Happy Eyeballs, Version 3: *Better Connectivity Using Concurrency*

Tommy Pauly, David Schinazi, Kenichi Ishibashi, Nidhi Jaju
ALLDISPATCH - IETF 119 Brisbane - March 2024

Make the users' eyeballs "happy" by making connections **quickly** to servers that **work**

Happy Eyeballs started as an algorithm
to ease the transition to IPv6

This assumes that IPv6 is more desirable
for clients to use, partly based on performance

Fewer NATs

More optimized servers

More server locations

Better routing

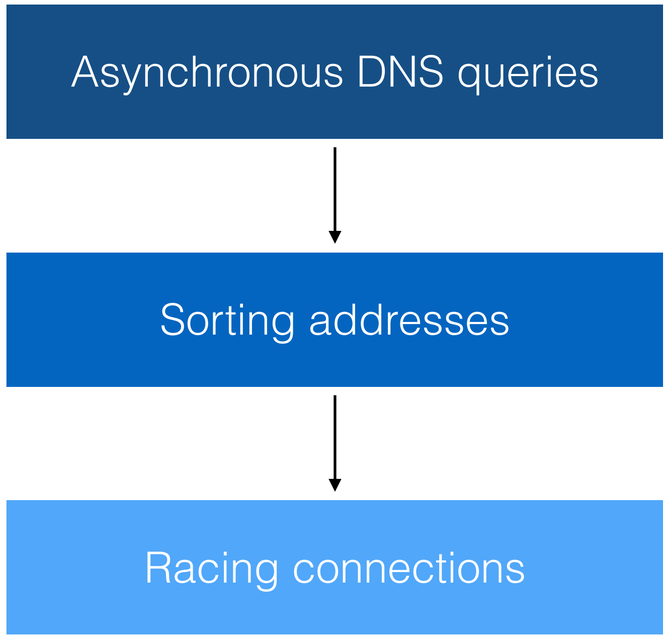
Why not always use IPv6 when it's available?

Servers might be broken or slow on IPv6

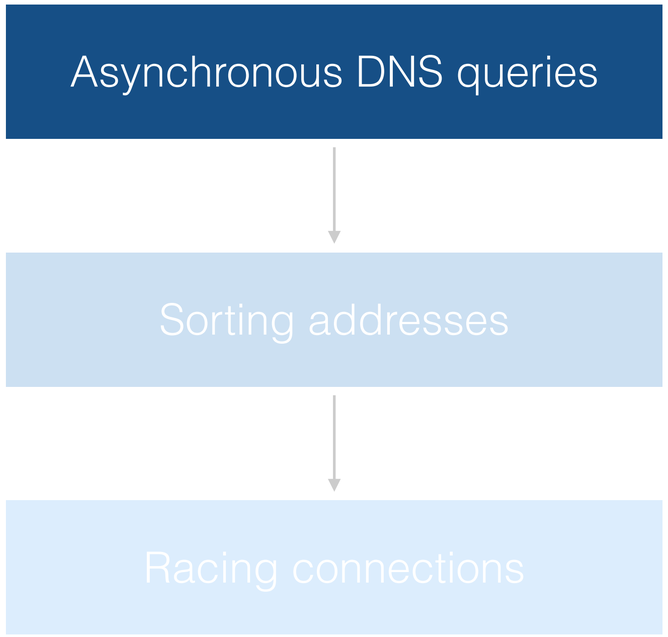
Networks might be broken or slow on IPv6

Waiting on a single address is a **bad idea**

Algorithm (RFC 8305)



Algorithm (RFC 8305)



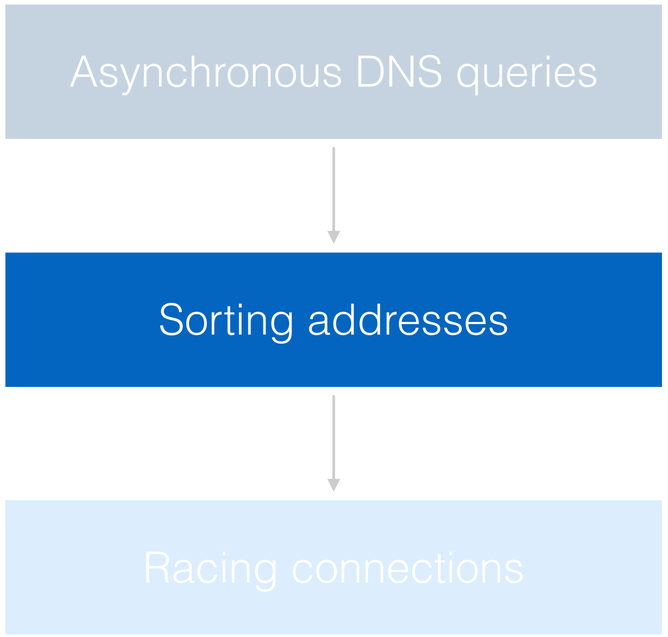
Prefer IPv6 DNS resolver addresses

Issue AAAA / A queries in parallel (AAAA first)

Act on AAAA responses immediately

Wait up to 50ms for AAAA if A comes back first

Algorithm (RFC 8305)



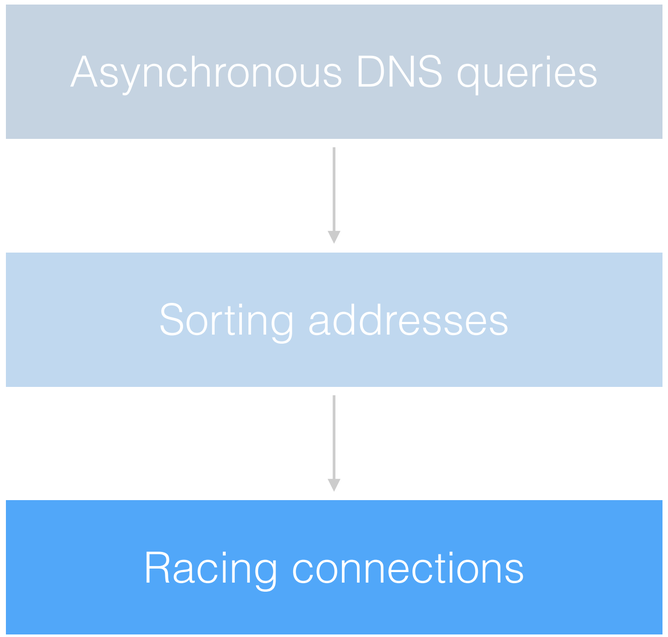
RFC 6724 address sorting

Historical round trip times

Place IPv6 at the start; two IPv6 addresses first if available

Use IPv4 first if historical data shows IPv6 brokenness

Algorithm (RFC 8305)

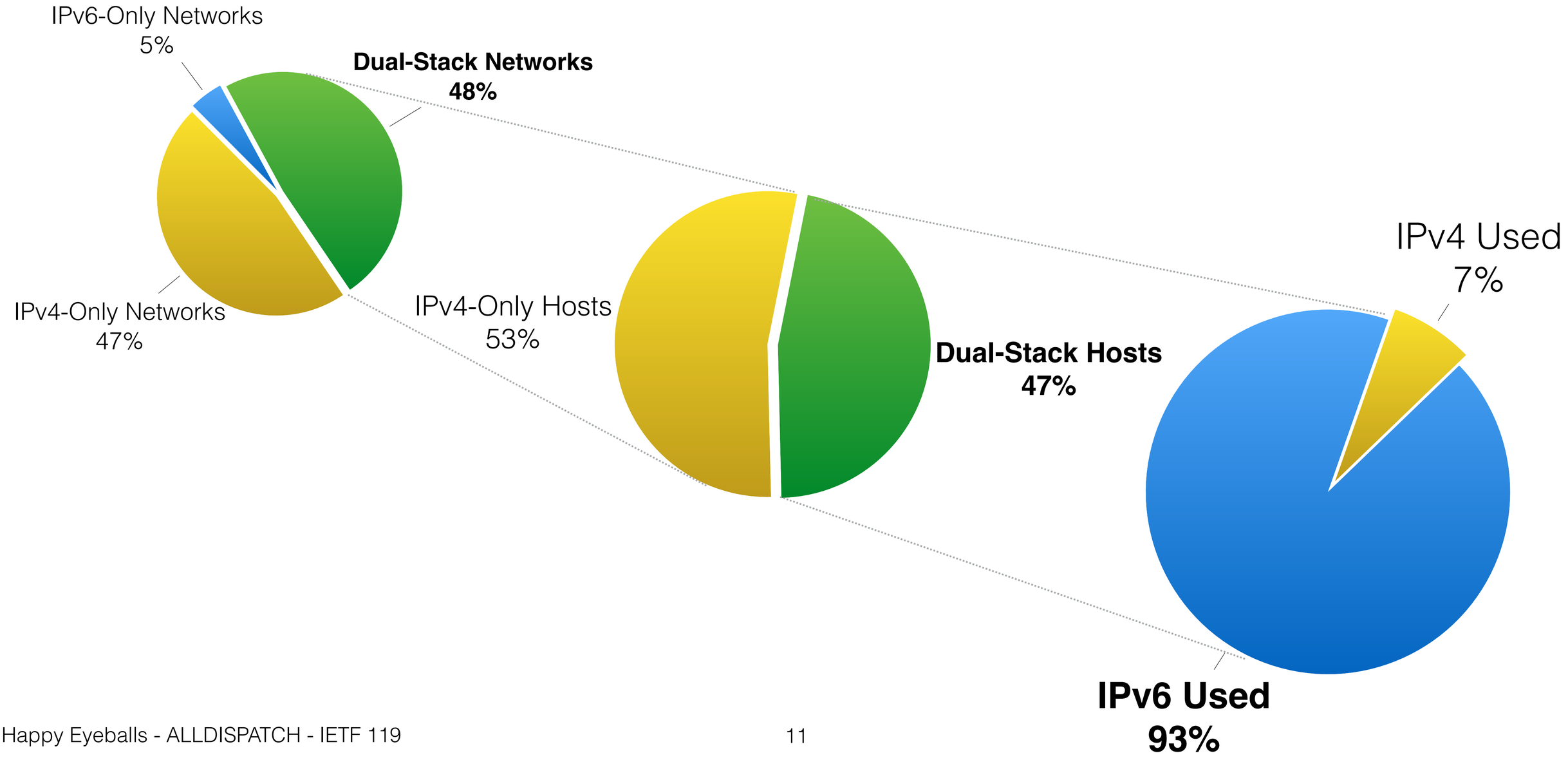


Start attempts to addresses in parallel, staggered by a delay

Delay based on the TCP retransmission timeout

Race ends when one connection completes

Worldwide per-connection Happy Eyeballs statistics



Sidebar: Reporting IPv6 brokenness

Discussions in v6ops have raised an issue that Happy Eyeballs can make users not notice IPv6 deployment issues

Authors believe this is important to discuss, but goes beyond the core algorithm RFC

Should clients automatically report issues? Via what mechanism?

Still an area of open discussion and investigation, could apply to many client retry behaviors

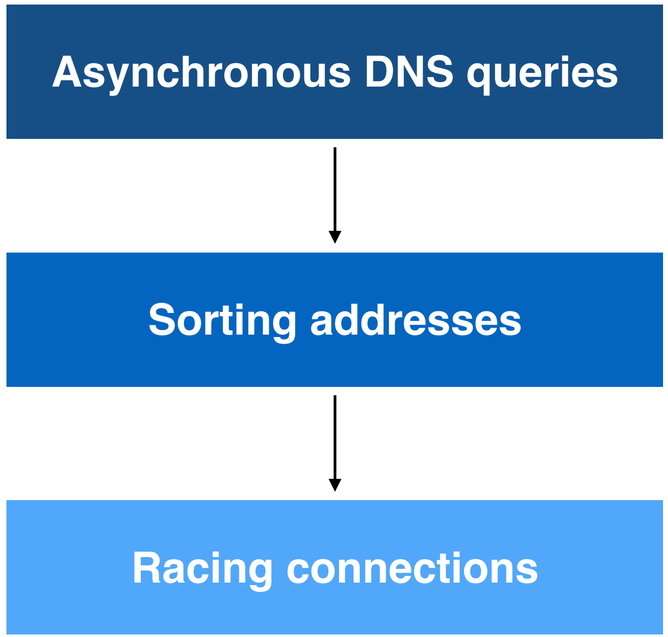
Happy Eyeballs, **Version 3**

Why a new version?

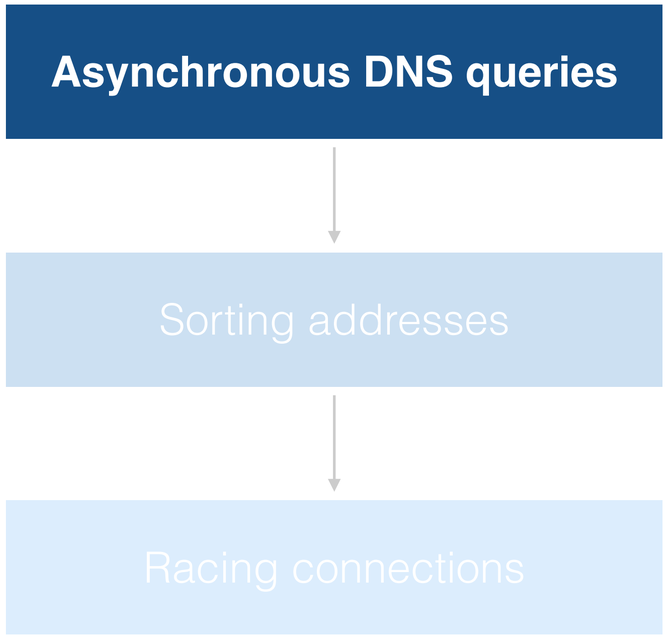
Lots of changes to DNS and transports since RFC 8305!

- QUIC standardization
- More IPv6-only networks and mechanisms
- SVCB / HTTPS records (address hints, priorities, ALPN)
- Encrypted Client Hello

Updated Algorithm



Updated Algorithm



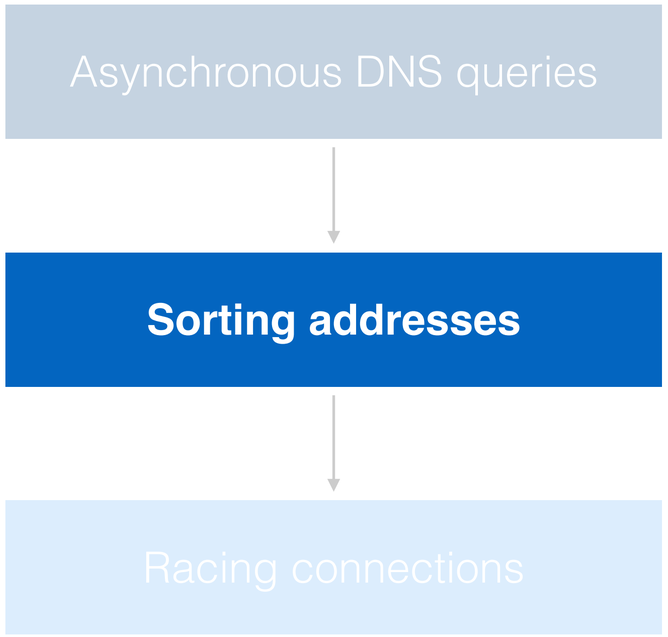
Prefer IPv6 DNS resolver addresses

Query **SVCB** / AAAA / A RRs in parallel (**SVCB** first)

Act on AAAA responses immediately, **if SVCB RRs not requested**

Wait up to 50ms for **SVCB and/or AAAA** if A comes back first

Updated Algorithm



Prefer ECH keys, if present

SVCB priorities, if present

Preferred ALPNs, if present

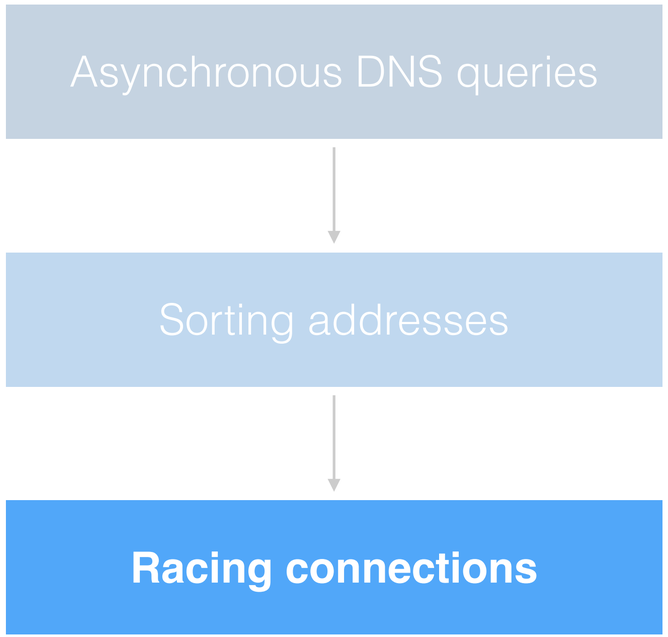
RFC 6724 address sorting

Historical round trip times

Place IPv6 at the start; two IPv6 addresses first if available

Use IPv4 first if historical data shows IPv6 brokenness

Updated Algorithm



Start attempts to addresses in parallel, staggered by a delay

Delay based on the TCP / **QUIC** retransmission timeout

Race ends when one connection completes

Racing through **full handshake (TCP / TLS / QUIC, etc.)**

Generalizing for QUIC

Prefer services with QUIC-capable ALPNs when sorting endpoints, after ECH keys and SvcPriority

- QUIC provides improved delivery and congestion control, connection migration, etc.

Adjust connection establishment logic to not just mention TCP

- Race until QUIC completes
- Also allow racing until TLS above TCP completes

ECH considerations

If client is SVCB-optional,

- May start a TCP handshake, but not TLS/QUIC
- Wait until a timeout for "`ech`" SvcParamKey
  - Is it reasonable to proceed if the timer expires?

If client is/becomes SVCB-reliant,

- Wait until "`ech`" SvcParamKey to start TLS/QUIC handshake
  - Is it safe to start a TCP handshake?

Dispatching

Options that have been previously suggested:

Existing WGs

- v6ops, where the original Happy Eyeballs RFCs were developed
- tsvwg, due to transport selection impact
- ...others?

Form a new WG? *(Seems heavyweight...)*

- WIT AREA, focused on client algorithms and operations

No matter what, should work with IPv6 + DNS + Transport + TLS experts to review