Transitioning to IPv6 as a Preferred Network Protocol

Jason Healy – jhealy@suffieldacademy.org

Connecticut Education Network Member Presentation

October 22, 2024

Summary

What this talk is not:

- Debating the merits of IPv6
- Extensive introduction to IPv6

Why IPv6-Mostly[1]?

- The reality of IPv4 and IPv6 has changed
- Dual-stack wasn't supposed to be forever
- IPv6 is becoming a necessity
- Maintaining multiple stacks is a pain
- NAT is NAT

This process takes time... start now!

[1] https://www.ietf.org/archive/id/draft-ietf-v6ops-6mops-00.html

Checklist (1/2)

Pre-IPv6 Deployment

- ❑ Revise system and purchasing requirements to mandate IPv6 compatibility
- ❑ Obtain public IPv6 address allocation
- ❑ Plan IPv6 subnets

Dual-Stack Deployment

- ❑ Deploy IPv6 RA on test subnet, with internet reachability
- ❑ Convert servers to dual-stack (ready for v6-only clients)
- ❑ Adjust Security Policies and ACLs
- ❑ Publish IPv6 internal servers in DNS (alongside IPv4)
- ❑ Make IPv6 available to all subnets (alongside IPv4)

Checklist (2/2)

Transition Mechanisms

- ❑ Build NAT64, test internally
- ❑ Build DNS64, test as opt-in (alternate DNS server address)
- ❑ Switch to DNS64 as default for clients (dual-stack)

Deprecate IPv4

- ❑ Deploy IPv6 RA PREF64 (if supported on router)
- ❑ Deploy DHCPv4 Option 108 ("IPv6-only Preferred")

Optionally/Eventually:

- ❑ Turn off IPv4 completely for nodes!

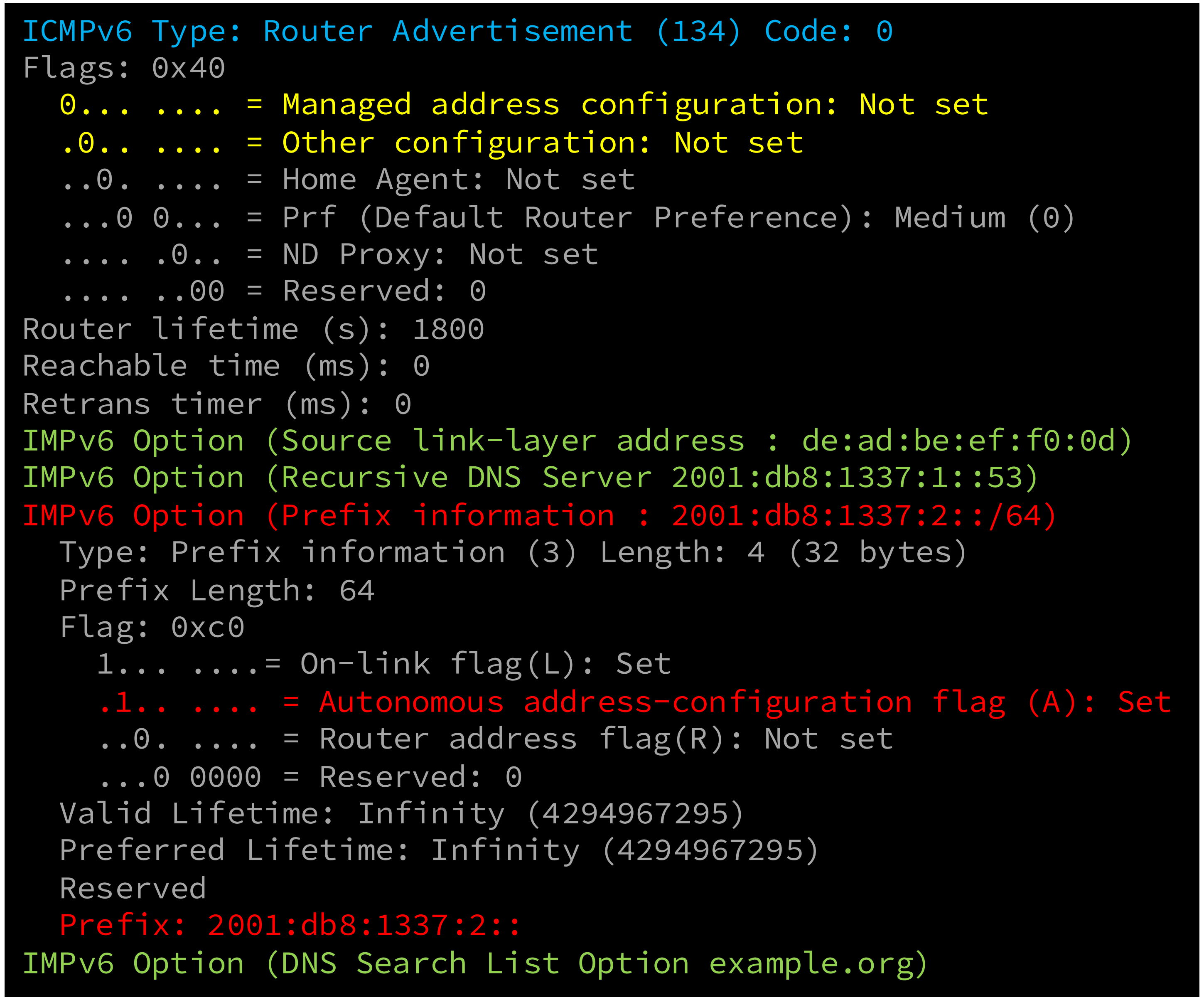
Pre-IPv6 Deployment

IPv6 Micro-review

- 128-bit addresses, represented as 8 groups of 4 hex digits:
  ```
  2001:0db8:1234:5678:0000:90ab:cdef
  ```
- Can "zero-compress" and remove leading zeros:
  ```
  2001:db8:1234:5678::90ab:cdef
  ```
- Overwhelming default of 64 bits for network, 64 bits for address:
  ```
  2001:db8:1234:5678::/64
  ```
- Multiple addresses per node are common
- Extensive use of multicast, Neighbor Discovery instead of ARP (but that's for another talk)

IPv6 Router Advertisements

- Periodic broadcasts, or response to Router Solicitations
- Contains options that help replace DHCP
- Contains address info to support SLAAC



```
ICMPv6 Type: Router Advertisement (134) Code: 0
Flags: 0x40
  0... .... = Managed address configuration: Not set
  .0.. .... = Other configuration: Not set
  ..0. .... = Home Agent: Not set
  ...0 0... = Prf (Default Router Preference): Medium (0)
  .... .0.. = ND Proxy: Not set
  .... ..00 = Reserved: 0
Router lifetime (s): 1800
Reachable time (ms): 0
Retrans timer (ms): 0
IMPv6 Option (Source link-layer address : de:ad:be:ef:f0:0d)
IMPv6 Option (Recursive DNS Server 2001:db8:1337:1::53)
IMPv6 Option (Prefix information : 2001:db8:1337:2::/64)
  Type: Prefix information (3) Length: 4 (32 bytes)
  Prefix Length: 64
  Flag: 0xc0
    1... ....= On-link flag(L): Set
    .1.. .... = Autonomous address-configuration flag (A): Set
    ..0. .... = Router address flag(R): Not set
    ...0 0000 = Reserved: 0
  Valid Lifetime: Infinity (4294967295)
  Preferred Lifetime: Infinity (4294967295)
  Reserved
  Prefix: 2001:db8:1337:2::
IMPv6 Option (DNS Search List Option example.org)
```

Specification Updates

Need to make sure you are holding vendors accountable!

Update any specifications you provide to vendors, and include language mandating IPv6-only.

Example: Suffield Academy's construction specifications (27-20-00)[1]:

IP version support. Suffield Academy is transitioning away from IPv4; the protocol is now deprecated and is not supported on new equipment. New equipment that uses IP to communicate must support operating using only IPv6. It must be possible to deploy, configure, and operate the equipment fully using only IPv6. Some critical requirements are provided below; vendors wishing to familiarize themselves with IPv6-only requirements are directed to the US OMB Memorandum M-21-07 as a starting point...

This doesn't magically make things work, but it helps with accountability.

[1] https://web.suffieldacademy.org/ils/netadmin/Suffield-Academy-Communications-Standards-universal.pdf

Obtain IPv6 Addresses

Ask CEN for allocation!

How does 1,208,925,819,614,629,174,706,176 addresses sound?

- You'll get a /48, so `2001:db8:NNNN`
- A /48 is large enough to carve up into 65535 standard-size /64 subnets (same amount as dividing 10/8 into individual /24 subnets)

```
2001:db8:1337:SSSS::H
        (48)  (16)  (64)
```

Allocated Prefix Subnet Host portion

```
10.SSS.SSS.H
(8) (8+8) (8)
```

- Handoff and routing are just like IPv4 (next-hop, default route, *etc.*)

Plan IPv6 Subnets

Now that you have all this IPv6 space, it's time to divide it up

- Recommend nybble[1] (4-bit) boundary alignment for administrative groupings[2]
- Lots of ways to do this (by site, building, function)
- We are a single campus, and assigned one subnet per VLAN ID:

```
2001:db8:1337:VVV::
```

Examples:

```
VLAN  2 – 2001:db8:1337:2::
VLAN 37 – 2001:db8:1337:25::
```

- Give it some thought before you start; planning makes rollout easier

[1] AKA nibble, hexit, semi-octet, half-byte, tetrade, quadbit, quartet
[2] https://www.ripe.net/media/documents/BasicIPv6-Appendix-AddressingPlanHowTo.pdf

Dual-Stack Deployment

Testing Dual-Stack: Routers

- Identify a test segment: VLAN 2, IPv6 subnet `2001:db8:1337:2::/64`
- Assign router address in this prefix and enable router advertisements with recursive DNS server specified

```
vlan 2                                ! (Cisco)

interface Vlan2
no ip address
 ipv6 address 2001:DB8:1337:2::1/64
 ipv6 enable
 ipv6 nd ra dns server 2001:DB8:1337::53 200
!
```

```
set vlans test vlan-id 2              # (Juniper)
set vlans test l3-interface irb.2

set interfaces irb unit 2 family inet6 address 2001:db8:1337:2::1/64

set protocols router-advertisement interface irb.2 prefix 2001:db8:1337:2::/64
set protocols router-advertisement interface irb.2 dns-server-address \
                                                2001:db8:1337::53 lifetime 86400
```

Testing Dual-Stack: Nodes

Nodes should now autoconfigure an IPv6 address in your prefix:

```
en0: flags=88e3<UP,BROADCAST,SMART,RUNNING,NOARP,SIMPLEX,MULTICAST> mtu 1500
      options=6460<TSO4,TSO6,CHANNEL_IO,PARTIAL_CSUM,ZEROINVERT_CSUM>
      ether fa:ca:de:de:ad:00
      inet6 fe80::f2ca:deff:fede:ad00%en0 prefixlen 64 secured scopeid 0xf
      inet6 2001:db8:1337:2:f2ca:deff:fede:ad00 prefixlen 64 autoconf secured
      inet6 2001:db8:1337:2:cafe:babe:8bad:f00d prefixlen 64 autoconf temporary
      ...
      media: autoselect
      status: active
```

Your DNS servers should also show up, possibly merged with IPv4.

Try it out!

```
% ping6 2600::1
PING6(56=40+8+8 bytes) --> 2600::1
16 bytes from 2600::1, icmp_seq=0 hlim=54 time=88.186 ms
...
```

Convert Servers to Dual-Stack

You want your new IPv6 network to be able to talk to something!

```
# /etc/network/interfaces

allow-hotplug eth0

# auto-configure IPv6
iface eth0 inet6 auto

# can also have static v6
iface eth0 inet6 static
        address 2001:db8:1337:2::42:0/64

# legacy v4
iface eth0 inet dhcp
```

May need to add "listen" directives to daemons to activate IPv6

Update Security Policies and ACLs

- Some nodes will prefer IPv6 if it is available (Happy Eyeballs)
- If "allowlist" ACLs only have IPv4 addresses listed, this may block or delay traffic
- Duplicate your security policies in IPv6
- Test with a scratch box that has IPv4 turned off
- Add IPv6 addresses to service monitoring

Publish IPv6 DNS Records

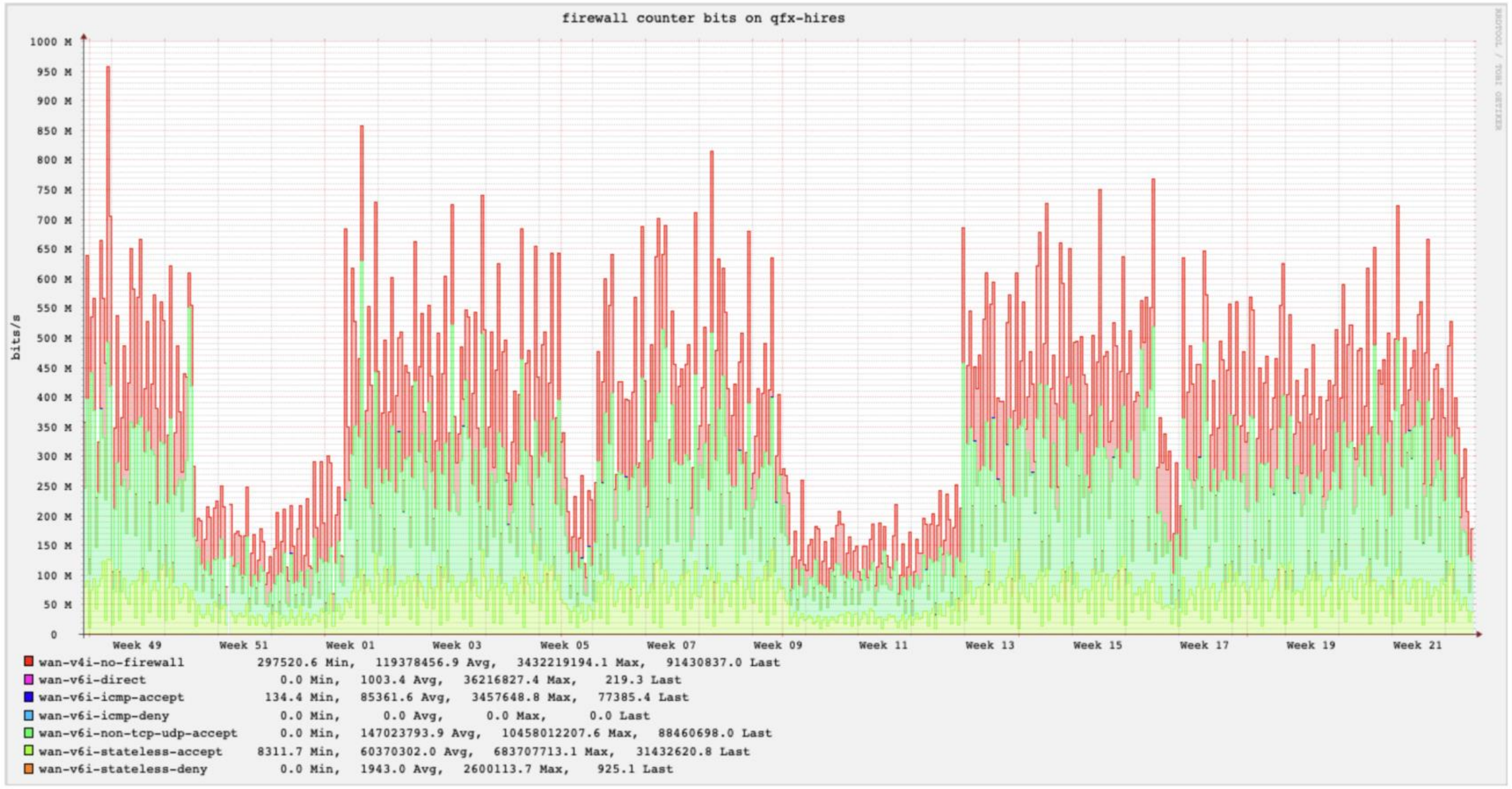
- Create "AAAA" records for internal servers
- Can coexist with IPv4 ("A" records), CNAME, MX, etc
- Test continuously (again, with IPv4 off if possible)

Deploy IPv6 Dual-Stack On All Subnets

1. Keep IPv4 on and available
2. Add IPv6 subnets to all VLANs
3. ???
4. PROFIT!!!

Suffield Academy after dual-stack deployment

Red is IPv4
Greens are IPv6 (50+%)



```
🟥 wan-v4i-no-firewall          297520.6 Min,  119378456.9 Avg,  3432219194.1 Max,  91430837.0 Last
🟪 wan-v6i-direct                   0.0 Min,       1003.4 Avg,    36216827.4 Max,      219.3 Last
🟦 wan-v6i-icmp-accept            134.4 Min,      85361.6 Avg,     3457648.8 Max,    77385.4 Last
🟦 wan-v6i-icmp-deny                0.0 Min,          0.0 Avg,           0.0 Max,        0.0 Last
🟩 wan-v6i-non-tcp-udp-accept       0.0 Min,  147023793.9 Avg, 10458012207.6 Max,  88460698.0 Last
🟩 wan-v6i-stateless-accept      8311.7 Min,   60370302.0 Avg,   683707713.1 Max,  31432620.8 Last
🟧 wan-v6i-stateless-deny           0.0 Min,       1943.0 Avg,     2600113.7 Max,      925.1 Last
```

Transition Mechanisms

Why Transition?

Under dual-stack, things seem pretty good, but…

- Does not address IPv4 shortages
- Now have two network stacks to debug
- Address selection rules can be difficult to understand
- Happy Eyeballs can hide disfunction
- Goal is to reduce or eliminate IPv4 on the nodes

Intro to NAT64

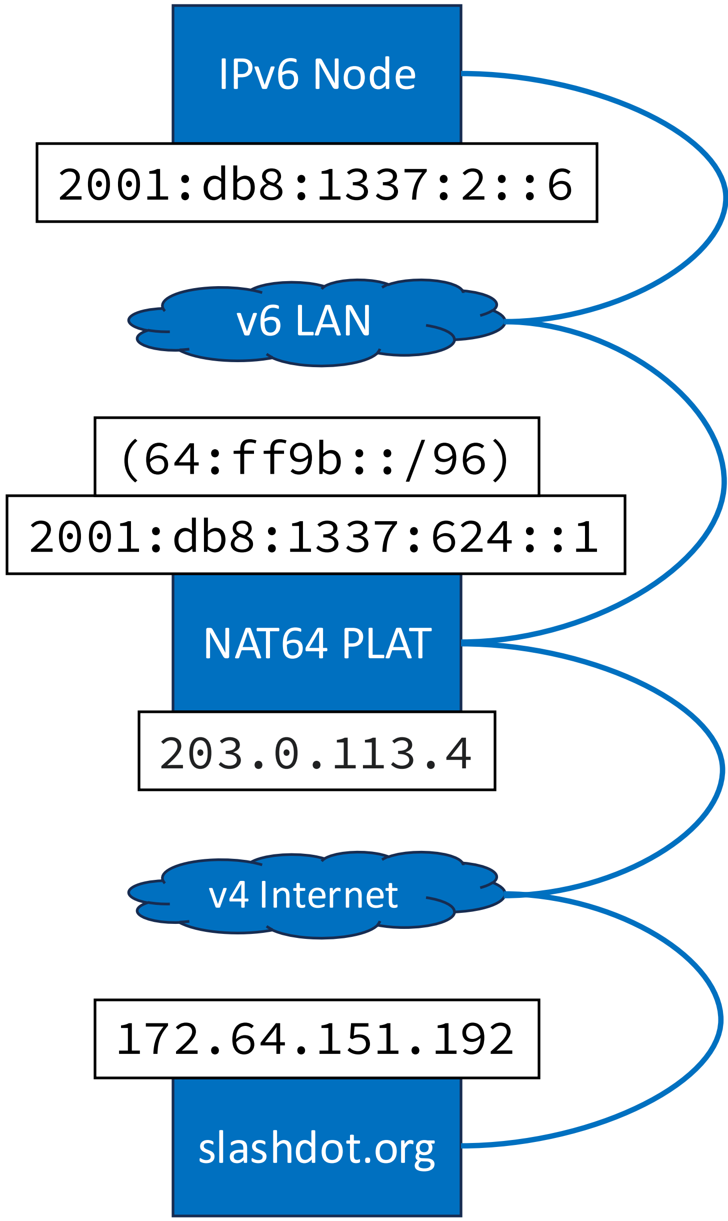
- Mechanism to allow single-stack IPv6 nodes to reach IPv4 resources
- When node wants to reach an IPv4 host, it "embeds" the IPv4 address in an IPv6 address
- Requires a router at the network edge to "unembed" IPv4 and forward to the IPv4 resource
- Thus, IPv4 addresses are still needed, but only on this NAT64 router, and not on the host
- Several flavors of NAT64 exist, we will primarily discuss "Stateful NAT64"[1]

[1] https://www.rfc-editor.org/rfc/rfc6146

NAT64 Example (1/5)

Our setup:

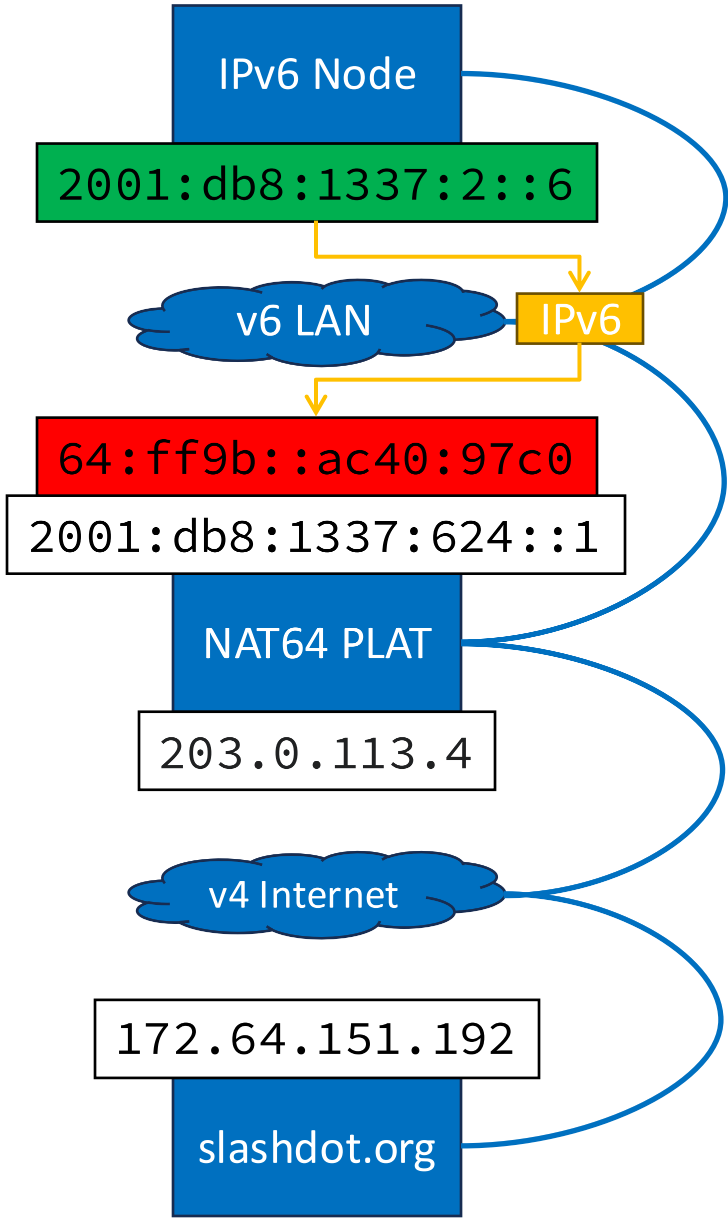
- IPv6-only node: `2001:db8:1337:2::6`
- IPv4-only server: `172.64.151.192` (slashdot.org)
- PLAT box that changes the address family
- NAT64 shared public IPv4 address: `203.0.113.4`
- Dedicated NAT64 prefix: `64:ff9b::/96`
  (AKA the "well-known prefix")
- L3 route on LAN directing NAT64 prefix to PLAT:
  `64:ff9b::/96 next-hop 2001:db8:1337:624::1`



NAT64 Example (2/5)

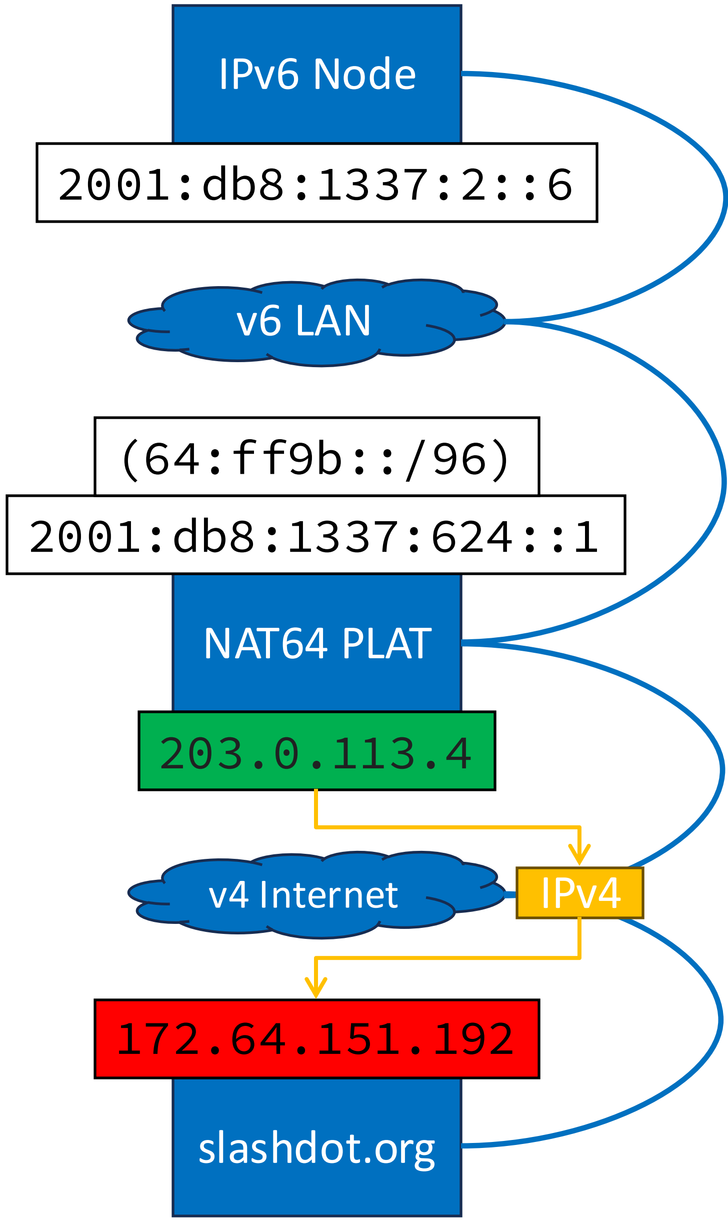
- Node queries DNS for `slashdot.org`
- Receives A record answer of `172.64.151.192`
- Converts to hex: `ac.40.97.c0`
- Adds NAT64 prefix: `64:ff9b::ac40:97c0`
- Sends packet via IPv6

We will talk about how this happens in a minute!



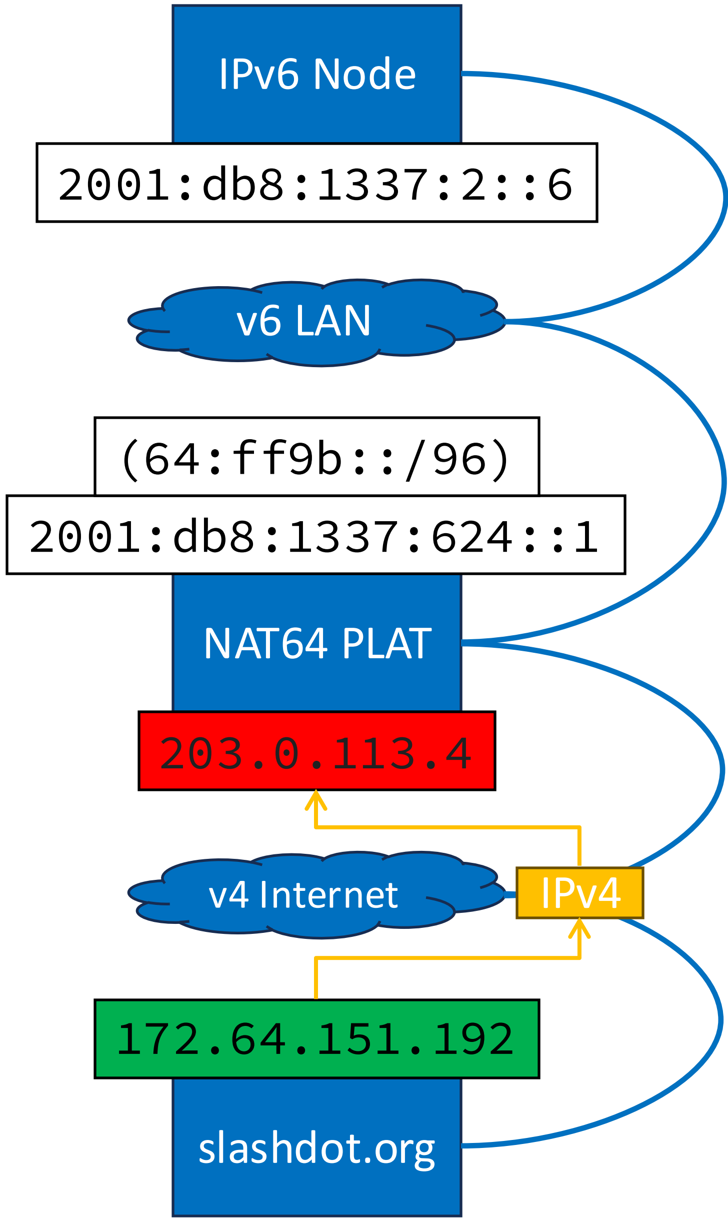
NAT64 Example (3/5)

- PLAT receives IPv6 packet
- Extracts IPv4 portion of address: `ac.40.97.c0`
- Translates to IPv4: `172.64.151.192`
- Replaces packet source with public IPv4 address: `203.0.113.4`
- Stores NAT state of IPv6 source/port, IPv4 source/port, IPv4 dest/port
- Sends packet via IPv4



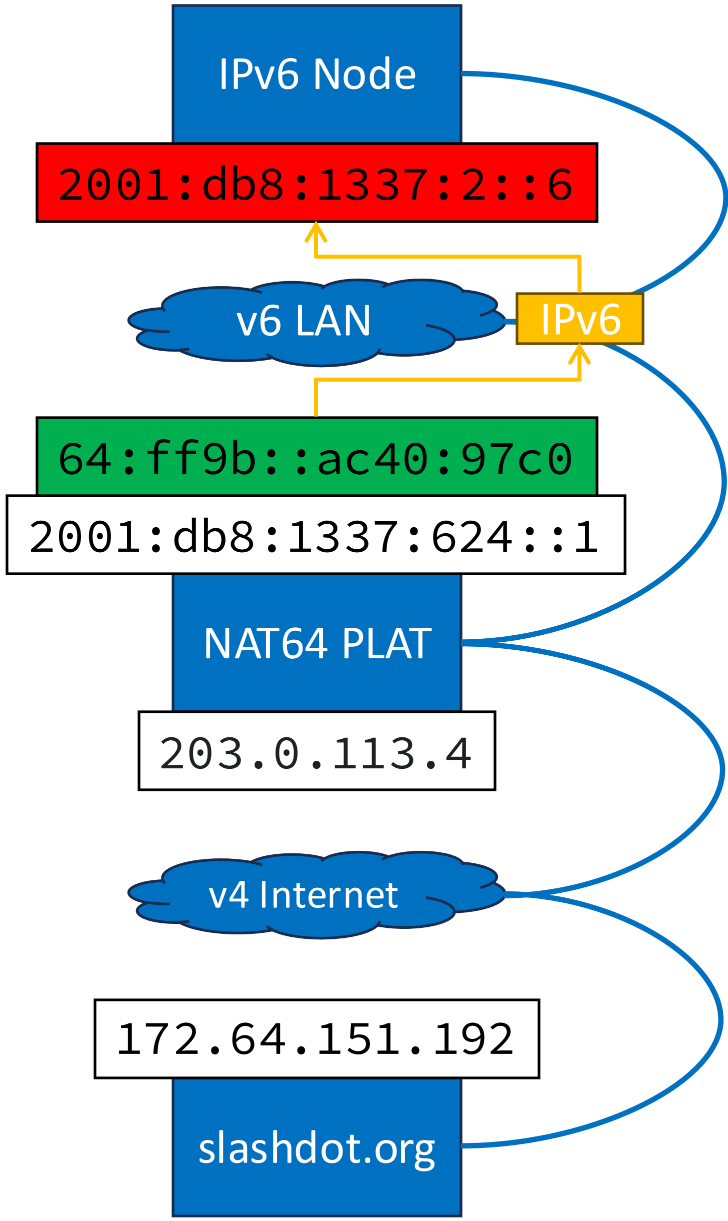
NAT64 Example (4/5)

- Slashdot receives packet
- Slashdot generates reply to PLAT public IPv4
- PLAT looks up state using packet source: `172.64.151.192`
- PLAT finds `172.64.151.192` associated with state to IPv6 address `2001:db8:1337:2::6`



NAT64 Example (5/5)

- PLAT translates address family, embeds source address in the NAT64 prefix, sends IPv6 packet back to our node
- Node sees that response matches the original source address it sent packet to
- Node is not aware that packet translation took place; traffic looks like normal IPv6



NAT64 Support

Free and Open Source:

- Jool (Linux)
  https://nicmx.github.io/Jool/en/index.html
- PF (OpenBSD)
  https://man.openbsd.org/pf.conf.5#af-to
- TAYGA (Linux)
  http://www.litech.org/tayga/
- FD.io VPP (Linux)
  https://s3-docs.fd.io/vpp/24.10/developer/plugins/nat64.html

Also, commercial vendors (A10, PaloAlto, Fortigate, Cisco, etc)

Can test some for free:
https://go6lab.si/current-ipv6-tests/nat64dns64-public-test/

NAT64 Summary

- Allows IPv6-only nodes to communicate with IPv4-only resources
- Only works with "big three": TCP, UDP, ICMP
- Has similar address sharing properties to NAT44
- Stateless varieties exist, typically need 1:1 v4:v6 address mapping

Back to our earlier question:
How do nodes know to use mapped addresses???

DNS64

- Created to work with NAT64 PLAT
- Tricks hosts into using mapped addresses for IPv4-only resources
- No special client support needed
- Easy to deploy (point nodes to a DNS64 server)

```
# unbound DNS64 proxy (uses upstream server for recursion)
server:
        dns64-prefix: 64:ff9b::0/96
        module-config: "dns64 validator iterator"

forward-zone:
        # forward all queries
        name: "."
        forward-addr: 2001:db8:1337::53
        # don't cache (we'll let upstream do that)
        forward-no-cache: yes
        # don't try to recurse on our own if the upstream fails
        forward-first: no
```

DNS64 The Easy Way

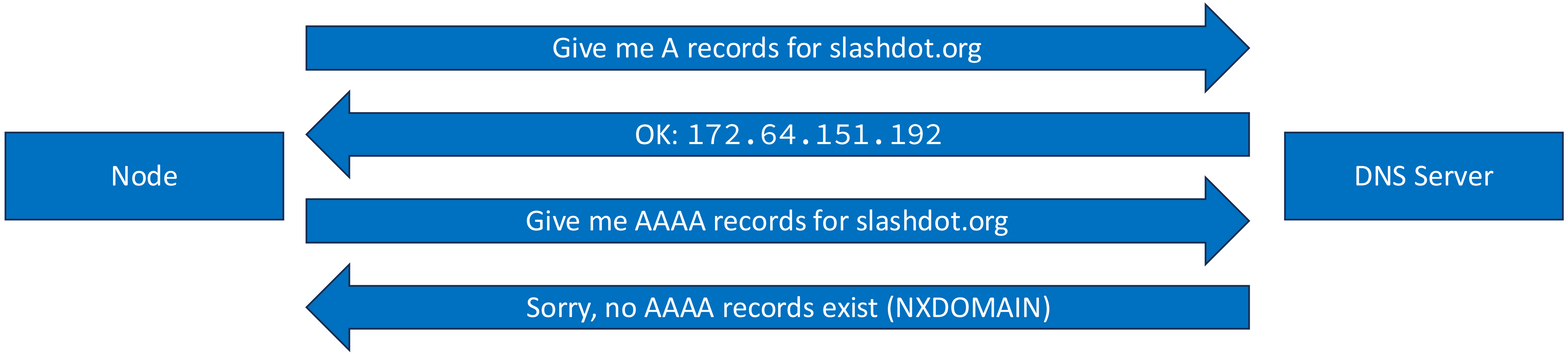
Many public DNS providers also offer DNS64 with the WKP:

- nat64.net[1]
  ```
  2a01:4ff:f0:9876::1
  2a00:1098:2c::1
  2a01:4f8:c2c:123f::1
  ```
- Google DNS[2]
  ```
  2001:4860:4860::6464
  2001:4860:4860::64
  ```
- Cloudflare DNS[3]
  ```
  2606:4700:4700::64
  2606:4700:4700::6400
  ```

[1] https://nat64.net
[2] https://developers.google.com/speed/public-dns/docs/dns64
[3] https://developers.cloudflare.com/1.1.1.1/infrastructure/ipv6-networks/

DNS64 In Action

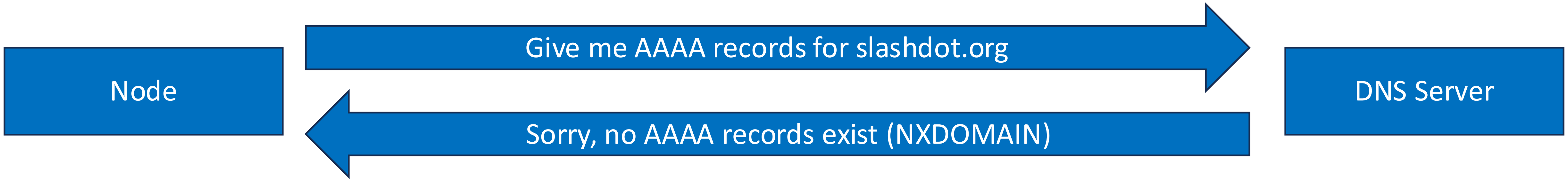
Dual-stack node, normal DNS server:


Node
Give me A records for slashdot.org
OK: 172.64.151.192
Give me AAAA records for slashdot.org
Sorry, no AAAA records exist (NXDOMAIN)
DNS Server

Node only receives IPv4 addresses, so communicates over IPv4

DNS64 In Action

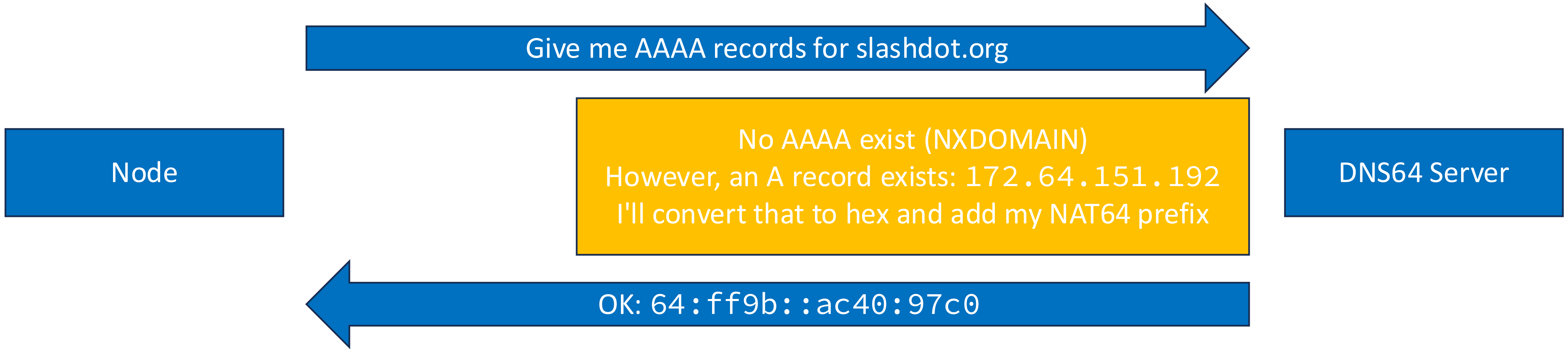
IPv6-only node, normal DNS server:


Node
Give me AAAA records for slashdot.org
Sorry, no AAAA records exist (NXDOMAIN)
DNS Server

No IPv6 addresses, so unable to communicate

DNS64 In Action

IPv6-only node, DNS64 server:


Give me AAAA records for slashdot.org
Node
No AAAA exist (NXDOMAIN)
However, an A record exists: 172.64.151.192
I'll convert that to hex and add my NAT64 prefix
DNS64 Server
OK: 64:ff9b::ac40:97c0

Node receives IPv6 answer, proceeds normally. Packets are routed to NAT64 PLAT which handles translation to IPv4 for delivery.

DNS64 Drawbacks

- Breaks DNSSEC
- Breaks if clients choose their own DNS server
- Breaks with legacy software that opens AF_INET only
- Breaks when IPv4 literals are used instead of hostnames (protocols with embedded addresses or lazy programmers)

```
# ping6 -c 3 64:ff9b::808:808
PING 64:ff9b::808:808(64:ff9b::808:808) 56 data bytes
64 bytes from 64:ff9b::808:808: icmp_seq=1 ttl=115 time=6.02 ms
64 bytes from 64:ff9b::808:808: icmp_seq=2 ttl=115 time=6.15 ms
64 bytes from 64:ff9b::808:808: icmp_seq=3 ttl=115 time=6.15 ms
--- 64:ff9b::808:808 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 6.018/6.107/6.152/0.063 ms

# ping -c 3 8.8.8.8
ping: connect: Network is unreachable
```

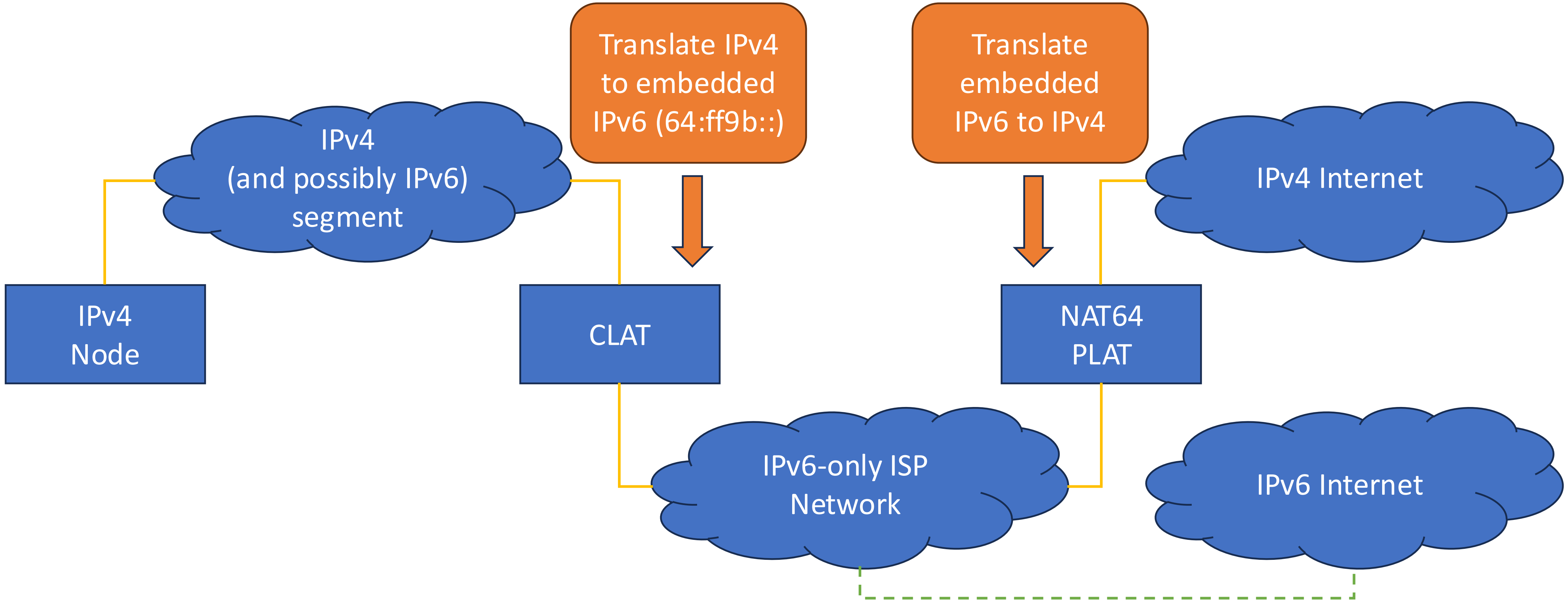
CLAT / 464XLAT

- Client-side transLATor is like the "opposite" of NAT64 (NAT46?)[1]
- Still requires the use of NAT64 / PLAT (builds on it)
- Double translation: 4 to 6 (CLAT), then 6 back to 4 (PLAT) (hence the name "464", 4 to 6 to 4)
- Node appears to be dual-stack to software on the host (even though it's IPv6-only); IPv4 software thinks it's regular IPv4 with NAT
- Used extensively in mobile networks (e.g., T-Mobile)
- Been working for over a decade!

[1] https://www.rfc-editor.org/rfc/rfc6877

CLAT (Discrete)

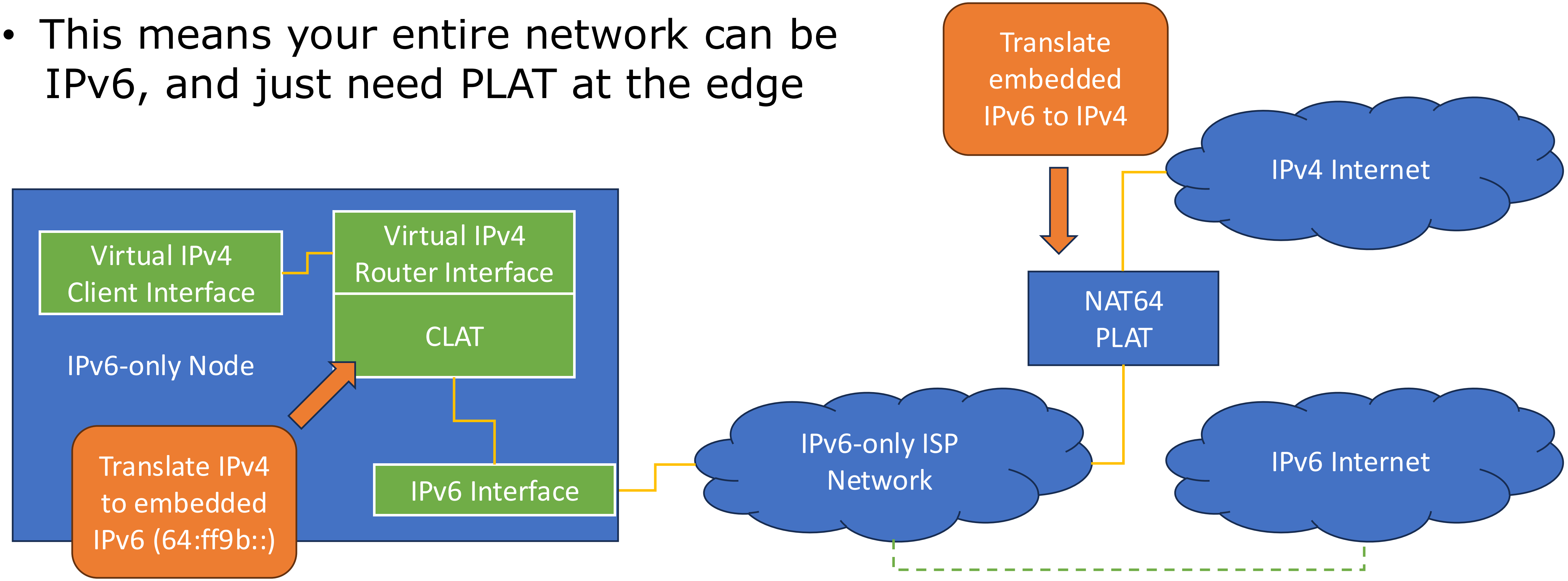
- CLAT is a router (e.g., home router, cable modem)
- CLAT translates "inside" IPv4 to IPv6 for transport across ISP network
- PLAT translates back to IPv4 at the network edge


Translate IPv4 to embedded IPv6 (64:ff9b::)
Translate embedded IPv6 to IPv4
IPv4 (and possibly IPv6) segment
IPv4 Internet
IPv4 Node
CLAT
NAT64 PLAT
IPv6-only ISP Network
IPv6 Internet

CLAT (Embedded)

- CLAT is a virtual router in the OS of the device
- All IPv4 packets are embedded in IPv6 before leaving the device
- PLAT still translates back to IPv4 at the network edge

- This means your entire network can be IPv6, and just need PLAT at the edge


Translate embedded IPv6 to IPv4
IPv4 Internet
Virtual IPv4 Client Interface
Virtual IPv4 Router Interface
CLAT
IPv6-only Node
NAT64 PLAT
Translate IPv4 to embedded IPv6 (64:ff9b::)
IPv6 Interface
IPv6-only ISP Network
IPv6 Internet

CLAT Pros and Cons

Pros:

- All IPv4 traffic is translated (not just addresses from DNS)
- Physical equipment not required (good for enterprise / BYOD)
- DNSSEC works (not messing with DNS)

Cons:

- CLAT functionality must exist on each device (more in a minute)
- CLAT must be configured with NAT64 prefix

CLAT Configuration

How do we configure and enable CLAT?

RFC 7050: PREF64 discovery via DNS

- Only requires working DNS64
- Not well-defined when to use CLAT or abandon IPv4
- Not granular (can't decide per-subnet)

RFC 8781: PREF64 discovery via IPv6 RA

- Doesn't require discovery
- More granular (per-subnet RA)
- Router support has been gradual, but rolling out now (check docs)

RFC 8925: IPv6-only preferred (DHCPv4 option 108)

- Tells node to not request IPv4 if supported (conserves addresses)
- More granular
- Works in conjunction with RFC750/RFC8781 to enable CLAT

CLAT Client Support (Android)

Supported since Android 4.3 in 2013 (thanks, T-Mobile!)

Android 12+ supports RFC 8925 (DHCP 108)[1]

Supports RFC 8781 (PREF64 RA)[2]

[1] https://indico.cern.ch/event/1274792/contributions/5444353/attachments/2676554/4642104/IPv6%20mostly.pdf
[2] https://openwrt.org/docs/guide-user/network/ipv6/nat64

CLAT Client Support (Apple)

- Supported since macOS 13.1, iOS 16 (and iPadOS/tvOS)

```
% ifconfig en0
en0: flags=88e3<UP,BROADCAST,SMART,RUNNING,NOARP,SIMPLEX,MULTICAST> mtu 1500
	options=6460<TSO4,TSO6,CHANNEL_IO,PARTIAL_CSUM,ZEROINVERT_CSUM>
	ether fa:ca:de:de:ad:00
	inet6 fe80::f2ca:deff:fede:ad00%en0 prefixlen 64 secured scopeid 0xf
	inet6 2001:db8:1337:2:f2ca:deff:fede:ad00 prefixlen 64 autoconf secured
	inet6 2001:db8:1337:2:cafe:babe:8bad:f00d prefixlen 64 autoconf temporary
	inet 192.0.0.2 netmask 0xffffffff broadcast 192.0.0.2
	inet6 2001:db8:1337:2:7ac7:1e55:dead:d00d prefixlen 64 clat46
	nat64 prefix 64:ff9b:: prefixlen 96
	nd6 options=201<PERFORMNUD,DAD>
	media: autoselect
	status: active
```

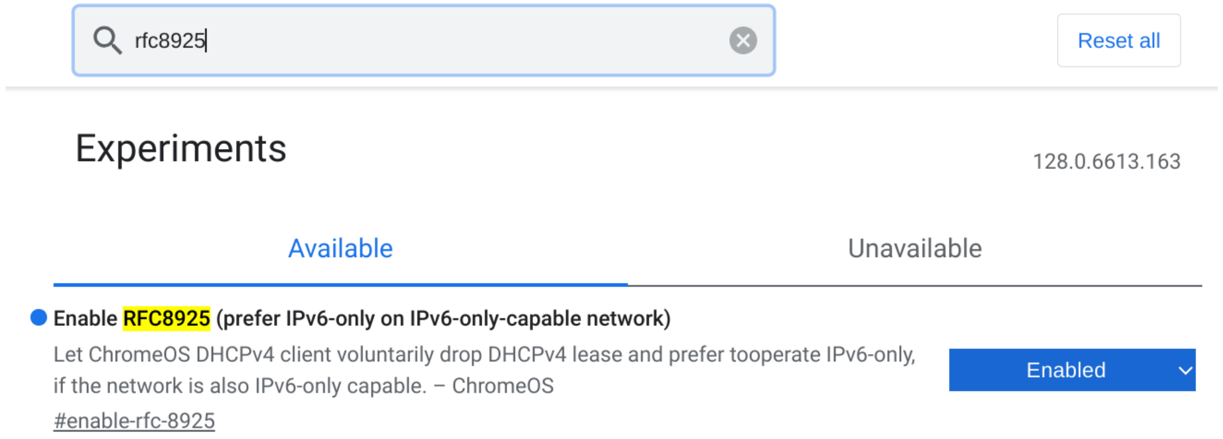
```
% netstat -f inet -rn
Destination        Gateway            Flags               Netif Expire
default            192.0.0.1          UGScg                 en0
...
```

- Need DNS64 as well?

CLAT Client Support (ChromeOS)

Since v114

- activated via DHCP option 108
- chrome://flags
- enable RFC8925
- reboot



rfc8925

Reset all

Experiments

128.0.6613.163

Available | Unavailable

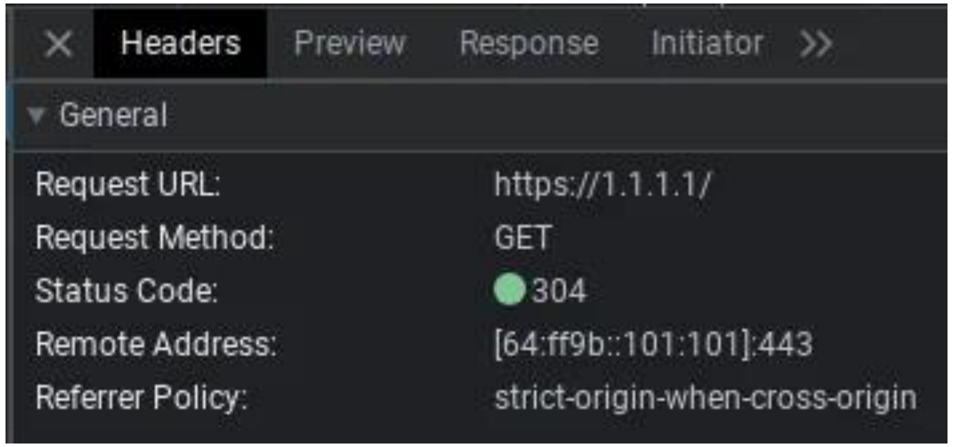
● Enable RFC8925 (prefer IPv6-only on IPv6-only-capable network)

Let ChromeOS DHCPv4 client voluntarily drop DHCPv4 lease and prefer tooperate IPv6-only, if the network is also IPv6-only capable. – ChromeOS

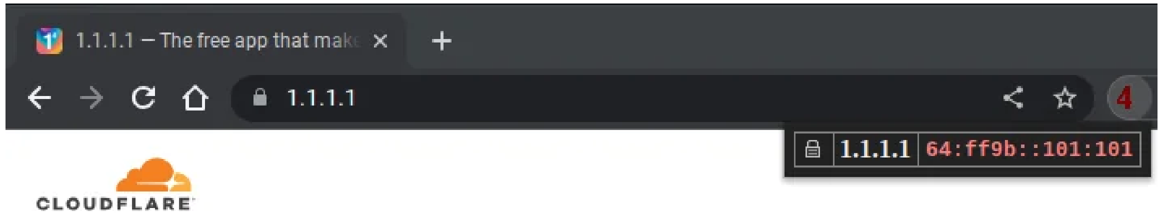
#enable-rfc-8925

Enabled

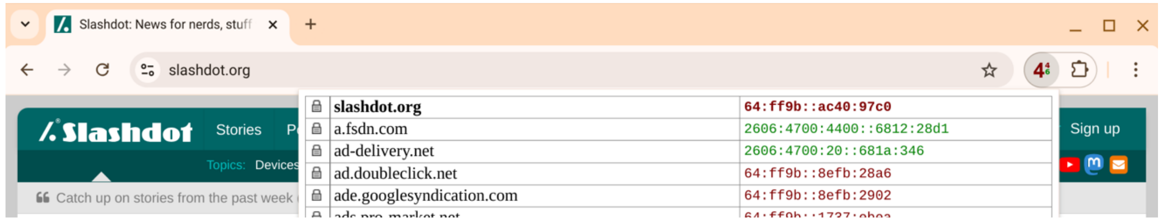
No visible IPv4 CLAT[1], but IPv4-only and IPv4 literals work:



Request URL:	https://1.1.1.1/
Request Method:	GET
Status Code:	● 304
Remote Address:	[64:ff9b::101:101]:443
Referrer Policy:	strict-origin-when-cross-origin



1.1.1.1	64:ff9b::101:101


Slashdot: News for nerds, stuff
slashdot.org
Slashdot
Stories
Topics: Devices
Catch up on stories from the past week
Sign up

slashdot.org	64:ff9b::ac40:97c0
a.fsdn.com	2606:4700:4400::6812:28d1
ad-delivery.net	2606:4700:20::681a:346
ad.doubleclick.net	64:ff9b::8efb:28a6
ade.googlesyndication.com	64:ff9b::8efb:2902

[1] https://www.reddit.com/r/ipv6/comments/1abpp71/chromeos_supports_rfc_8925_in_a_nondefault/

CLAT Client Support (Linux)

Userspace daemon only[1], no automatic behavior

We haven't found this necessary for servers:

- Recent software is IPv6-capable
- DNS64 handles most cases for outbound
- NAT64 SIIT handles inbound translation

[1] https://github.com/toreanderson/clatd

CLAT Client Support (Windows)

Has "supported" since Windows 10 (thanks, T-Mobile!)…

… but only when connected to an LTE device

Official plans for support in Windows 11 [1] …

… but nothing as of October 2024 (Windows 11 24H2) [2]

Like Linux, DNS64 and pure-IPv6 is still an option, but the possibility of issues is higher

[1] https://techcommunity.microsoft.com/t5/networking-blog/windows-11-plans-to-expand-clat-support/ba-p/4078173
[2] https://mailarchive.ietf.org/arch/msg/v6ops/mPSMFY-StOgYMaNpAK-yflLyP8Q/

Deprecate IPv4

Procedure

Once dual-stack and DNS64 are operational, can start to deprecate IPv4

- Start by manually testing a node with IPv4 turned off
  - Worst-case scenario
  - May activate CLAT (macos/ios)
- Enable PREF64 RA (if available)
- Enable DHCP Option 108

The Secret Switch

Summer 2024, enabled DHCP 108 for guest and student subnets (without telling anyone)

After return to school:

< 10 help desk tickets regarding network issues (all but one were related to VPN malfunction)

Standardized testing (SAT), only one issue (guest had IPv6 disabled on their machine)

Major guest events (parent weekend, alumni weekend), no reported issues

DHCP server broke one day and we didn't notice for 12+ hours… 😬

Results

Real-world measurements from our network, October 2024

- All-Apple campus (macos/ios), so YMMV
- BYOD allowed (a few Windows gaming PCs)
- DHCP 108 rolled out to students and guests, not for employees (yet)
- Counts show unique MAC with at least one address for that family (MAC is only counted once even if multiple addresses assigned)

Sample	OU	IPv4	IPv6
A	Employees	228	224
A	Students	36	**746**
A	Guests	39	169
B	Employees	242	238
B	Students	37	**772**
B	Guests	40	161

Next Steps

- Roll out DHCP 108 to all subnets (relatively easy)

Then, the "long tail" of IPv4…

- Begin restricting IPv4 DHCP to "known" clients
- Possibly pool non-IPv6-capable hosts in their own subnets
- Consider implementing intermediate CLAT for IPv4-only nodes so that core infrastructure can be converted to IPv6-only
- Shut off IPv4 and DHCP completely once subnets appear fully IPv6

Gotchas

Gotchas (General IPv6)

Unfortunately, not all 🌈 + 🦄

- ACL debugging
- Service configuration
- Legacy software
- Devices that suck (Nintendo switch, building management, IoT garbage)
- Loss of DHCPv4 "tracking" vs SLAAC (consider ND table scraping)
- Wifi / Security software not supporting multiple IPv6 addresses
  (see Linkova for details)

Gotchas (DNS64)

- Split-horizon (especially with RFC1918 addresses)
- Breaking DNSSEC
- Dealing with rogue client configuration
- Happy Eyeballs (may fall back to IPv4 if not explicitly IPv6-only)

Gotchas (NAT64 and CLAT)

- RFC1918 addresses and NAT64
- Non-TCP/UDP/ICMP protocols (VPNs)
- Embedded-IPv4 protocols (SIP)
- MTU woes (IPv4 to IPv6 reduces path MTU)
- Lack of device support (Windows, game consoles, IoT)
- Lack of testing (YouTube AppleTV app)

Worth It?

Worth It

- NAT64 + CLAT looks just like NAT44 to software
- Client support is getting quite good (c'mon Windows!)
- Time is right to shift to IPv6 as primary protocol and deprecate IPv4
- Get your growing pains out of the way now

Additional Resources

Jen Linkova at Google, *Mission ~~Im~~Possible: Turning IPv4 Off in an Enterprise Network*:

- https://ripe87.ripe.net/wp-content/uploads/presentations/32-IPv6-Mostly-Office_-JenLinkova_RIPE87.pdf
- https://www.youtube.com/watch?v=hb98hAb5_W8 and https://ripe87.ripe.net/archives/video/1160/

Ondřej Caletka at RIPE:

- https://labs.ripe.net/author/ondrej_caletka_1/deploying-ipv6-mostly-access-networks/
- 2022 talk: https://www.ripe.net/media/documents/NLNOG_Ondrej.pdf

Jool "Introduction to IPv4/IPv6 Translation", has overview of multiple NAT64 modes with diagrams:

- https://nicmx.github.io/Jool/en/intro-xlat.html

RFC Draft *IPv6-Mostly Networks: Deployment and Operations Considerations*

- https://www.ietf.org/archive/id/draft-ietf-v6ops-6mops-00.html

RFC Draft *IPv6 Address Accountability Considerations* (discusses how to track IPv6 without using DHCP)

- https://datatracker.ietf.org/doc/html/draft-ccc-v6ops-address-accountability-00

RFC 8683 Additional Deployment Guidelines for NAT64/464XLAT in Operator and Enterprise Networks:

- https://www.rfc-editor.org/rfc/rfc8683

My "blog", ***very*** occasionally updated: https://web.suffieldacademy.org/~jhealy/