignments/ipv6-unicast-address-assignments.xhtml
[4] https://www.iana.org/assignments/ipv6-address-space/ipv6-address-space.xhtml

As seen in the above, while there are ways to burn through addresses quickly there are also lots of opportunities for good address planning to improve manageability. When doing IPv6 address assignment planning it often makes sense to consider network and geographic topology to reduce the number of routing table entries that are needed.

Some cloud hosting providers may also make it easy to get a /64 (or even a /56) assigned and routed to each individual VM host (which can be very useful for container and pod networking systems like Kubernetes that want to give an address per container).

ISPs assigning a /60 (or even a /56) per subscriber means that each subscriber can carve out 16 to 256 individual subnets within their home network from the assignment. It is common for an ISP to have an /24 which means that it could assign 4.3 billion /56 prefixes to subscribers and still be doing so in a sparse manner which allows it to have regional structure to its addressing.

Another common pattern in enterprise networks is to use a few bits to indicate the type of network (eg, so “guest wifi” vs “server” vs “client” can be determined just by looking at the address in the context of that particular enterprise network).

Special Addresses and Prefixes

The IANA Special-Purpose Address Registry assigns a number of special-purpose addresses and prefixes. Some of them are described in more detail later.

Address Block	Name	Description and IPv4 Analogue	RFC
::1/128	Loopback Address	Similar to 127.0.0.1 in IPv4. Note that unlike IPv4 which has all of 127.0.0.0/8 available, IPv6 only has the single address of "::1"	[RFC4291]
::/128	Unspecified Address	Similar to INADDR_ANY (0.0.0.0) in IPv4.	[RFC4291]
::ffff:0:0/96	IPv4-mapped Address	Useful as a way to represent IPv4 addresses in IPv6 addresses, and often in the form with the IPv4 octets at the end. These aren't routable but are useful for cases where applications or data sets want to include a mix of IPv4 and IPv6 addresses in a single 128 bit value. As an example, ::ffff:198.51.100.7" is the same as ":ffff:c633:6407" and would represent the mapped IPv4 address of "198.51.100.7".	[RFC4291]
64:ff9b::/96 and 64:ff9b:1::/48	IPv4-IPv6 Translatator	Common prefix used for NAT64	[RFC6052] [RFC8215]
100::/64	Discard-Only Address Block	Used to blackhole traffic (eg, via RTBH)	[RFC6666]
2001:db8::/32	Documentation	When writing examples you should try and use this prefix, similar to 192.0.2.0/24, 198.51.100.0/24, and 203.0.113.0/24 in IPv4.	[RFC3849]
2002::/16 [3]	6to4	Used for an old but now deprecated transition technology called "6to4".	[RFC3056]
fc00::/7 (currently only fd00::/8 is defined)	Unique-Local (ULA)	ULA are non-globally routable addresses that can be used local to applications and sites. You randomly generate a 40-bit value and prepend fd00::/8 to get a hopefully unique /48. While there is an analog to RFC1918 private addresses, the uses are different and administrators should think long and hard before actually using ULA for anything.	[RFC4193] [RFC8190]
fe80::/10	Link-Local Unicast	Every link has one of these, but they are not unique and are *not* globally routable. They are discussed in more detail below.	

ff00::/8	Multicast	Specific multicast addresses and ranges are used heavily in IPv6 for a variety of purposes. Some of these are discussed below. Note that there is no broadcast in IPv6, so everything is done via multicast instead. See IPv6 Multicast Address Space Registry for more details.	[RFC3513] [RFC4291]

Link Local addresses and Scopes

Not all IPv6 addresses are globally scoped. In-particular, Link-Local Unicast addresses have a scope per link (eg, interface). A host will then have a unicast address within this fe80::/10 range per-interface in-addition to any globally scoped addresses. A host with only Link-Local Unicast addresses is unable to contact the outside world via IPv6 and should not try to do so. These Link-Local Unicast addresses are used for many purposes by the underlying IPv6 protocol, such as for getting globally scoped addresses.

The Link-Local address is usually generated as a function of the MAC address of that interface. Link-Local addresses are often also used for purposes such as next-hop routers, local DNS resolvers, and fixed services such as cloud metadata services.

Note that applications wishing to reach a link-local address will need to specify the interface scope. This is often specified add "ADDR%SCOPE", so "`fe80::abcd:abcd:1234:1234%eth0`" would specify that it was specifically on the "eth0" interface scope. Some tools take this via a separate argument. For example with ping you might use:

```
ping6 -I eth0 fe80::abcd:abcd:1234:1234
```

In the cases where this is being used with a URL, you'd put it in square brackets like "`https://[fe80::abcd:abcd:1234:1234%eth0]:8443/`" although not all browsers/clients support this.

Reverse DNS

Instead of in-addr.arpa, IPv6 reverse DNS lives in ip6.arpa. The address is reversed and then each 4 bits gets its own DNS label. For example, `2001:db8:abcd::1234` would have a PTR record under `4.3.2.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.d.c.b.a.8.b.d.0.1.0.0.2.ip6.arpa`.

(Similar to IPv4, both the `nslookup` and `host` tools will take an IPv6 address and do a reverse lookup.)

Useful tools for working with addresses

Before moving on, I have to call out the "sipcalc" tool. I use it all the time for performing math on IPv6 (and IPv4) prefixes.

Discovery, configuration, and ICMPv6

IPv4 uses ARP to discover other hosts on the network and to translate from Network Layer addresses to Link Layer (eg, MAC) addresses. It is also common in IPv4 to use DHCP to assign addresses to hosts and to provide them with other configuration information such as for nameservers.

IPv6 takes a different approach (and in some cases two annoyingly parallel approaches). There is no ARP in IPv6. Instead, hosts and routers use the "ND" protocol to discover each other and to find routers. This is based on ICMPv6 messages.

While IPv4 has the ICMP transport layer protocol for doing a range of things, IPv6 has the similar but distinct ICMPv6 transport layer protocol. Similar to ICMP in IPv4, ICMPv6 can be used for pings (echos) and traceroutes, as well as for communication things like unreachable destinations and refused connections. However, IPv6 also uses ICMPv6 heavily for neighbor discovery and network configuration.

The ICMPv6 Packet Too Big (PTB) messages are also critical for Path MTU Discovery (PMTUD), or signaling back that a packet couldn't be forwarded due to being too big. As IPv6 won't fragment in the middle of the network, it is critical to not drop (or not properly forward) PTB messages or otherwise connections can stall out if some packets exceed the path MTU, such as if some tunneling exists along the path.

For network configuration, IPv6 annoyingly has two parallel approaches, and not all client OS implementations support both fully, and neither covers all use-cases:

- IPv6 Neighbor Discovery (ND) and IPv6 Stateless Address Autoconfiguration (SLAAC) use the ICMPv6 messages for auto-configuring hosts. The network tells hosts their /64 prefix and the hosts select an address within that prefix. Nameservers can be configured with an RA RDNSS option. This approach is preferable for individual hosts as it "just works". However, there is no way to get entire prefixes delegated through this approach.
- DHCPv6-PD handles the case of getting a prefix from an upstream router, thus allowing a downstream router (such as a home gateway) to further subdivide it into (often /64) subnetworks. There is also a full parallel set of functionality within DHCPv6 to SLAAC+RDNSS but not all operating systems support it for configuring addresses to hosts (notably Android does not[5]).

[5] This is a long story and perhaps a future version of this footnote will expand on why…

IPv6 Neighbor Discovery (ND)

TODO – write this section

At a high-level, IPv6 uses ICMPv6 messages to find other devices and routers on-link. Messages are send using Link Local addresses (eg, in fe80::) and are either sent unicast to other Link Local addresses or are multicast to groups derived from Link Local addresses. For the moment, reading the RFCs might be a good way to understand this if more detail is needed.

IPv6 Stateless Address Autoconfiguration (SLAAC)

The primary way most hosts get their IPv6 addresses is via SLAAC. Hosts use ICMPv6 ND messages, such as Router Solicitation (RS), to get information back on available local network prefixes. Routers respond back with Router Advertisements (RA) that contain both a router preference level, a router lifetime (ie, how long the RA is valid for), and a Prefix Information Option (PIO) that contains available /64 prefixes.

For each prefix, hosts will generate an IPv6 address by using the network part of the prefix (the top 64 bits) and an IID value they construct in some way. The host will then perform Duplicate Address Detection (DAD) by sending multicast messages to see if anyone else is using that address. If not, the hosts proceeds with using it.

There are a variety of ways hosts may select their IID:

- The original way was to construct it deterministically from their MAC (or EUI-64) address. This has privacy challenges, especially for mobile devices such as laptops, as it means a host can be tracked around the internet based on its MAC address. The use of MAC-based IIDs also makes it much more feasible to scan IPv6 address space.
- Privacy Addressing effectively has hosts randomly select an IID and then rotate it at some frequency (eg, every 6 hours). This means hosts IPv6 addresses are constantly changing. This is the default in many modern OSes.
- Many OSes have a way to statically configure the IID value. For example, on Linux you can do:
    - ip token set TOKEN dev eth0
- (add in the newer deterministic one)

Note that hosts may have many addresses derived from SLAAC at any given point-in-time. This is to allow for gradual transitions between addresses and prefixes and to support graceful renumbering. Older addresses on a link will be listed as “deprecated” as they age out (but hopefully still work for existing connections) while newer addresses will get used for new connections.

Hosts may also have multiple prefixes on the same link or multiple links (eg, from multiple routers), and there are sets of heuristics for how clients will do Source Address Selection in the cases where they have multiple active non-deprecated addresses.

DHCPv6-PD

(writeme)

IPv6 and DNS

Whereas IPv4 uses the DNS “A” resource record type (RRTYPE) to resolve IPv4 addresses, IPv6 uses the “AAAA” RRTYPE to resolve IPv6 addresses.

- When the requested DNS record type is A, DNS returns IPv4 addresses
- When the requested DNS record type is AAAA, DNS returns IPv6 addresses

A dual-stacked hostname will have both A and AAAA records, returning both IPv4 and IPv6 addresses. Dual-stacked clients will look up both record types. Clients prefer either IPv4 or IPv6 depending on client-specific behaviors, which vary by browser and OS. Today, when dual-stacked clients look up AAAA records for IPv4-only sites, they receive a “no answer” response — when this happens, the dual-stacked clients use the IPv4 addresses from the A record.

For a dual-stacked website, this means that a name such as www.example.com would resolve to both A and AAAA records (perhaps at the end of a CNAME chain). For example:

```
www.example.com     A      192.0.42.2
www.example.com     A      192.0.42.5
www.example.com     AAAA   2001:db8:0:44::a11:aba3
www.example.com     AAAA   2001:db8:0:44::a11:aba4
```

Note that A & AAAA records can only be at end of a CNAME chain. You can NOT split off A vs. AAAA in CNAME chain.

For authoritative nameservers themselves to have IPv6 support, the name specified by the NS record (eg, ns1.example.com) would have both A and AAAA records. Different recursive resolvers have different behaviors as to how and when they prefer using A vs AAAA records on authorities.

Note that there is independence between which RRTYPE is being looked up, the IP version used to talk to a recursive resolver, and the IP version used by the recursive resolver to talk to its DNS authorities. For example, all of these are possible:

- AAAA lookup to a recursive resolver over IPv4, which then asks the authorities for the AAAA record over IPv6 (including the IPv4 address prefix in ECS)
- AAAA lookup to a recursive resolver over IPv6, which then asks the authorities for the AAAA record over IPv4 (including the IPv6 address prefix in ECS)

- A lookup to a recursive resolver over IPv6, which then asks the authorities for the A record over IPv4 (including the IPv6 address prefix in ECS)
- … (and all of the other possibilities)

A useful tool for debugging issues here is the whoami.ds.akahelp.net and whoami.ipv6.akahelp.net domains. For example:

```
# dig +short TXT whoami.ds.akahelp.net
"ns" "2001:db8::abcd"
"ecs" "203.0.113.0/24/0"
"ip" "203.0.113.132"
```

Where the recursive resolver talked to an authority over IPv6 but passed along IPv4 addresses in ECS.

Client and Server behaviors

Dual-stack client behavior and Happy Eyeballs

Dual-stack clients typically lookup both A and AAAA records when they believe they have a valid globally routable IPv6 address.

Clients will often prefer IPv6 if AAAA records are returned. The actual client/browser heuristics vary widely.

Some older clients (eg, some things written with old versions of Java) may deal poorly if an AAAA is returned and they are unable to reach it, such as due to broken connectivity on the client or server end. Depending on the nature of the failure, most clients should timeout and fallback to IPv4.

Most modern clients implement some form of “Happy Eyeballs” where they will kick off both IPv4 and IPv6 in parallel, often giving a bonus to IPv6 or waiting some time before starting IPv4. If you are implementing a new client, I’d strongly encourage reading and implementing “RFC 8305: Happy Eyeballs Version 2: Better Connectivity Using Concurrency” which is used by OSX/iOS.

One downside of Happy Eyeballs is that IPv6 connectivity can silently break and users won’t know the difference (minus some potential for delays).

Some clients such as some versions of Windows will do a connectivity check (eg, NCSI in the Windows case) and will only use IPv6 if it had success getting a resource over IPv6.

HTTP(S) Servers

It is important to keep in-mind that HTTP(S) servers will see the IP address of the end-point talking to them, either with the client's address or via a NAT44 or NAT64 or a proxy. Given a single connection, there is no way for a server to reliably determine the IPv4 address of a client over IPv6 or vice-versa.

Logs of servers typically want to continue to contain a client IP address in the same fields as before (eg, the c_ip field). Now the address can just be either an IPv4 address or an IPv6 address. (Checking if the address contains a ":" is often how the IP version of the address in logs gets determined, for better or worse.)

Note that proxies and NATs may result in some conversion. Proxies may put IPv6 addresses into X-Forwarded-For, even for IPv4 requests, so even IPv4-only servers should expect to sometimes see IPv6 addresses in "X-Forwarded-For".

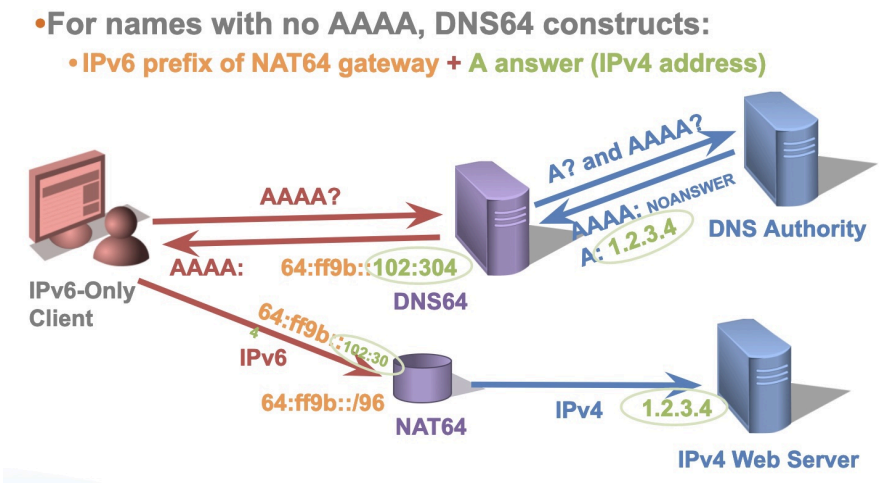
Transition Technologies

NAT64 / DNS64 / 464XLAT

As mentioned above, it is increasingly common to deploy clients into IPv6-only environments and use an IPv4aaS technology to access legacy IPv4-only resources. One of the most common is NAT64/DNS64/464XLAT which is used by some of the largest mobile networks in the world (including T-Mobile US, even for their Home Internet service). This model can also be useful in Cloud environments for running IPv6-only services that also need access to a few IPv4-only services (eg, github being the most commonly lamented).

This consists of two main pieces:

- NAT64: a NAT server that translates from IPv6 to IPv4. For packets incoming to it, the source address contains the actual client IPv6 address while the destination address includes a prefix of the NAT64 service (which is how the network routes to it) and a suffix of the legacy IPv4 address of the intended destination. For example, to reach "198.51.100.7" the destination would be 64:ff9b::c633:6407. The NAT64 retains a stateful NAT mapping per connection, as with a NAT44.
- DNS64: a DNS resolver that synthesizes AAAA responses from A records. When receiving a AAAA query it looks up the AAAA record. If one is found (i.e., a dual-stack or IPv6-only service) it is returned and the client connects to it. If none is found (i.e., an IPv4-only service), the resolver looks up an A record and constructs a AAAA response from it (such as 64:ff9b::c633:6407 from 198.51.100.7). Clients in this environment are generally IPv6-only so don't actually look up A records.



This works for most applications that work with hostnames, but some use IPv4 literals. For these, there are two approaches:

- 464XLAT: Software called a “CLAT” can run on client devices or on home gateway. It creates a local IPv4 stub network and translates IPv4 packets to IPv6 on the external interface, having the NAT64 prefix. Both Android and Windows include 464xlat implementations.
- “Bump in the API”: The OS networking stack implements an abstraction and will do synthesis to literal IPv4 addresses with a NAT64 prefix. Apple implements this and requires applications to be tested and work in IPv6-only environments.

There is also a useful special-use IPv4 address: ipv4only.arpa. It should only ever resolve to 192.0.0.171 and 192.0.0.170 via A records. If it has AAAA records as well, they were added by DNS64. This is used by the bump-in-the-API approach to determine the NAT64 prefix.

Useful Tools

Many of the tools useful for interacting with the network on IPv6 are the same as with IPv4, with some slight differences. Many newer versions of tools support both IPv4 and IPv6 addresses, often preferring IPv6, and with “-4” and “-6” options to force preference one way or another when provided with a hostname. Some older tools still require separate binaries for IPv4 vs IPv6 (eg, ping vs ping6).

Some common tools:

- mtr: trace+ping to IPv6 addresses
  - Use -4/-6 option to force a IPv4 or IPv6 usage on a dual-stack hostname
- ping and ping6:
  - Some operating systems now have “ping” support both (with -4/-6 options to control) while others keep them separate.
- curl: fetch resources via HTTP(S) and other protocols
  - Has a very good Happy Eyeballs behavior when using dual-stack hostnames.
  - -4/-6 options can force behavior

    - –resolve HOSTNAME:PORT:IPADDR can be used to override what IP address gets uses to connect to a hostname, but no square brackets are used which looks confusing. For example, "--resolve www.example.com:443:2001:db8::123" will use "2001:db8::123" for port 443 connections to www.example.com

- netstat: get network information
    - "netstat -r -n -6" will list the IPv6 routing table
    - "netstat -l -n -p" will list all listening sockets, both IPv4 and IPv6.
        - "::1:53" in the Local Address column means "port 53 listening on IPv6 localhost"
        - "::53" in the Local Address column means "port 53 listening externally and on all addresses"

- dig: can lookup AAAA, plus handles IPv6 authorities
    - Using -6 with +trace will force resolution over IPv6
    - Do: "dig +short www.example.com AAAA" to get just AAAA records for www.example.com

- host: on Linux, does both A and AAAA DNS lookups, and shows CNAME chains.
    - Example:

```
$ host www-dualstack.akamai.com
www-dualstack.akamai.com is an alias for
www-dualstack.akamai.com.edgesuite.net.
www-dualstack.akamai.com.edgesuite.net is an alias for
a152.dscg.akamai.net.
a152.dscg.akamai.net has address 80.67.64.114
a152.dscg.akamai.net has address 80.67.64.116
a152.dscg.akamai.net has IPv6 address 2001:418:2007:1::a88f:f11b
a152.dscg.akamai.net has IPv6 address 2001:418:2007:1::a88f:f133
```

- On Linux:
    - To list addresses on interfaces: ip -6 addr
    - To list routing info: ip -6 route

- tcpdump for looking at ICMPv6 packets on the wire to debug neighbor discovery:

```
tcpdump -i eth0 -n -vvv "icmp6 && ip6[40] == 134"
```

  Where "134" is router-advertisements but may be replace for other types such as:

```
unreachable: 1
too-big: 2
time-exceeded: 3
echo-request: 128
echo-reply: 129
router-solicitation: 133
router-advertisement: 134
neighbor-solicitation: 135
neighbor-advertisement: 136
```

- `rdisc6 eth0`
  - Performs router solicitation on the interface eth0 and prints out what it sees. Useful for debugging what is going on with SLAAC.

- `ip6tables`
  - Used for managing IPv6 firewall rules on Linux. Works the same as iptables and you will want many of the same rules. Note that you do not want to configure NAT. Instead, you likely wish to configure connection tracking to only allow inbound traffic (on INPUT and FORWARD) for established connections except for ports you explicitly want to accept.
  - Note that you generally will want to allow ICMPv6 through. Blocking ICMPv6 on the INPUT/OUTPUT chains will cause neighbor discovery (ND) to stop working. Blocking ICMPv6 can also break PMTUD in some cases as the ICMPv6 PTB messages will not be able to make it through.

Web based tools and resources

Test if you have IPv6 connectivity:

http://test-ipv6.com/ and https://test-ipv6.com/

(There are actually cases where it is worth trying both, at least in places where there may be transparent proxies such as airports.)

Get your IPv6 address via an HTTP request:

http://ipv6.whatismyip.akamai.com/
http://ds.whatismyip.akamai.com/

Acknowledgements

Thank you to the following people who made improvements or recommendations: Kyle Rose, and others.

Appendix: Further Reading

- Brian Carpenter's open source IPv6 Book
- The RFCs:
  - RFC 8200: Internet Protocol, Version 6 (IPv6) Specification
  - … and many more

Appendix: Other useful Akamai-related IPv6 resources

- An Overview of IPv6 on Linode
- Akamai Blog | 10 Years Since World IPv6 Launch
- Enable IPv6 for Your Akamai Edge Hostnames

Topics to Add

- 6RD
- Configuring firewalls and security products for IPv6
- Best practices for IPv6 in software
    - Default to IPv6 just working (enabled by default) but in a way that degrades gracefully when IPv6 isn't present or is broken
    - Treat IP addresses as IP addresses (eg, in log fields)