How We Explain IP Networks to Sound Engineers

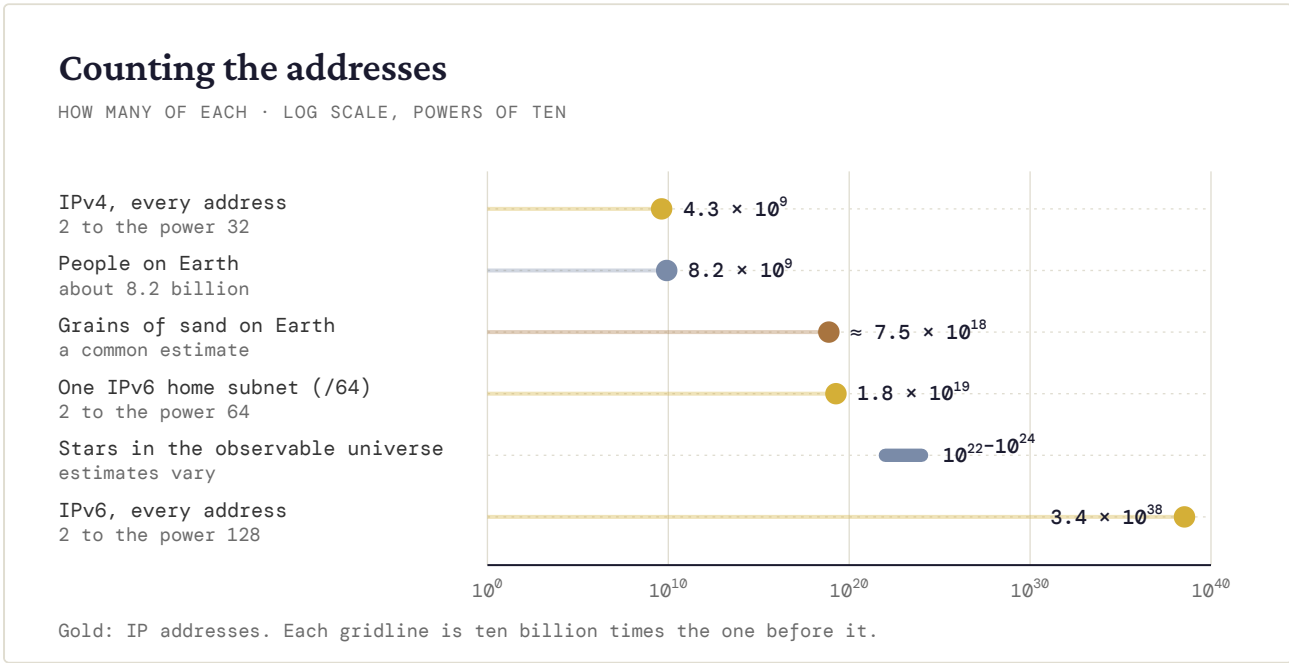
Kieran Walsh · November 2026 · kdwalsh.com

A follow-up to Explaining Digital Audio to Musicians: the myths about IP addressing, NAT, UDP and IPv6 that still trip up the AV industry, and why the next protocol is already quietly carrying most of your traffic.

Following on from Explaining Digital Audio to Musicians, it's time to tackle how we explain IP networks to sound engineers. Despite spending years authoring substantial training materials to demystify this, there are still far too many myths circulating, chief among them myths about IP address space.

I was dismayed to see someone describe IPv6 as "the broccoli of the internet": the idea that it's supposedly good for us, and maybe it will do something useful later. That is factually, entirely incorrect.

Let's backtrack and look at the broad numbers. At present, there are about 4.3 billion possible IPv4 addresses, which comes from it being a 32-bit address space.[1] There are more than 4.3 billion people on the planet, meaning we cannot even have one IPv4 address each. That matters enormously because of the original intent of an IP address in the first place.



IPv4 can't give everyone on Earth one address. A single IPv6 home subnet outnumbers the grains of sand.

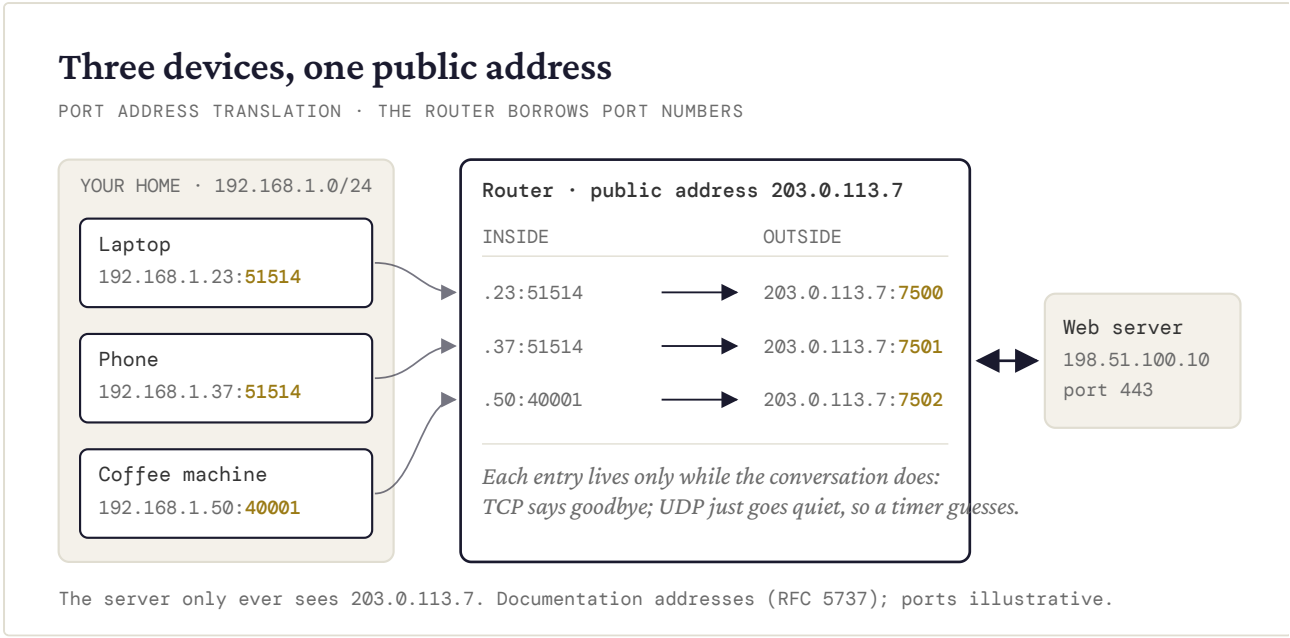
The Myth of "Private" Space and the NAT Bodge

One of the major misconceptions in networking is that there is such a thing as a "private IP address." Technically, there isn't. The only truly private IP address is one assigned to an interface that isn't plugged in. The confusion stems from IETF RFC 1918 [1], which defines private IP address space. All that actually means is that these are blocks of addresses not assigned to, or carried directly over, the public internet. RFC 1918 was the IETF conceding that we were going to run out of IPv4 addresses and needed a temporary stopgap while moving forwards towards a bigger address space. The document itself is explicit about this: Section 2 cites the community's concern that *"globally unique address space will be exhausted,"* while Section 7 concludes that conserving addresses through private space is simply there to *"effectively lengthen the lifetime of the IP address space"* [1].

In the mid-1990s, that replacement was settled upon as IPv6, a 128-bit address space [2].[2] Going from 32 bits to 128 bits doesn't sound like a huge leap on paper, but the scale is vast. IPv4 is a scarcity problem. IPv6 is still just IP addresses, only longer, but there are enough of them to more than amply cover every single star in the observable universe. Put slightly more sympathetically, there are more IPv6 addresses than there are grains of sand on planet Earth. Since microchips are essentially made of sand (and you need more than one grain of sand to make a microchip), we aren't going to run out anytime soon.

The whole architecture of IP comes out of the 1970s, where the fundamental principle of data communication was end-to-end routability, which requires end-to-end IP addresses. RFC 1918 breaks that model by introducing private subnets that cannot be carried across the public internet.

To accommodate the vanishing supply of IPv4 addresses, a very clever workaround was introduced: what is commonly called Network Address Translation (NAT), originally laid out in RFC 1631 [3]. Like RFC 1918, its own authors framed NAT as a strictly short-term solution to address depletion. In their conclusions (Section 4), they explicitly warned that NAT breaks certain applications and causes *"problems with SNMP, DNS, … you name it,"* and noted that future applications needing a rich traffic matrix (*"for instance, distributed resource discovery"*) would end up *"making long-term use of NAT unattractive"* [3]. Hold that thought about discovery; we'll come back to it. Strictly speaking, what we engage in is Dynamic Port Address Translation (PAT) overload, which the IETF later formalised as NAPT [4]. You substitute your public IP address for the private IP address on your local network, and because we have 16 bits' worth of Layer 4 port numbers[3], and a typical domestic machine uses far fewer concurrent connections than that, we use those ports as quasi-IP addresses. The router swaps and maps an ephemeral port number (for example, mapping 7500 to 7501), maintains that state only while the transaction is happening, and closes it down once the transaction finishes.



Port numbers pressed into service as quasi-addresses: the router keeps the table, and the outside world sees one address.

That works well enough for an acknowledged connection like TCP. With UDP, however, there is no formal teardown, so the router has to run timers to guess when a session is over, making the whole thing even more complicated. In reality, it is a horrible, dirty botch. The only reason it survives is that most people do not run web servers. If everyone were running servers and transacting symmetrically, we would have been doomed decades ago.

End-to-End Routability and Endpoint Security

When we use IPv6, we get end-to-end routability back. Every single machine in an IPv6 space has a globally unique IP address[4], which is right where we started with IPv4. In the 1970s, Layer 3 was designed for exactly this; there were so few computers connected to each other that it was an entirely achievable goal. Showing my age a bit, when I went to university in 1998, every single machine in the building had its own public IP address because academia had such vast address space allocations. They were doing it properly. Even in IPv4, it was achievable.

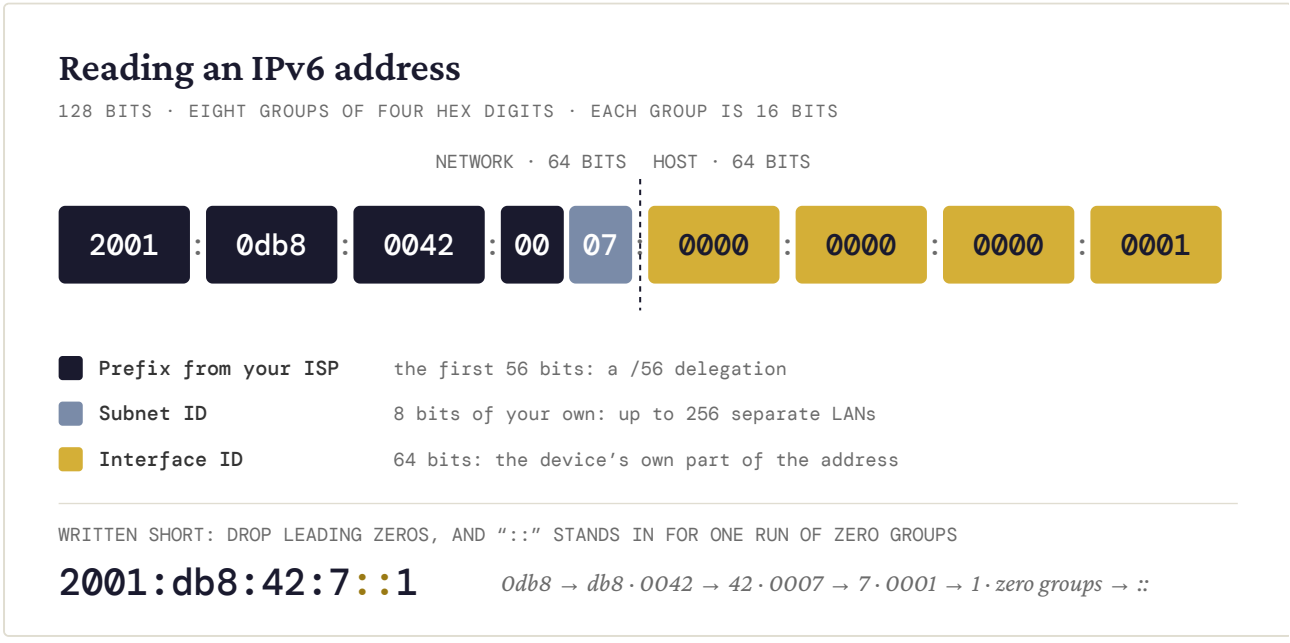
The main concern people raise with doing it the right way is security. In an end-to-end model, you rely on the host firewall on every single endpoint, and you still should. NAT is not a firewall. NAT is not a security feature. As stated earlier, NAT is simply a bodge to deal with address scarcity.

So why do we want end-to-end routability? For one, it makes things more efficient. As for the fear that every hacker on the planet is suddenly going to be staring at your specific IP address: remember we have more IPv6 addresses than there are grains of sand on Earth. Given that most people in audio-video networking seem oddly allergic to setting up DNS, it's hard to see how anyone would even begin to find you. Once we look at the sheer scale of even your own personally allocated IPv6 subnet, scanning for a host becomes literally like looking for a needle in a haystack [5].

Counting in Hexadecimal

The principal objection people have (and there is some emotional legitimacy to it) is that IPv6 has been kicked around, abused, and explained incredibly badly. The analogies don't really hold up, so I am sorry I even used the grains of sand on planet Earth; it's factually correct, just not very helpful. Fundamentally, there is a problem of counting. We look at dotted decimal, which I have to be careful not to ridicule: IPv4 is addressed as four octets, each running from 0 to 255, to represent 32 bits. Decimal doesn't count like that: the next number after 255 is 256, not 1.0. It is simply a reasonably human-readable way of avoiding hexadecimal.

IPv6 is continuous and expressed in hexadecimal. I've used hexadecimal for a long time; I don't particularly like it, but it makes more sense as a sequential way of counting than Roman numerals, which don't even have a concept of zero. That is precisely why we became far more comfortable with Arabic numerals and adopted them almost globally, even though we can still glance at a clock face with Roman numerals and tell the time. In IPv6, the address is written as eight groups of four hex digits, each group counting continuously from 0000 to ffff and separated by colons. Because we are dealing with such large numbers, there are lots of zeros, so IPv6 gives us two shorthand rules: we drop leading zeros, and a double colon (::), used once per address, means everything in between is zeros [6].[5] Repeating strings of zeros across an address is impossible for a human to read and completely unhelpful.



Network on the left, host on the right, split straight down the middle. The shorthand does the rest.

On the security side, it's completely normal for an endpoint to run a straightforward default-deny rule. Everyone throws this around like some kind of magic secret weapon, just like NAT, when it is actually very simple: if you aren't a server, you drop any unsolicited inbound traffic. If you didn't start the conversation, it isn't a reply to anything, so it gets dropped. That's the first rule of firewalling: don't accept connections

you didn't solicit. For a web server, you do what's called pinholing: listening on port 443 for HTTPS, maybe port 80 for HTTP, or a specific port for SSH. Otherwise, the default-deny rule stands, and any sensible operating system rightly makes you put in some effort to punch those holes.

Routing, Wire-Speed Silicon, and Multi-WAN

We also haven't thrown away anything fundamental from IPv4; there's nothing to throw away. I saw someone claim that running dual-WAN is difficult on IPv6, which simply isn't true. An IP address connects to another IP address, and a router connects networks together. Both IPv4 and IPv6 use the exact same concept of an upstream default gateway: if a router doesn't have a direct match for a destination, it hands the traffic upstream to someone who might, and so on. Whilst routing is straightforward (if the network is entirely under your control, you can use any interior routing protocol and it will work beautifully), when you are heading out to the public internet, ISPs talk BGP exclusively. BGP is a fantastic routing protocol, though it takes some serious setting up. Nevertheless, multi-homing in and of itself is not a blocker; it simply comes down to understanding how routing tables are communicated, and that is just as true in IPv4 as it is in IPv6. The one real wrinkle is that without your own address space, two ISPs mean two prefixes on every host, so the router has to send each packet out of the link its source address belongs to [7]. That's source-based routing, not magic, and it's a solved problem.

It is also worth putting the hardware into historical context. In the 1990s and early 2000s, the switching speed of silicon was genuinely slow, and routing was a heavy bottleneck. In the core, line-rate routing has been around for a while; what changed everything at the edge was cheap switch silicon with fast SerDes, putting wire-speed routing into small networks that could never afford it before. As a result, worrying about thousands or even millions of packets per second is rapidly disappearing into the rearview mirror. That is not a critique of the people building networks in the 1990s and early 2000s; it simply wasn't physically possible with the silicon of the day. Now it is, meaning many of the old compromises we used to make out of necessity no longer apply.

The Silent Success Story and the UDP Myth

The other myth about IPv6, parroted by everybody, is that nobody is using it. That is the biggest lie of all. Google's own measurement passed 50% in March 2026 [8], and Google can't even see China, which reports 865 million active IPv6 users [9]. By head count, IPv6 is now the majority. On cellular networks, it is almost certain your phone is connecting over IPv6, and across many developing economies, as well as massive markets like India and China, IPv6 is the go-to. It has become increasingly hilarious to ask people who know a bit about networking, "Have you got IPv6?" only for them to say no. When I ask them to open their IP allocation, they read out 192.168.1.23 followed by "some stuff underneath that looks like letters and numbers with a bunch of colons." If it starts with fe80, that's a link-local address; if it starts with a 2 or a 3 (like 2a05), that's a globally unique public IPv6 address [10].[6] It's ubiquitous. Most people at home don't even notice, and the big hyperscalers like Google and Amazon prefer connecting over IPv6 because

it's cheaper (Amazon now charges its own cloud customers for every public IPv4 address [11]), meaning much of your on-demand streaming and broadcast traffic is already carried over IPv6. It is the silent success story nobody realised was working, simply because it worked seamlessly.

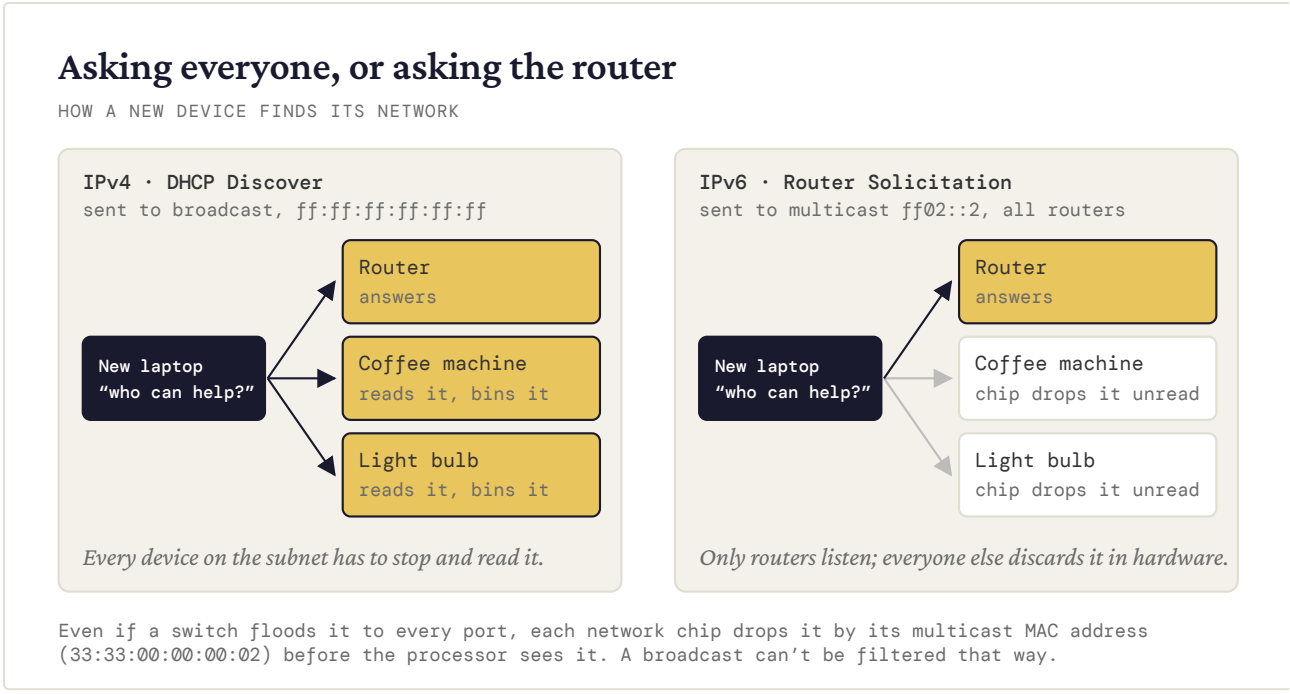
The same people who insist nobody is using IPv6 are usually the same people who claim that UDP is an "unreliable" method of communication. Strictly speaking, *unreliable* in networking simply means *unacknowledged*. If that is your definition, fair enough. If you think *unreliable* means *dodgy*, however, you are wide of the mark.

Most people are perfectly happy to use online banking. They may assume it all runs over TCP (and indeed, the website itself is typically delivered over TCP), but the bank's own back-end infrastructure is almost certainly running on UDP. Modern large-scale, multi-tenanted data centre networks communicate internally over VXLAN, and VXLAN is entirely UDP-based [12].[7] Whatever rides inside the tunnel, even TCP, the fabric itself carries it as UDP and leaves reliability to the endpoints. And closer to home: AES67, RTP and PTP are all UDP. Every sound engineer reading this already trusts it with live audio. If UDP is reliable enough for your bank to make sure your money is in your account and their ledger balances, it is certainly good enough for the rest of us. Yes, there are checks and balances (pardon the pun), but the idea that TCP is "reliable" and UDP is "unreliable" is a matter of semantics, not physics. UDP is far more efficient, uses significantly less power at scale, and is much faster because it doesn't sit around waiting for acknowledgements; it just hammers the data through. Looking after data integrity at the far end is an application-layer concern, not something that inherently has to live inside the transport stack. That is exactly how QUIC, the UDP-based transport underneath HTTP/3, works [13].

Multicast, Coffee Machines, and the /64 House Party

So there must be a reason why the big players prefer IPv6, and saving money is a good one. One of the things I like most about it is that IPv6 does not have the concept of broadcast. Back when I was first doing my CCNA, the instructor (quite rightly at the time, conceptually at least) taught us that multicast was horrible, behaved just like broadcast, and that we should avoid both. That is not the case in IPv6. In IPv4, multicast was a bolt-on, which is why it was so problematic and often devolved into broadcast-like behaviour. IPv6 baked multicast in from day one, and when done properly, multicast is a beautiful thing.

Compare how a device gets an IP address. In IPv4, when you solicit a DHCP-served address, you send a broadcast message [14], waking up every single device on your subnet to ask for an IP address. My coffee machine (and yes, I have an IP-connected coffee machine) is not a DHCP server. I am just annoying it for no good reason. The same goes for all my IPv4-connected light bulbs; none of them are DHCP servers either. IPv6 fixes this cleanly. Instead of shouting at everyone, your device calls a special all-routers multicast address[8], asking the routers, the things that actually do the job, for the network prefix [15]. You stop cluttering up the network with loads of spurious, ridiculous traffic, and when you build big networks, that actually matters.



The coffee machine is not a DHCP server. IPv6 stops asking it.

Then there is the sheer scale of how subnets work. Best practice for a domestic installation is a /56 delegation [16], and the smallest standard LAN subnet available in IPv6 is 64 bits long. The address is split straight down the middle: the first 64 bits are the network prefix, and the last 64 bits are the host space. None of your variable-length subnet masking headaches going on here. To put a single /64 subnet into perspective: if I invited the entire population of planet Earth round for a party (let's be generous and call it nine billion people), I could hand out two billion IP addresses to every single one of them.[9] If I were hosting a party for everyone on Earth, I would be a lot more worried about making sure water and toilet facilities were available than handing out IP addresses, but in theory, it could be done.

DNS for Infrastructure, Brokers for Discovery

So we're dealing with billions of IP addresses each, all our own, all globally unique, and that's wonderful. Which brings us back to the obvious question: how do we actually find anything? Hexadecimal, as I said before, is obscene. It's a computer thing; computers are good at it, and humans (this human included) are not. That is where DNS comes in. DNS maps a human-readable text name to an IP address: an A record in IPv4, and a AAAA record in IPv6 [17]. (I go into this in more detail in DNS with IPv4 and IPv6.)

Now, if you publish those DNS records to a public DNS server, you've suddenly exposed all your hostnames and mapped them to globally routable IP addresses. Whilst security through obscurity is not a thing (and it is very important to underline that), advertising your internal infrastructure to the entire world is hardly wise. Fortunately, you don't have to. You can run dynamic DNS without sharing it globally, or run an internal DNS server within your subnet that doesn't propagate out, letting you look up devices by name locally. However you implement it, I cannot encourage you enough to set up DNS earlier rather than later. It makes life immeasurably easier and stops you wasting time trying to remember numbers.

That said, it is crucial to draw a line between infrastructure and discovery. DNS is for infrastructure. It is essential for reconciling a name to an address, but it has been leaned on as a one-trick pony for far too long. Much of what gets called "discovery" has relied on mDNS and DNS-SD, implemented by Apple as Bonjour [18][19]. It was originally built as a convenient way to find a printer on a local network, and it mostly works (usually any hiccups are down to printers being flaky rather than the technology itself). Standard DNS, meanwhile, was never built for rapid, dynamic discovery: answers are cached all over the world until they expire, so a change can take a long time to be seen: historically hours or days. Even though it's much quicker today, it is still a heavy way of telling the whole world that something has changed.

Discovery needs to be much faster. Knowing *what* a device is, along with its live status, capabilities and other details, should be far more dynamic than a static infrastructure map. For practical, modern deployments, and across the Internet of Things, there is a much lighter-weight alternative: a publish-subscribe broker model. Take MQTT, the wonderful protocol invented in 1999 by Andy Stanford-Clark of IBM and Arlen Nipper of Arcom to monitor Phillips 66 oil pipelines[10], which is now an OASIS standard [20]. Whilst MQTT doesn't do discovery inherently on its own, pairing it with a simple database gives you discovery that is vastly more rapid, infinitely flexible, secure and, critically, unicast, meaning it scales globally. All those smart light bulbs I mentioned earlier, and even my coffee machine, rely on a publish-subscribe broker model. Even where a specific device isn't running MQTT directly, that is where its roots lie, and it is how you achieve fast, strictly controlled discovery for trusted users while keeping untrusted ones out. (More on that in MQTT – Large Scale Control.)

Evolving Conventions

Ultimately, none of us want to remember numbers anyway. As a child of the 1980s, I can still remember the direct-dial telephone numbers of my childhood friends, yet I have completely lost the ability to memorise anyone's number today. From Rolodexes in the 1980s, to digital assistants, to the contacts list in every modern mobile phone, we have always offloaded numbers to tools so we can just tap a name and talk. DNS, and the modern discovery tools built alongside it, are simply that telephone directory for your network.

In the end, this is simply the nature of how conventions evolve. IPv4 is pretty much concluding, and IPv6 is not the final destination either; we don't yet know what comes after it. It is no different from how we write language and numbers down. We moved from Roman numerals to Arabic numerals because they work better. I have my greatest facility in the Latin alphabet, which is somewhat ironic, given that the word *alphabet* is inherently Greek and comes from a completely different character set. And then you have something as clever as the Cyrillic alphabet, built largely on Greek letters and adapted to fit the sounds of the Slavonic languages it was made for. It is all about updating and continual modernisation.

We want to deal with the world in a human way, backed by the convenience of technology. More importantly, technology is supposed to save us time, freeing us up to get on with the creative, interesting work that humans are actually good at. Moving forwards with IPv6 isn't about eating your vegetables; it's

just the next sensible step in letting machines handle the housekeeping they are better at, so the rest of us can get back to doing our jobs.

> ***A quick note on honesty:*** *when I first drafted this, kdwalsh.com was IPv4-only. My hosting provider of many years would only give me IPv6 if I paid extra, despite it being cheaper for them to run. So I moved. The site now runs on AWS with IPv6 as standard: you may well be reading this over it.*

Notes

1. Specifically, 2^{32} = 4,294,967,296 total IPv4 addresses. Once reserved blocks (private space, loopback, multicast and experimental ranges) are excluded, roughly 3.7 billion publicly routable unicast addresses remain.
2. IPv6 was chosen as "IPng" in 1995 (RFC 1752), first specified in RFC 1883 (December 1995), revised in RFC 2460 (December 1998) and made a full Internet Standard in RFC 8200 (July 2017). A 128-bit space provides $2^{128} \approx 3.4 \times 10^{38}$ addresses (340 undecillion). Estimates for the number of grains of sand on Earth run from roughly 7.5×10^{18} to 10^{20}.
3. A 16-bit port field yields 65,536 port numbers per transport protocol (0–65535), of which IANA designates ports 49152–65535 as dynamic or ephemeral.
4. IPv6 does have Unique Local Addresses (fd00::/8, RFC 4193) for internal use, but they are designed to sit alongside global addresses rather than hide behind NAT, and their random prefixes make collisions between networks vanishingly unlikely.
5. RFC 5952 defines the canonical text representation for IPv6 addresses, ensuring leading zeros are suppressed and the "::" shorthand is applied at most once per address to eliminate ambiguity. The examples in the figures use the documentation prefix 2001:db8::/32 (RFC 3849) and the IPv4 documentation ranges (RFC 5737).
6. Under RFC 4291 and IANA allocation policy, Global Unicast IPv6 addresses are currently assigned from the 2000::/3 block (prefixes beginning with 2 or 3), while link-local addresses occupy fe80::/10.
7. VXLAN (RFC 7348) encapsulates Layer 2 Ethernet frames directly inside UDP datagrams (IANA destination port 4789), serving as the standard overlay fabric in modern multi-tenant cloud and enterprise data centres.
8. Specifically ff02::2, the link-local all-routers multicast address defined in RFC 4291, used by Router Solicitation messages (RFC 4861) so only routers process the request. Even where a switch floods it to every port, each device's network interface discards it in hardware by its multicast MAC address, 33:33:00:00:00:02.
9. A /56 delegation provides 256 separate /64 subnets. A single /64 contains 2^{64} = 18,446,744,073,709,551,616 addresses, roughly 2.05 billion addresses per person for a population of 9 billion. While /64 is the standard LAN boundary (and required for SLAAC), point-to-point router links may use /127 prefixes (RFC 6164).
10. Created in 1999 by Andy Stanford-Clark (IBM) and Arlen Nipper (then of Arcom Control Systems) to monitor oil pipeline telemetry over low-bandwidth, high-latency satellite links for Phillips 66; subsequently standardised by OASIS. See Ian Craggs, The History of MQTT, Part 1: The Origin (HiveMQ), and Why the MQTT Protocol is So Popular (Automation World).

References

1. Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G. J., & Lear, E. (1996). *Address Allocation for Private Internets* (RFC 1918). IETF. rfc-editor.org/rfc/rfc1918
2. Deering, S., & Hinden, R. (1998; updated 2017). *Internet Protocol, Version 6 (IPv6) Specification* (RFC 2460 / RFC 8200). IETF. rfc-editor.org/rfc/rfc8200
3. Egevang, K., & Francis, P. (1994). *The IP Network Address Translator (NAT)* (RFC 1631). IETF. rfc-editor.org/rfc/rfc1631
4. Srisuresh, P., & Egevang, K. (2001). *Traditional IP Network Address Translator (Traditional NAT)* (RFC 3022). IETF. rfc-editor.org/rfc/rfc3022

5. Gont, F., & Chown, T. (2016). *Network Reconnaissance in IPv6 Networks* (RFC 7707). IETF. rfc-editor.org/rfc/rfc7707
6. Kawamura, S., & Kawashima, M. (2010). *A Recommendation for IPv6 Address Text Representation* (RFC 5952). IETF. rfc-editor.org/rfc/rfc5952
7. Baker, F., Bowers, C., & Linkova, J. (2019). *Enterprise Multihoming using Provider-Assigned IPv6 Addresses without Network Prefix Translation: Requirements and Solutions* (RFC 8678). IETF. rfc-editor.org/rfc/rfc8678
8. Google. *IPv6 Statistics*. google.com/ipv6/statistics.html; see also Ford, M. (2026). 18 Years Later, IPv6 Reaches Majority. Internet Society Pulse.
9. APNIC Blog (2025). China reports major IPv6 growth, summarising *China's IPv6 Development Report (2025)*.
10. Hinden, R., & Deering, S. (2006). *IP Version 6 Addressing Architecture* (RFC 4291). IETF. rfc-editor.org/rfc/rfc4291
11. Barr, J. (2023). New – AWS Public IPv4 Address Charge + Public IP Insights. AWS News Blog.
12. Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., & Wright, C. (2014). *Virtual eXtensible Local Area Network (VXLAN)* (RFC 7348). IETF. rfc-editor.org/rfc/rfc7348
13. Iyengar, J., & Thomson, M. (Eds.). (2021). *QUIC: A UDP-Based Multiplexed and Secure Transport* (RFC 9000). IETF. rfc-editor.org/rfc/rfc9000
14. Droms, R. (1997). *Dynamic Host Configuration Protocol* (RFC 2131). IETF. rfc-editor.org/rfc/rfc2131
15. Narten, T., Nordmark, E., Simpson, W., & Soliman, H. (2007). *Neighbor Discovery for IP version 6 (IPv6)* (RFC 4861). IETF. rfc-editor.org/rfc/rfc4861
16. RIPE NCC (2017). *Best Current Operational Practice for Operators: IPv6 Prefix Assignment for End-Users* (RIPE-690). ripe.net/publications/docs/ripe-690
17. Thomson, S., Huitema, C., Ksinant, V., & Souissi, M. (2003). *DNS Extensions to Support IP Version 6* (RFC 3596). IETF. rfc-editor.org/rfc/rfc3596
18. Cheshire, S., & Krochmal, M. (2013). *Multicast DNS* (RFC 6762). IETF. rfc-editor.org/rfc/rfc6762
19. Cheshire, S., & Krochmal, M. (2013). *DNS-Based Service Discovery* (RFC 6763). IETF. rfc-editor.org/rfc/rfc6763
20. Banks, A., Briggs, E., Borgendale, K., & Gupta, R. (Eds.). (2019). *MQTT Version 5.0* (OASIS Standard). OASIS. docs.oasis-open.org/mqtt/mqtt/v5.0

© 2026 Kieran Walsh. Online version with links: https://kdwalsh.com/articles/how-we-explain-ip-networks-to-sound-engineers/
Contact: website@kdwalsh.com · +44 1273 117672