# IPv6 – that looked too easy!

The cold turkey from being “sold” all your life!

## That looked too easy?

Well, I touched on the key components. However – it’s still straight forward!

First, we need to compare how a DHCP address is “born” in IPv4.

1.New device joins network (plugged in)

2. New device issues a broadcast frame (remember it doesn’t have an IP address yet – that’s what we are doing) The Broadcast frame is a “DHCP DISCOVER” frame. This is a little silly… because by broadcasting it hits all the other devices in the network segment… my coffee machine doesn’t do dhcp address allocation (it’s a dhcp client no server) so why are you asking it?

### Wait! Your coffee machine has an IP address?

Anyone who has been to my home knows there are a few gadgets. One friend quipped “but you haven’t got a smart fridge!” My reply was – yes I have it’s 4 floors down (I live above a large supermarket – they have huge databases that manage stock ordering, so kinda the “smartest” fridge!)

Anyway – after the new device has solicited a DHCP address from (effectively) all the light fittings, the speakers, the fans, the light switches, the heating system, the washing machine, the kettle etc. it gets a DHCP offer from the one device that can – the DHCP server.

It’s like the joke about the viola player trying to buy a new A string, only to be told that they can’t have it, because this is a fish n chip shop! I hope this illustrates the absurdity of broadcast traffic, and why IPv6 banned it!

After the offer is received the client decides whether it “likes” the offer, it checks using ANOTHER broadcast message – an ARP to see if anyone else is using that IP address (the server “should” have checked before making the offer – but it’s only a “should” not a “must” – this is “standards” jargon at its finest.) It is of course possible that the client has received multiple offers (lucky client) but being a monogamous client, it only sends a request for one of the offers. Again, this is broadcast, so any other potential servers know they have not had their offer taken up.

The server responds with an ack(nowledgement) and now the client has their own unique IP address!

## Contrast this with IPv6

In this version, the clever violinist goes to the right shop. It wants an E string. Which the violin string shop happily sells them. In IPv6 the client calls the “all routers” ip address. It doesn’t bother anything else!

I’m desperately trying to avoid making this complicated. However, there are two methods for IP address assignment available in IPv6 (don’t worry, they play nice with each other).

## Method 1: SLAAC (Stateless Address Autoconfiguration)

The elegant one. Device sends Router Solicitation to ff02::2. Router responds with prefix. Client generates its own address - traditionally using EUI-64 (MAC address + FF:FE insertion), but modern systems use **privacy extensions** - random addresses that rotate every few hours. No tracking, no database, no begging.
DNS servers? Router Advertisement can include them (RDNSS option), or you combine SLAAC with stateless DHCPv6 (see Method 1.5 below).

## Method 1.5: SLAAC + Stateless DHCPv6 (the hybrid)

SLAAC for address, DHCPv6 for DNS/NTP/domain name. Router sets the O-flag ('Other configuration') in its advertisement. Still no address database, but clients get the infrastructure details they need.

## Method 2: DHCPv6 (the enterprise concession)

Much more like IPv4. Central server assigns addresses, maintains logs, tracks lease times. Clients can update DNS records (if you've configured dynamic DNS). Management gets their audit trails. Security teams can correlate addresses with actions.
Router sets M-flag ('Managed') to tell clients: 'Don't self-configure, ask the DHCPv6 server.'

## Does this actually matter?

In short, not really to the user. Yes, you can do Static IPv6 (just don't… seriously). Also consider that IP addresses are like driveways. Not in and of themselves secure. Yes, some driveways have gates, others don't. It's like how do you know you have a drummer at your front door? The beat slows down, and they don't know when to come in! (and you still don't let them in!)

## IP addresses don't have anything to do with security!

This detail is where secure certificates come in, which are a completely different matter, and are actually secure – think massive gates, drawbridge, and crocodile infested moat would be easier to traverse! This is because, by the time you crack a 4096 bit x509; The gates will have rusted, the moat will have dried out, the crocodiles will have long since died, and so will have YOU!

# 2 pages in and I get to the interesting bit!

Now we have extolled the virtues of IPv6 and debunked IP address as security. Lets do something practically useful!

# DNS- how we get rid of IP addresses!

If there wasn't DNS, I'd have no visitors to my site. People aren't any good at remembering numbers. People work with real words! This is why we have www.google.com, www.amazon.com and www.kdwalsh.com one is the apotheosis of search goodness, the other is shopping, and the third, some silly middle aged ranty geek! DNS is the telephone directory for the Internet. Just like a telephone directory it has a name (an A record in IPv4 and an AAAA record – its IPv6 equivalent) this points your browser at the ip address to access the pages.

Yes, we can do dual stack DNS because we have A records and AAAA records. We have other types to make stuff smoother, but these are the fundamental ones. It is entirely possible to have your device IP address populate the DNS server with your IP address. Remember what I said about IP addresses not being a security feature. Well, your IP address in V4 will most likely be an IETF RFC1918 private IP address – 192.168.0.0 – 192.168.255.255 or 10.0.0.0 – 10.255.255.255 or 172.16.0.0-172.31.255.255 this means they can't be accessed from the Public Internet.

It is still questionable whether it is a good idea to advertise all your "stuff" to the world. It gives away whether there is anything worth stealing. This is why we have Internal DNS for device-to-device communication – yes, you can fire up your own DNS and no more IP addresses!

DNS is not inherently secure, not for reading anyway. Its point is advertising, Years ago I spoke to someone who tried to say that they were doing "stealth marketing" I'm still unsure whether they were serious!

# Insecure, be careful what you advertise, sounds dodgy

Well, yes and no. a kitchen knife can be used to make a delicious meal, or it can be used as a weapon. It is all about chosen use of tools. Public DNS is essential. Private DNS, this is where we should be more careful.

## Device Discovery

In 2026 using DNS to advertise devices and components of said devices is inelegant. We should use a secure method. MQTT is one such method. We can find out about the deep features of a device (and even its existence) on a "need to know basis". Using DNS for this was the only viable way in 1998 to, say "find a printer" and even then, the printer was probably disappearing (again!) This was Bonjour (apple's name) or mDNS (multicast DNS) or avahi (Linux daemon). This is because MQTT was owned by IBM, and when the patent ran out Only WhatsApp, Facebook Messenger, Google Home, Alexa et al (you know those tiny insignificant companies) used it. We all know WhatsApp etc can be very secure. Also, these services reach global scale. So why do we have variations on a theme of printer discovery? This must be some long-lasting homage to Xerox in gratitude for Ethernet, obviously.

# Conclusion – DNS is very useful it's how the world works

Yes, DNS IS useful, but applied wrong it's potentially damaging. The same is true for IP addresses.

- DNS points us at stuff others want us to find quickly and easily
- DNS is bad for nosey people finding out stuff you might prefer they didn't know
- MQTT can be secured nicely – TLS encryption, but DNS to find the broker
- IPv4 has broadcast think viola in a fish and chip shop
- IPv6 doesn't bother the coffee machine for an IP address.
- DNS (in either flavour) gets rid of numbers
- MQTT gives information on a need-to-know basis

# Appendix – IPv6 is hard and will take time to develop

This is quite false! Below is the difference between IPv4 and IPv6 from a software code perspective. All the clever software functions remain the same, we just need to add the following to make it “dual stack” and then, maybe later we take out the IPv4 bit.

c

```
/* These libraries are tried and tested in billions of devices - they work! */

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>


/* IPv4 Socket Setup */


int sockfd;                     // This is our socket - think of it as a phone line

struct sockaddr_in server_addr;     // This holds the address information


// Create socket (open the phone line)

sockfd = socket(AF_INET, SOCK_STREAM, 0);


// Setup address (configure our phone number and settings)

server_addr.sin_family = AF_INET;           // We're using IPv4

server_addr.sin_port = htons(8080);         // Port 8080 (like an extension number)

server_addr.sin_addr.s_addr = INADDR_ANY;     // Listen on all network interfaces


// Bind socket (claim this address as ours - like registering the phone number)

bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr));


// Listen (wait for incoming connections - phone is ready to ring)

listen(sockfd, 5);
```

**To bring in IPv6 we simply add a "6" in three places:**

c

```
/* Same proven libraries - no new dependencies needed */
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

/* IPv6 Socket Setup - Dual Stack */

int sockfd;                    // This is our socket - think of it as a phone line
struct sockaddr_in6 server_addr;    // This holds the address information (Added "6")

// Create socket (open the phone line)
sockfd = socket(AF_INET6, SOCK_STREAM, 0);   // Added "6" - now it's IPv6

// Setup address (configure our phone number and settings)
server_addr.sin6_family = AF_INET6;         // We're using IPv6 (Added "6")
server_addr.sin6_port = htons(8080);        // Port 8080 (Added "6")
server_addr.sin6_addr = in6addr_any;        // Listen on all interfaces (Added "6")

// Bind and listen - COMPLETELY UNCHANGED
// Claim this address and wait for connections - exact same code!
bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr));
listen(sockfd, 5);
```

**This is all a software developer has to do (and test of course).**