Workgroup: HAPPY Working Group
Internet-Draft: draft-ietf-happy-happyeyeballs-v3-02
Published: 20 October 2025
Intended Status: Standards Track
Expires: 23 April 2026
Authors: T. Pauly *Apple Inc*
D. Schinazi *Google LLC*
N. Jaju *Google LLC*
K. Ishibashi *Google LLC*

Happy Eyeballs Version 3: Better Connectivity Using Concurrency

Abstract

Many communication protocols operating over the modern Internet use hostnames. These often resolve to multiple IP addresses, each of which may have different performance and connectivity characteristics. Since specific addresses or address families (IPv4 or IPv6) may be blocked, broken, or sub-optimal on a network, clients that attempt multiple connections in parallel have a chance of establishing a connection more quickly. This document specifies requirements for algorithms that reduce this user-visible delay and provides an example algorithm, referred to as "Happy Eyeballs". This document updates the algorithm description in RFC 8305.

The RFC Editor will remove this note

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://ietf-wg-happy/draft-happy-eyeballs-v3/draft-ietf-happy-happyeyeballs-v3.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-happy-happyeyeballs-v3/.

Discussion of this document takes place on the HAPPY Working Group mailing list (mailto:happy@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/happy/. Subscribe at https://www.ietf.org/mailman/listinfo/happy/.

Source for this draft and an issue tracker can be found at https://github.com/ietf-wg-happy/draft-happy-eyeballs-v3.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material

or to cite them other than as "work in progress."

This Internet-Draft will expire on 23 April 2026.

Copyright Notice

Table of Contents

1. Introduction

Many communication protocols operating over the modern Internet use hostnames. These often resolve to multiple IP addresses, each of which may have different performance and connectivity characteristics. Since specific addresses or address families (IPv4 or IPv6) may be blocked, broken, or sub-optimal on a network, clients that attempt multiple connections in parallel have a chance of establishing a connection more quickly. This document specifies requirements for algorithms that reduce this user-visible delay and provides an example algorithm.

This document defines the algorithm for "Happy Eyeballs", a technique for reducing user-visible delays on dual-stack hosts. This definition updates the description in [HEV2], which itself obsoleted [RFC6555].

The Happy Eyeballs algorithm of racing connections to resolved addresses has several stages to avoid delays to the user whenever possible, while respecting client priorities, such as preferring the use of IPv6 or the availability of protocols like HTTP/3 [HTTP3] and TLS Encrypted Client Hello [ECH]. This document discusses how to initiate DNS queries when starting a connection, how to sort the list of destination addresses received from DNS answers, and how to race the connection attempts.

The major difference between the algorithm defined in this document and [HEV2] is the addition of support for SVCB / HTTPS resource records [SVCB]. SVCB records provide alternative endpoints and information about application protocol support, Encrypted Client Hello [ECH] keys, address hints, and other

relevant details about the services being accessed. Discovering protocol support during resolution, such as for HTTP/3 over QUIC [HTTP3], allows upgrading between protocols on the current connection attempts, instead of waiting for subsequent attempts to use information from other discovery mechanisms such as HTTP Alternative Services [AltSvc]. These records can be queried along with A and AAAA records, and the updated algorithm defines how to handle SVCB responses to improve connection establishment.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Overview

This document defines a method of connection establishment, named the "Happy Eyeballs Connection Setup". This approach has several distinct phases:

1. Asynchronous resolution of a hostname into destination addresses (Section 4)
2. Sorting of the resolved destination addresses (Section 5)
3. Initiation of asynchronous connection attempts (Section 6)
4. Successful establishment of one connection and cancellation of other attempts (Section 6)

Note that this document assumes that the preference policy for the host destination address favors IPv6 over IPv4. IPv6 has many desirable properties designed to be improvements over IPv4 [IPV6].

This document also assumes that the preference policy favors QUIC over TCP. QUIC only requires one packet to establish a secure connection, making it quicker compared to TCP [QUIC].

If the host is configured to have different preferences, the recommendations in this document can be easily adapted.

4. Hostname Resolution

When a client is trying to establish a connection to a named host, it needs to determine which destination IP addresses it can use to reach the host. The client resolves the hostname into IP addresses by sending DNS queries and collecting the answers. This section describes how a client initiates DNS queries and asynchronously handles the answers.

4.1. Sending DNS Queries

Clients first need to determine which DNS resource records they will include in queries for a named host.

This decision is based on if client has "connectivity" using IPv4 and IPv6. In this case, "connectivity" for an address family is defined as having at least one local address of the family from which to send packets, and at least one non-link local route for the address family.

When a client has both IPv4 and IPv6 connectivity, it needs to send out queries for both AAAA and A records. On a network with only IPv4 connectivity, it will send a query for A records. On a network with only IPv6 connectivity, the client will either send out queries for both AAAA and A records, or only a query for AAAA records, depending on the network configuration. See Section 8 for more discussion of handling

IPv6-mostly and IPv6-only networks.

In addition to requesting AAAA and A records, depending on which application is establishing the connection, clients can request either SVCB or HTTPS records [SVCB]. For applications using HTTP or HTTPS (including applications using WebSockets), the client SHOULD send a query for HTTPS records.

All of the DNS queries SHOULD be made as soon after one another as possible. The order in which the queries are sent SHOULD be as follows (omitting any query that doesn't apply based on the logic described above):

1. SVCB or HTTPS query
2. AAAA query
3. A query

4.2. Handling DNS Answers Asynchronously

Once the client receives sufficient answers to its DNS queries, it can move onto the phases of sorting addresses (Section 5) and establishing connections (Section 6).

Implementations SHOULD NOT wait for all answers to return before starting the next steps of connection establishment. If one query fails or takes significantly longer to return, waiting for those answers can significantly delay connection establishment that could otherwise proceed with already received answers.

Therefore, the client SHOULD treat DNS resolution as asynchronous, processing different record types independently. Note that if the platform does not offer an asynchronous DNS API, this behavior can be simulated by making separate synchronous queries for each record type in parallel.

The client moves onto sorting addresses and establishing connections once one of the following condition sets is met:

Either:

- Some positive (non-empty) address answers have been received AND
- A postive (non-empty) or negative (empty) answer has been received for the preferred address family that was queried AND
- SVCB/HTTPS service information has been received (or has received a negative response)

Or:

- Some positive (non-empty) address answers have been received AND
- A resolution time delay has passed after which other answers have not been received

Positive answers can be addresses received either from AAAA or A records, or address hints received directly in SVCB/HTTPS records.

Negative answers are exclusively responses to AAAA or A records that contain no addresses (with or without an error like NXDOMAIN). If all answers come back with negative answers, the connection establishment will fail or need to wait until other answers are received.

On networks that have both default routes for IPv6 and IPv4, IPv6 is assumed to be the preferred address family. If only one of IPv6 or IPv4 has a default route, that address family should be considered the preferred address family for progressing the algorithm.

The resolution time delay is a short time that provides a chance to receive preferred addresses (via AAAA records) along with service information (via SVCB/HTTPS records). This accounts for the case where the AAAA or SVCB/HTTPS records follow the A records by a few milliseconds. This delay is referred to as the "Resolution Delay".

The RECOMMENDED value for the Resolution Delay is 50 milliseconds.

4.2.1. Resolving SVCB/HTTPS Aliases and Targets

SVCB and HTTPS records describe information for network services. Individual records are either AliasMode or ServiceMode records, where AliasMode requires another SVCB/HTTPS query for the alias name. ServiceMode records either are associated with the original name being queried, in which case their TargetName is ".", or are associated with another service name (see Section 2.5 of [SVCB]).

The algorithm in this document does not consider service information to be received until ServiceMode records are available.

ServiceMode records can contain address hints via `ipv6hint` and `ipv4hint` parameters. When these are received, they SHOULD be considered as positive non-empty answers for the purpose of the algorithm when A and AAAA records corresponding to the TargetName are not available yet. Note that clients are still required to issue A and AAAA queries for those TargetNames if they haven't yet received those records. When those records are received, they replace the hints and update the available set of responses as new answers (see Section 4.3).

4.2.2. Examples

TODO: Provide examples of various scenarios (simple dual stack, SVCB, delayed AAAA, delayed SVCB, SVCB hints providing early answers)

4.3. Handling New Answers

If new records arrive while connection attempts are in progress, but before any connection has been established, then any newly received addresses are incorporated into the list of available candidate addresses (see Section 7) and the process of connection attempts will continue with the new addresses added, until one connection is established.

4.4. Handling Multiple DNS Server Addresses

If multiple DNS server addresses are configured for the current network, the client may have the option of sending its DNS queries over IPv4 or IPv6. In keeping with the Happy Eyeballs approach, queries SHOULD be sent over IPv6 first (note that this is not referring to the sending of AAAA or A queries, but rather the address of the DNS server itself and IP version used to transport DNS messages). If DNS queries sent to the IPv6 address do not receive responses, that address may be marked as penalized and queries can be sent to other DNS server addresses.

As native IPv6 deployments become more prevalent and IPv4 addresses are exhausted, it is expected that IPv6 connectivity will have preferential treatment within networks. If a DNS server is configured to be accessible over IPv6, IPv6 should be assumed to be the preferred address family.

Client systems SHOULD NOT have an explicit limit to the number of DNS servers that can be configured, either manually or by the network. If such a limit is required by hardware limitations, the client SHOULD use at least one address from each address family from the available list.

5. Grouping and Sorting Addresses

Before attempting to connect to any of the resolved destination addresses, the client defines the order in which to start the attempts. Once the order has been defined, the client can use a simple algorithm for racing each option after a short delay (see Section 6). It is important that the ordered list involve all addresses from both families and all protocols that have been received by this point, as this allows the client to get the racing effect of Happy Eyeballs for the entire list, not just the first IPv4 and first IPv6 addresses.

The client performs three levels of grouping and sorting of addresses based on the DNS answers received. Each subsequent level of sorting only changes orders and preferences within the previously defined groups.

1. Grouping and sorting by application protocol and security requirements (Section 5.1)
2. Grouping and sorting by service priorities (Section 5.2)
3. Sorting by destination address preferences (Section 5.3)

5.1. Grouping By Application Protocols and Security Requirements

Clients first group based on which application protocols the destination endpoints support and which security features those endpoints offer. These are based on information from SVCB/HTTPS records about application-layer protocols ("alpn" values) and other parameters like TLS Encrypted Client Hello configuration ("ech" values, see [SVCB-ECH]).

For cases where the answers do not include any SVCB/HTTPS information, or if all of the answers are associated with the same SVCB/HTTPS record, this step is trivial: all answers belong to one group, and the client assumes they support the same protocols and security properties.

However, the client is aware of different sets of destination endpoints that advertise different capabilities when it receives multiple distinct SVCB/HTTPS records. The client SHOULD separate these addresses into different groups, such that all addresses in a group share the same application protocols and relevant security properties. The specific parameters that are relevant to the client depend on the client implementation and application.

Note that some destination addresses might need to be added to multiple groups at this stage. For example, consider the following HTTPS records:

```
example.com. 60 IN HTTPS 1 svc1.example.com. (
    alpn="h3,h2" ipv6hint=2001:db8::2 )
example.com. 60 IN HTTPS 1 svc2.example.com. (
    alpn="h2" ipv6hint=2001:db8::4 )
```

In this case, 2001:db8::2 can be used with HTTP/3 and HTTP/2, but 2001:db8::4 can only be used with HTTP/2. If the client creates a grouping for HTTP/3-capable addresses and HTTP/2-capable addresses, 2001:db8::2 would exist in both groups (assuming that all other security properties are the same).

Connection racing as described in Section 6 applies to different destination address options within one of these groups. The logic for prioritizing and falling back between groups of addresses with different security properties and protocol properties is implementation-defined.

5.1.1. When to Apply Application Preferences

Whether or not specific application protocols or security features are grouped separately is a client application decision. Clients SHOULD avoid grouping and sorting separately in cases where their use of an application protocol or feature is non-critical.

For example, an HTTP client loading a simple webpage may not see a large difference between using HTTP/3 or HTTP/2, and thus can group the ALPNs together to respect service-determined priorities where HTTP/3 might be prioritized behind HTTP/2. However, another client might see significant performance improvements by using HTTP/3's ability to send unreliable frames for its application use-case and will group HTTP/3 before HTTP/2.

Similarly, a particular application might require or strongly prefer the use of TLS ECH for privacy-sensitive traffic, while others might support ECH opportunistically.

Section 8 of [SVCB-ECH] recommends against SVCB record sets that contain some answers that include ECH configuration and some that don't, but notes that such cases are possible. It is possible that services only include ECH configurations on SVCB answers that are prioritized behind others that don't include ECH configurations; for example, this might be used as an experimenation or roll-out strategy. Due to such cases, clients ought to not arbitrarily group ECH-containing answers and sort them first if they won't use the ECH information, or if the connection would not benefit from the use of ECH. However, for cases where there is a reason for an application preference for ECH, the client MAY group and prioritize those answers separately. Even though this might conflict with the published service record priorities, any answers published by the service are eligible to be used by clients, and clients can choose to use them.

5.2. Grouping By Service Priority

The next step of grouping and sorting is to group across different services (as defined by SVCB/HTTPS records), and sort these groups by priority.

This step allows server-published priorities to be reflected in the client connection establishment algorithm.

SVCB [SVCB] records indicate a priority for each ServiceMode response. This priority applies to any IPv4 or IPv6 address hints in the record itself, as well as any addresses received on A or AAAA queries for the name in the ServiceMode record. The priority in a SVCB ServiceMode record is always greater than 0.

SVCB answers with the lowest numerical value (such as 1) are sorted first, and answers with higher numerical values are sorted later.

Note that a SVCB record with the TargetName "." applies to the owner name in the record, and the priority of that SVCB record applies to any A or AAAA records for the same owner name. These answers are sorted according to that SVCB record's priority.

All addresses received from a particular SVCB service (within a group as defined in Section 5.1), either by an associated AAAA or A record or address hints, SHOULD be separated into a group by the client. These service-based groups SHOULD then be sorted using the service priority.

For cases where the answers do not include any SVCB/HTTPS information, or if all of the answers are associated with the same SVCB/HTTPS record, this step is trivial: all answers belong to one group that has the same priority.

When there are multiple services, and thus multiple groups, with the same priority, the client SHOULD shuffle these groups randomly.

If there are some SVCB/HTTPS services received, but there are AAAA or A records that do not have an associated service (for example, if no SVCB/HTTPS record is received for the original name using the "." TargetName), the unassociated addresses SHOULD be put in a group that is prioritized at the end of the list.

5.3. Sorting Destination Addresses Within Groups

Within each group of addresses, after grouping based on the logic in Section 5.1 and Section 5.2, the client sorts the addresses based on preference and historical data.

First, the client MUST sort the addresses using Destination Address Selection ([RFC6724], Section 6).

If the client is stateful and has a history of expected round-trip times (RTTs) for the routes to access each address, it SHOULD add a Destination Address Selection rule between rules 8 and 9 that prefers addresses with lower RTTs. If the client keeps track of which addresses it used in the past, it SHOULD add another Destination Address Selection rule between the RTT rule and rule 9, which prefers used addresses over unused ones. This helps servers that use the client's IP address during authentication, as is the case for TCP Fast Open [RFC7413] and some Hypertext Transport Protocol (HTTP) cookies. This historical data MUST be partitioned using the same boundaries used for privacy-sensitive information specific to that endpoint, and MUST NOT be used across different network interfaces. The data SHOULD be flushed whenever a device changes the network to which it is attached. Clients that use historical data MUST ensure that clients with different historical data will eventually converge toward the same behaviors. For example, clients can periodically ignore historical data to ensure that fresh addresses are attempted.

Next, the client SHOULD modify the ordered list to interleave address families. Whichever address family is first in the list should be followed by an endpoint of the other address family. For example, if the first address in the sorted list is an IPv6 address, then the first IPv4 address should be moved up in the list to be second in the list. An implementation MAY choose to favor one address family more by allowing multiple addresses of that family to be attempted before trying the next. The number of contiguous addresses of the first address family of properties will be referred to as the "Preferred Address Family Count" and can be a configurable value. This avoids waiting through a long list of addresses from a given address family if connectivity over that address family is impaired.

Note that the address selection described in this section only applies to destination addresses; Source Address Selection ([RFC6724-UPDATE], Section 3.2) is performed once per destination address and is out of scope of this document.

6. Connection Attempts

Once the list of addresses received up to this point has been constructed, the client will attempt to make connections. In order to avoid unreasonable network load, connection attempts SHOULD NOT be made simultaneously. Instead, one connection attempt to a single address is started first, followed by the others, one at a time. Starting a new connection attempt does not affect previous attempts, as multiple connection attempts may occur in parallel. Once one of the connection attempts succeeds (Section 6.1), all other connections attempts that have not yet succeeded SHOULD be canceled. Any address that was not yet attempted as a connection SHOULD be ignored. At that time, any asynchronous DNS queries MAY be canceled as new addresses will not be used for this connection. However, the DNS client resolver SHOULD still process DNS replies from the network for a short period of time (recommended to be 1 second), as they will populate the DNS cache and can be used for subsequent connections.

If grouping addresses by application or security requirements (Section 5.1) produced multiple groups, the application SHOULD start with connection attempts to the most preferred option. The policy for attempting any addresses outside of the most preferred group is up to the client implementation and out of scope for this document.

If grouping addresses by service (Section 5.2) produced multiple groups, all of the addresses of the first group SHOULD be started before starting attempts using the next group. Attempts across service groups SHOULD be allowed to continue in parallel; in effect, the groups are flattened into a single list.

A simple implementation can have a fixed delay for how long to wait before starting the next connection attempt. This delay is referred to as the "Connection Attempt Delay". One recommended value for a default delay is 250 milliseconds. A more nuanced implementation's delay should correspond to the time when the previous attempt is retrying its handshake (such as sending a second TCP SYN or a second QUIC Initial), based on the retransmission timer ([RFC6298], [RFC9002]). If the client has historical RTT data gathered from other connections to the same host or prefix, it can use this information to influence its delay. Note that this algorithm should only try to approximate the time of the first handshake packet retransmission, and not any further retransmissions that may be influenced by exponential timer back off.

The Connection Attempt Delay MUST have a lower bound, especially if it is computed using historical data. More specifically, a subsequent connection MUST NOT be started within 10 milliseconds of the previous attempt. The recommended minimum value is 100 milliseconds, which is referred to as the "Minimum Connection Attempt Delay". This minimum value is required to avoid congestion collapse in the presence of high packet-loss rates. The Connection Attempt Delay SHOULD have an upper bound, referred to as the "Maximum Connection Attempt Delay". The current recommended value is 2 seconds.

The Connection Attempt Delay is used to set a timer, referred to as the "Next Connection Attempt Timer". Whenever this timer fires and a connection has not been successfully established, the next connection attempt starts, and the timer either is reset to a new delay value or, in the case of the end of the list being reached, is cancelled. Note that the delay value can be different for each connection attempt (depending on the protocol being used and the estimated RTT).

6.1. Determining successful connection establishment

The determination of when a connection attempt has successfully completed (and other attempts can be cancelled) ultimately depends on the client application's interpretation of the connection state being ready to use. This will generally include at least the transport-level handshake with the remote endpoint (such as the TCP or QUIC handshake), but can involve other higher-level handshakes or state checks as well.

Client connections that use TCP only (without TLS or another protocol on top, such as for unencrypted HTTP connections) will determine successful establishment based on completing the TCP handshake only. When TLS is used on top of of TCP (such as for encrypted HTTP connections), clients MAY choose to wait for the TLS handshake to successfully complete before cancelling other connection attempts. This is particularly useful for networks in which a TCP-terminating proxy might be causing TCP handshakes to succeed quickly, even though end-to-end connectivity with the TLS-terminating server will fail. QUIC connections inherently include a secure handshake in their main handshakes, and thus usually only need to wait for a single handshake to complete.

Beyond TCP, TLS, and/or QUIC handshakes, clients may also wait for other requirements to be met before determining that the connection establishment was successful. For example, clients generally validate that the server's certificate provided via TLS is trusted, and that operation can be asynchronous.

In cases where the connection establishment determination goes beyond the initial transport handshake,

the Next Connection Attempt Timer ought to be adjusted after the initial transport handshake is completed. When the connection establishment makes progress, but has not completed, the timer SHOULD be extended to a new value that represents an estimated time for the full connection establishment to complete.

For example, consider a case where connection establishment involves both a TCP handshake and a TLS handshake. If the timer is initially set to be roughly at the time when a TCP SYN packet would be retransmitted, and the TCP handshake completes before the timer fires, the timer should be adjusted to allow for the time in which the TLS handshake could complete.

While transport layer handshakes generally do not have restrictions on attempts to establish a connection, some cryptographic handshakes may be dependent on SVCB ServiceMode records and could impose limitations on establishing a connection. For instance, ECH-capable clients may become SVCB-reliant clients (Section 3 of [SVCB]) when SVCB records contain the "ech" SvcParamKey [SVCB-ECH]. If the client is either an SVCB-reliant client or a SVCB-optional client that might switch to SVCB-reliant connection establishment during the process, the client MUST wait for SVCB records before proceeding with the cryptographic handshake.

6.2. Handling Application Layer Protocol Negotiation (ALPN)

The `alpn` and `no-default-alpn` SvcParamKeys in SVCB records indicate the "SVCB ALPN set," which specifies the underlying transport protocols supported by the associated service endpoint. When the client requests SVCB records, it SHOULD perform the procedure specified in Section 7.1.2 of [SVCB] to determine the underlying transport protocols that both the client and the service endpoint support. The client SHOULD NOT attempt to make a connection to a service endpoint whose SVCB ALPN set does not contain any protocols that the client supports. For example, suppose the client is an HTTP client that only supports TCP-based versions such as HTTP/1.1 and HTTP/2, and it receives the following HTTPS record:

```
example.com. 60 IN HTTPS 1 svc1.example.com. (
    alpn="h3" no-default-alpn ipv6hint=2001:db8::2 )
```

In this case, attempting a connection to 2001:db8::2 or any other address resolved for `svc1.example.com` would be incorrect because the record indicates that `svc1.example.com` only supports HTTP/3, based on the ALPN value of "h3".

If the client is an HTTP client that supports both Alt-Svc [AltSvc] and SVCB (HTTPS) records, the client SHOULD ensure that connection attempts are consistent with both the Alt-Svc parameters and the SVCB ALPN set, as specified in Section 9.3 of [SVCB].

6.3. Dropping or Pending Connection Attempts

Some situations related to handling SVCB responses can require connection attempts to be dropped, or pended until SVCB responses return.

Section 3.1 of [SVCB] describes client behavior for handling resolution failures when responses are "cryptographically protected" using DNSSEC [DNSSEC] or encrypted DNS ([DOT], [DOH], or [DOQ], for example). If SVCB resolution fails when using cryptographic protection, clients SHOULD abandon connection attempts altogether to avoid downgrade attacks.

Use of cryptographic protection in DNS can influence other parts of Happy Eyeballs connection establishment, as well.

Situations in which DNS is not protected allow for any records to be blocked or modified, so security properties derived from SVCB records are opportunistic only. However, when DNS is cryptographically protected, clients can be stricter about relying on the properties from SVCB records.

Section 5.1 of [SVCB] explains that clients "MUST NOT transmit any information that could be altered by the SVCB response until it arrives", and specifically mentions properties that affect the TLS ClientHello. This restriction specifically applies when a client's behavior will be altered by the SVCB response, which depends both on the client implementation's ability to support a particular feature, as well as the client implementation's willingness to rely on the SVCB response to enable a particular feature.

Based on this, clients in some scenarios MUST pend starting a TLS handshake (either after TCP or as part of QUIC) until SVCB responses have been received, even after the "Resolution Delay" defined in Section 4 has been reached. Specifically, clients MUST pend starting handshakes if *all* of the following are true:

1. DNS responses are cryptographically protected with DNSSEC or encrypted DNS. Note that, if unencrypted and unsigned DNS is used, SVCB information is opportunistic; clients MAY wait for SVCB responses but do not need to.
2. The client implementation supports parsing and using a particular security-related SVCB parameter, such as the "ech" SvcParamKey [SVCB-ECH]. (In contrast, implementations that do not support actively using ECH do not need to wait for SVCB resolution if that is the only reason to do so).
3. The client relies on the presence of the particular SVCB-related parameter to enable the relevant protocol feature. For example, if a connection attempt would normally be using cleartext HTTP unless an HTTPS DNS record would cause the client to upgrade, the client needs to wait for the record; however, if the client already would be using HTTP over TLS, then it is not relying on that signal from SVCB. As another example, some SVCB properties can affect the TLS ClientHello in ways that optimize performance (like `tls-supported-groups` [I-D.ietf-tls-key-share-prediction]) but only aim to save round trips. The other TLS groups can be discovered through the TLS handshake itself, instead of SVCB, and thus do not require waiting for SVCB responses.

7. DNS Answer Changes During Happy Eyeballs Connection Setup

If, during the course of connection establishment, the DNS answers change by either adding resolved addresses (for example due to DNS push notifications [RFC8765]) or removing previously resolved addresses (for example, due to expiry of the TTL on that DNS record), the client should react based on its current progress. Additionally, once A and AAAA records are received, addresses received via SVCB hints that are not included in the A and AAAA records for the corresponding address family SHOULD be removed from the list, as specified in Section 7.3 of [SVCB].

If an address is removed from the list that already had a connection attempt started, the connection attempt SHOULD NOT be canceled, but rather be allowed to continue. If the removed address had not yet had a connection attempt started, it SHOULD be removed from the list of addresses to try.

If an address is added to the list, it should be sorted into the list of addresses not yet attempted according to the rules above (see Section 5).

8. Supporting IPv6-Mostly and IPv6-Only Networks

While many IPv6 transition protocols have been standardized and deployed, most are transparent to client devices. Supporting IPv6-only networks often requires specific client-side changes, especially when interacting with IPv4-only services. Two primary mechanisms for this are the combined use of NAT64

[RFC6146] with DNS64 [RFC6147], or leveraging NAT64 with a discovered PREF64 prefix [RFC8781].

One possible way to handle these networks is for the client device networking stack to implement 464XLAT [RFC6877]. 464XLAT has the advantage of not requiring changes to user space software; however, it requires per-packet translation if the application is using IPv4 literals and does not encourage client application software to support native IPv6. On platforms that do not support 464XLAT, the Happy Eyeballs engine SHOULD follow the recommendations in this section to properly support IPv6-mostly ([V6-MOSTLY]) and IPv6-only networks.

The features described in this section SHOULD only be enabled when the host detects an IPv6-mostly or IPv6-only network. A simple heuristic to detect one of these networks is to check if the network offers routable IPv6 addressing, does not offer routable IPv4 addressing, and offers a DNS resolver address.

8.1. IPv4 Address Literals

If client applications or users wish to connect to IPv4 address literals, the Happy Eyeballs engine will need to perform NAT64 address synthesis for them. The solution is similar to "Bump-in-the-Host" [RFC6535] but is implemented inside the Happy Eyeballs client.

Note that some IPv4 prefixes are scoped to a given host or network, such as 0.0.0.0/8, 127.0.0.0/8, 169.254.0.0/16, and 255.255.255.255/32, and therefore do not require NAT64 address synthesis.

8.2. Discovering and Utilizing PREF64

When an IPv4 address is passed into the Happy Eyeballs implementation instead of a hostname, it SHOULD use PREF64s received from Router Advertisements [RFC8781].

With PREF64 available, networks might choose to not deploy DNS64, as the latter has a number of disadvantages (see [V6-MOSTLY], Section 4.3.4). To ensure compatibility with such networks, if PREF64 is available, clients SHOULD send an A query in addition to an AAAA query for a given hostname. This allows the client to receive any existing IPv4 A records and perform local NAT64 address synthesis, eliminating the network's need to run DNS64.

If the network does not provide PREF64s, the implementation SHOULD query the network for the NAT64 prefix using "Discovery of the IPv6 Prefix Used for IPv6 Address Synthesis" [RFC7050]. It then synthesizes an appropriate IPv6 address (or several) using the encoding described in "IPv6 Addressing of IPv4/ IPv6 Translators" [RFC6052]. The synthesized addresses are then inserted into the list of addresses as if they were results from DNS A queries; connection attempts follow the algorithm described above (see Section 6).

Such translation also applies to any IPv4 addresses received in A records and IPv4 address hints received in SVCB records.

8.3. Supporting DNS64

If PREF64 is not available and the NAT64 prefix cannot be discovered, clients SHOULD assume the network is relying on DNS64 for IPv4-to-IPv6 address synthesis. In this scenario, clients will typically only receive AAAA records from DNS queries, as DNS64 servers synthese these records for IPv4-only domains.

8.4. Hostnames with Broken AAAA Records

At the time of writing, there exist a small but non-negligible number of hostnames that resolve to valid A records and broken AAAA records, which we define as AAAA records that contain seemingly valid IPv6

addresses but those addresses never reply when contacted on the usual ports. These can be, for example, caused by:

- Mistyping of the IPv6 address in the DNS zone configuration
- Routing black holes
- Service outages

While an algorithm complying with the other sections of this document would correctly handle such hostnames on a dual-stack network, they will not necessarily function correctly on IPv6-only networks with NAT64 and DNS64. Since DNS64 recursive resolvers rely on the authoritative name servers sending negative (no error, no data) responses for AAAA records in order to synthesize, they will not synthesize records for these particular hostnames and will instead pass through the broken AAAA record.

In order to support these scenarios, the client device needs to query the DNS for the A record and then perform local synthesis. Since these types of hostnames are rare and, in order to minimize load on DNS servers, this A query should only be performed when the client has given up on the AAAA records it initially received. This can be achieved by using a longer timeout, referred to as the "Last Resort Local Synthesis Delay"; the delay is recommended to be 2 seconds. The timer is started when the last connection attempt is fired. If no connection attempt has succeeded when this timer fires, the device queries the DNS for the IPv4 address and, on reception of a valid A record, treats it as if it were provided by the application (see Section 8.1).

8.5. Virtual Private Networks

Some Virtual Private Networks (VPNs) may be configured to handle DNS queries from the device. The configuration could encompass all queries or a subset such as "*.internal.example.com". These VPNs can also be configured to only route part of the IPv4 address space, such as 192.0.2.0/24. However, if an internal hostname resolves to an external IPv4 address, these can cause issues if the underlying network is IPv6-only. As an example, let's assume that "www.internal.example.com" has exactly one A record, 198.51.100.42, and no AAAA records. The client will send the DNS query to the company's recursive resolver and that resolver will reply with these records. The device now only has an IPv4 address to connect to and no route to that address. Since the company's resolver does not know the NAT64 prefix of the underlying network, it cannot synthesize the address. Similarly, the underlying network's DNS64 recursive resolver does not know the company's internal addresses, so it cannot resolve the hostname. Because of this, the client device needs to resolve the A record using the company's resolver and then locally synthesize an IPv6 address, as if the resolved IPv4 address were provided by the application (Section 8.1).

9. Summary of Configurable Values

The values that may be configured as defaults on a client for use in Happy Eyeballs are as follows:

- Resolution Delay (Section 4): The time to wait for a AAAA record after receiving an A record. Recommended to be 50 milliseconds.
- Preferred Address Family Count (Section 5): The number of addresses belonging to the preferred address family (such as IPv6) that should be attempted before attempting the next address family. Recommended to be 1; 2 may be used to more aggressively favor a particular combination of address family and protocol.
- Connection Attempt Delay (Section 6): The time to wait between connection attempts in the absence of RTT data. Recommended to be 250 milliseconds.
- Minimum Connection Attempt Delay (Section 6): The minimum time to wait between connection

attempts. Recommended to be 100 milliseconds. MUST NOT be less than 10 milliseconds.

- Maximum Connection Attempt Delay (Section 6): The maximum time to wait between connection attempts. Recommended to be 2 seconds.
- Last Resort Local Synthesis Delay (Section 8.4): The time to wait after starting the last IPv6 attempt and before sending the A query. Recommended to be 2 seconds.

The delay values described in this section were determined empirically by measuring the timing of connections on a very wide set of production devices. They were picked to reduce wait times noticed by users while minimizing load on the network. As time passes, it is expected that the properties of networks will evolve. For that reason, it is expected that these values will change over time. Implementors should feel welcome to use different values without changing this specification. Since IPv6 issues are expected to be less common, the delays SHOULD be increased with time as client software is updated.

10. Limitations

Happy Eyeballs will handle initial connection failures at the transport layer (such as TCP or QUIC); however, other failures or performance issues may still affect the chosen connection.

10.1. Path Maximum Transmission Unit Discovery

For TCP connections, since Happy Eyeballs is only active during the initial handshake and TCP does not pass the initial handshake, issues related to MTU can be masked and go unnoticed during Happy Eyeballs. For QUIC connections, a minimum MTU of at least 1200 bytes [RFC9000], Section 8.1-5 is guaranteed, but there is a chance that larger values may not be available. Solving this issue is out of scope of this document. One solution is to use "Packetization Layer Path MTU Discovery" [RFC4821].

10.2. Application Layer

If the DNS returns multiple addresses for different application servers, the application itself may not be operational and functional on all of them. Common examples include Transport Layer Security (TLS) and HTTP.

10.3. Hiding Operational Issues

It has been observed in practice that Happy Eyeballs can hide issues in networks. For example, if a misconfiguration causes IPv6 to consistently fail on a given network while IPv4 is still functional, Happy Eyeballs may impair the operator's ability to notice the issue. It is recommended that network operators deploy external means of monitoring to ensure functionality of all address families.

11. Security Considerations

Note that applications should not rely upon a stable hostname-to-address mapping to ensure any security properties, since DNS results may change between queries. Happy Eyeballs may make it more likely that subsequent connections to a single hostname use different IP addresses.

When using HTTP, HTTPS resource records indicate that clients should require HTTPS when connecting to an origin (see Section 9.5 of [RFC9460]), so an active attacker can attempt a downgrade attack by interfering with the successful delivery of HTTPS resource records. When clients use insecure DNS mechanisms, any on-path attacker can simply drop HTTPS resource records, so clients cannot tell the difference between an attack and a resolver that fails to respond to HTTPS queries.

However, when using cryptographically protected DNS mechanisms, as described in Section 3.1 of [RFC9460], both SVCB-reliant and SVCB-optional clients MUST NOT send any unencrypted data after the TCP handshake completes unless they have received a valid HTTPS response. Those clients need to complete a TLS handshake before proceeding if that response is non-negative.

12. IANA Considerations

This document does not require any IANA actions.

13. References

13.1. Normative References

[ECH] Rescorla, E., Oku, K., Sullivan, N., and C. A. Wood, "TLS Encrypted Client Hello", Work in Progress, Internet-Draft, draft-ietf-tls-esni-25, 14 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-25>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, DOI 10.17487/RFC4821, March 2007, <https://www.rfc-editor.org/rfc/rfc4821>.

[RFC6052] Bao, C., Huitema, C., Bagnulo, M., Boucadair, M., and X. Li, "IPv6 Addressing of IPv4/IPv6 Translators", RFC 6052, DOI 10.17487/RFC6052, October 2010, <https://www.rfc-editor.org/rfc/rfc6052>.

[RFC6146] Bagnulo, M., Matthews, P., and I. van Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers", RFC 6146, DOI 10.17487/RFC6146, April 2011, <https://www.rfc-editor.org/rfc/rfc6146>.

[RFC6147] Bagnulo, M., Sullivan, A., Matthews, P., and I. van Beijnum, "DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers", RFC 6147, DOI 10.17487/RFC6147, April 2011, <https://www.rfc-editor.org/rfc/rfc6147>.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <https://www.rfc-editor.org/rfc/rfc6298>.

[RFC6535] Huang, B., Deng, H., and T. Savolainen, "Dual-Stack Hosts Using "Bump-in-the-Host" (BIH)", RFC 6535, DOI 10.17487/RFC6535, February 2012, <https://www.rfc-editor.org/rfc/rfc6535>.

[RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, DOI 10.17487/RFC6724, September 2012, <https://www.rfc-editor.org/rfc/rfc6724>.

[RFC6724-UPDATE] Buraglio, N., Chown, T., and J. Duncan, "Prioritizing known-local IPv6 ULAs through address selection policy", Work in Progress, Internet-Draft, draft-ietf-6man-rfc6724-update-25, 11 August 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-6man-rfc6724-update-25>.

[RFC7050] Savolainen, T., Korhonen, J., and D. Wing, "Discovery of the IPv6 Prefix Used for IPv6 Address Synthesis", RFC 7050, DOI 10.17487/RFC7050, November 2013, <https://www.rfc-

editor.org/rfc/rfc7050>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8781] Colitti, L. and J. Linkova, "Discovering PREF64 in Router Advertisements", RFC 8781, DOI 10.17487/RFC8781, April 2020, <https://www.rfc-editor.org/rfc/rfc8781>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[RFC9460] Schwartz, B., Bishop, M., and E. Nygren, "Service Binding and Parameter Specification via the DNS (SVCB and HTTPS Resource Records)", RFC 9460, DOI 10.17487/RFC9460, November 2023, <https://www.rfc-editor.org/rfc/rfc9460>.

[SVCB] Schwartz, B., Bishop, M., and E. Nygren, "Service Binding and Parameter Specification via the DNS (SVCB and HTTPS Resource Records)", RFC 9460, DOI 10.17487/RFC9460, November 2023, <https://www.rfc-editor.org/rfc/rfc9460>.

[SVCB-ECH] Schwartz, B. M., Bishop, M., and E. Nygren, "Bootstrapping TLS Encrypted ClientHello with DNS Service Bindings", Work in Progress, Internet-Draft, draft-ietf-tls-svcb-ech-08, 16 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-svcb-ech-08>.

13.2. Informative References

[AltSvc] Nottingham, M., McManus, P., and J. Reschke, "HTTP Alternative Services", RFC 7838, DOI 10.17487/RFC7838, April 2016, <https://www.rfc-editor.org/rfc/rfc7838>.

[DNSSEC] Hoffman, P., "DNS Security Extensions (DNSSEC)", BCP 237, RFC 9364, DOI 10.17487/RFC9364, February 2023, <https://www.rfc-editor.org/rfc/rfc9364>.

[DOH] Hoffman, P. and P. McManus, "DNS Queries over HTTPS (DoH)", RFC 8484, DOI 10.17487/RFC8484, October 2018, <https://www.rfc-editor.org/rfc/rfc8484>.

[DOQ] Huitema, C., Dickinson, S., and A. Mankin, "DNS over Dedicated QUIC Connections", RFC 9250, DOI 10.17487/RFC9250, May 2022, <https://www.rfc-editor.org/rfc/rfc9250>.

[DOT] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)", RFC 7858, DOI 10.17487/RFC7858, May 2016, <https://www.rfc-editor.org/rfc/rfc7858>.

[HEV2] Schinazi, D. and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency", RFC 8305, DOI 10.17487/RFC8305, December 2017, <https://www.rfc-editor.org/rfc/rfc8305>.

[HTTP3] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/rfc/rfc9114>.

[I-D.ietf-tls-key-share-prediction] Benjamin, D., "TLS Key Share Prediction", Work in Progress, Internet-Draft, draft-ietf-tls-key-share-prediction-03, 29 August 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-key-share-prediction-03>.

[IPV6] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/rfc/rfc8200>.

[QUIC] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[RFC6555] Wing, D. and A. Yourtchenko, "Happy Eyeballs: Success with Dual-Stack Hosts", RFC 6555, DOI 10.17487/RFC6555, April 2012, <https://www.rfc-editor.org/rfc/rfc6555>.

[RFC6877] Mawatari, M., Kawashima, M., and C. Byrne, "464XLAT: Combination of Stateful and Stateless Translation", RFC 6877, DOI 10.17487/RFC6877, April 2013, <https://www.rfc-editor.org/rfc/rfc6877>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/rfc/rfc7413>.

[RFC8765] Pusateri, T. and S. Cheshire, "DNS Push Notifications", RFC 8765, DOI 10.17487/RFC8765, June 2020, <https://www.rfc-editor.org/rfc/rfc8765>.

[RFC9002] Iyengar, J., Ed. and I. Swett, Ed., "QUIC Loss Detection and Congestion Control", RFC 9002, DOI 10.17487/RFC9002, May 2021, <https://www.rfc-editor.org/rfc/rfc9002>.

[V6-MOSTLY] Buraglio, N., Caletka, O., and J. Linkova, "IPv6-Mostly Networks: Deployment and Operations Considerations", Work in Progress, Internet-Draft, draft-ietf-v6ops-6mops-04, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-v6ops-6mops-04>.

Acknowledgments

The authors thank Dan Wing, Andrew Yourtchenko, and everyone else who worked on the original Happy Eyeballs design [RFC6555], Josh Graessley, Stuart Cheshire, and the rest of team at Apple that helped implement and instrument this algorithm, and Jason Fesler and Paul Saab who helped measure and refine this algorithm. The authors would also like to thank Fred Baker, Nick Chettle, Lorenzo Colitti, Igor Gashinsky, Geoff Huston, Jen Linkova, Paul Hoffman, Philip Homburg, Warren Kumari, Erik Nygren, Jordi Palet Martinez, Rui Paulo, Stephen Strowes, Jinmei Tatuya, Dave Thaler, Joe Touch, and James Woodyatt for their input and contributions.

Authors' Addresses

Tommy Pauly
Apple Inc
Email: tpauly@apple.com

David Schinazi
Google LLC
Email: dschinazi.ietf@gmail.com

Nidhi Jaju
Google LLC
Email: nidhijaju@google.com

Kenichi Ishibashi
Google LLC
Email: bashi@google.com