v6ops	D. Wing
Internet-Draft	A. Yourtchenko
Intended status: Standards Track	Cisco
Expires: November 26, 2011	May 25, 2011

Happy Eyeballs: Trending Towards Success with Dual-Stack Hosts
draft-ietf-v6ops-happy-eyeballs-02

Abstract

This document describes an algorithm for a dual-stack client to quickly determine the functioning address family to a dual-stack server, and trend towards using that same address family for subsequent connections. This improves the dual-stack user experience during IPv6 or IPv4 server or network outages.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

In order to use HTTP successfully over IPv6, it is necessary that the user enjoys nearly identical performance as compared to IPv4. A combination of today's applications, IPv6 tunneling and IPv6 service providers, and some of today's content providers all cause the user experience to suffer (Section 3). For IPv6, a content provider may ensure a positive user experience by using a DNS white list of IPv6 service providers who peer directly with them, e.g. [whitelist]. However, this is not scalable to all service providers worldwide, nor is it scalable for other content providers to operate their own DNS white list.
Instead, this document suggests a mechanism for applications to quickly determine if IPv6 or IPv4 is the most optimal to connect to a server. The suggestions in this document provide a user experience which is superior to connecting to ordered IP addresses which is helpful during the IPv6/IPv4 transition with dual stack hosts.
This problem is also described in [RFC1671], published in 1994:

> *"The dual-stack code may get two addresses back from DNS; which does it use? During the many years of transition the Internet will contain black holes. For example, somewhere on the way from IPng host A to IPng host B there will sometimes (unpredictably) be IPv4-only routers which discard IPng packets. Also, the state of the DNS does not necessarily correspond to reality. A host for which DNS claims to know an IPng address may in fact not be running IPng at a particular moment; thus an IPng packet to that host will be discarded on delivery. Knowing that a host has both IPv4 and IPng addresses gives no information about black holes. A solution to this must be proposed and it must not depend on manually maintained information. (If this is not solved, the dual stack approach is no better than the packet translation approach.)"

Even after the transition, the procedure described in this document allows applications to strongly prefer IPv6 -- yet when an IPv6 outage occurs the application will quickly start using IPv4 and continue using IPv4. It will quietly continue trying to use IPv6 until IPv6 becomes available again, and then trend again towards using IPv6.
Following the procedures in this document, once a certain address family is successful, the application trends towards preferring that

address family. Thus, repeated use of the application DOES NOT cause repeated probes over both address families.
Applications would have to change in order to use the mechanism described in this document, by either implementing the mechanism directly, or by calling APIs made available to them. To improve IPv6 connectivity experience for legacy applications (e.g., applications which simply rely on the operating system's address preference order), operating systems may use other approaches. These can include changing address sorting based on configuration received from the network, other configuration, or dynamic detection of the host connectivity to IPv6 and IPV4 destinations.
While the application recommendations in this document are described in the context of HTTP clients ("web browsers") and SRV clients (e.g., XMPP clients) the procedure is also useful and applicable to other interactive applications.
Code which implements some of the ideas described in this document has been made available [Perreault] [Andrews].

2. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Problem Statement

As discussed in more detail in Section 3.1, it is important that the same URI and hostname be used for IPv4 and IPv6. Using separate namespaces causes namespace fragmentation and reduces the ability for users to share URIs and hostnames, and complicates printed material that includes the URI or hostname.
As discussed in more detail in Section 3.2, IPv6 connectivity is broken to specific prefixes or specific hosts, or slower than native IPv4 connectivity.

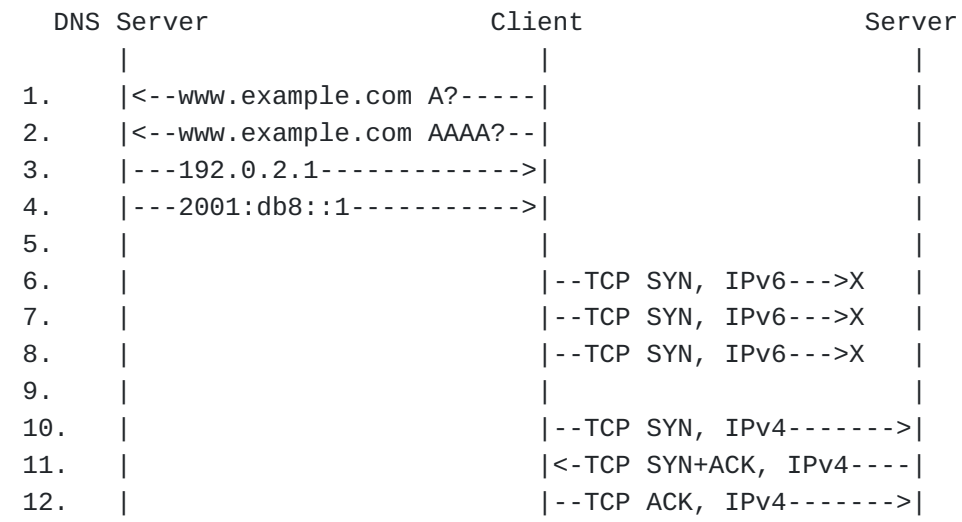
3.1. URIs and hostnames

URIs are often used between users to exchange pointers to content -- such as on social networks, email, instant messaging, or other systems. Thus, production URIs and production hostnames containing references to IPv4 or IPv6 will only function if the other party is also using an application, OS, and a network that can access the URI or the hostname.

3.2. IPv6 connectivity

When IPv6 connectivity is impaired, today's IPv6-capable web browsers incur many seconds of delay before falling back to IPv4. This harms the user's experience with IPv6, which will slow the acceptance of IPv6, because IPv6 is frequently disabled in its entirety on the end systems to improve the user experience.

Reasons for such failure include no connection to the IPv6 Internet, broken 6to4 or Teredo tunnels, and broken IPv6 peering.



The client obtains the IPv4 and IPv6 records for the server (1-4). The client attempts to connect using IPv6 to the server, but the IPv6 path is broken (6-8), which consumes several seconds of time. Eventually, the client attempts to connect using IPv4 (10) which succeeds. Delays experienced by users of various browser and operating system combinations have been studied [Experiences].

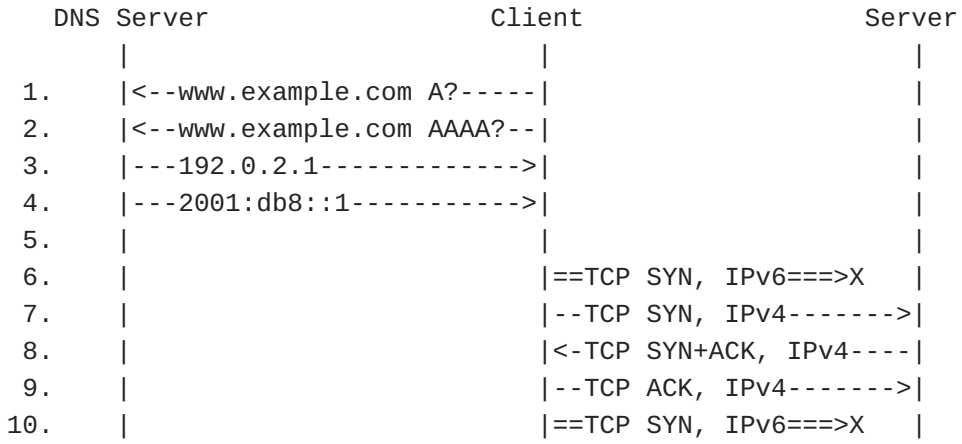
4. Client Recommendations

Happy Eyeballs does two things:

1. Provides fast connection for users. To provide fast connections for users, clients should make connections quickly over various technologies, automatically tune itself to avoid flooding the network with unnecessary connections (i.e., for technologies that have not made successful connections), and occasionally flush its self-tuning if it trended towards IPv4 Section 5.2.

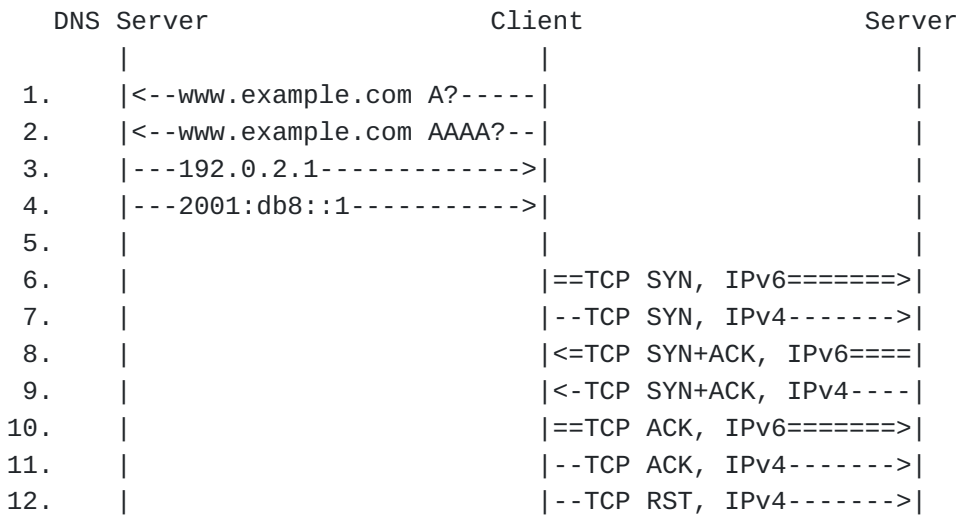
2. Avoids thrashing the network. Clients need to avoid flooding the network or servers with excessive connection initiation traffic. One way to accomplish this, without significant impairment to the user experience, is to cache which address family has been unsuccessful and successful, and use that address family for subsequent connections to the same host.

If a TCP client supports IPv6 and IPv4 and is connected to IPv4 and IPv6 networks, it can perform the procedures described in this section.



In the diagram above, the client sends two TCP SYNs at the same time over IPv6 (6) and IPv4 (7). In the diagram, the IPv6 path is broken but has little impact to the user because there is no long delay before using IPv4. The IPv6 path is retried until the application gives up (10).

After performing the above procedure, the client learns if connections to the host's IPv6 or IPv4 address were successful. The client MUST cache that information to avoid thrashing the network with excessive subsequent connection attempts. For example, in the diagram above, the client has noticed that IPv6 to that address failed, and it should provide a greater preference to using IPv4 instead.



The diagram above shows a case where both IPv6 and IPv4 are working, and IPv4 is abandoned (12).

5. Implementation details: A and AAAA

This section details how to provide robust dual stack service for both IPv6 and IPv4, so that the user perceives very fast application response.

Depending on implementation, the variables and procedures described below might be implemented or maintained within a specific application (e.g., web browser), library, framework, or by the operating system itself. An API call such as "connect_by_name()" is envisioned which would call the Happy Eyeballs routine and implement the functions described in this section.

5.1. Description of State Variables

The system maintains a Smoothed P (which provides the overall preference to IPv6 or IPv4), and an exception cache. Both of these change over time and are described below:

Exception Cache: This is a cache, indexed by IP prefixes, contains a "P" value for each prefix. Entries are added to this cache if a connection to the expected address family failed and a connection to the other address family succeeded. That is, these are exceptions to the Smoothed P variable. See Section 5.5 for description of how these prefixes are defined.

*(Note: In previous versions of this document, this was the "per-destination P (preference) value".)

P: Address family preference. This is computed for this connection attempt. A positive value is a preference to start the IPv6 connection first, a negative value to start the IPv4 connection first, and zero indicates both IPv6 and IPv4 connections are started simultaneously. The absolute value is the number of milliseconds between the connection attempts on two address families.

Smoothed P: Smoothed address family preference. This is the address family preference for destinations that are not in the exception cache. This variable can be positive or negative, with values having the same meaning as "P". In the absence of more specific configuration information, it is RECOMMENDED that implementations enforce a maximum value of 8000 (8 seconds) for this variable.

*(Note: In previous versions of this document, this was the "application-wide P (preference) value".)

The following values are configured and constant:

TI: Tolerance Interval, in milliseconds. This is the allowance in the time a connection is expected to complete and its actual completion, and is provided to accommodate slight differences in network and server responsiveness. In the absence of dynamic configuration

information from the network (e.g., DHCP) or other configuration information, it is RECOMMENDED to use 20ms.

Initial Headstart (IH): The initial headstart ("preference") for IPv6, in milliseconds. This value provides a preference towards IPv6 (if positive) or IPv4 (if negative) when the host joins a new network or otherwise flushes its cached information (see Section 5.2), and the distance to move P away from zero when P was zero. In the absence of dynamic configuration information from the network (e.g., [I-D.ietf-6man-addr-select-opt]) or other configuration information (e.g., the node's address selection policy has been modified to prefer IPv4 over IPv6), the value 100ms is recommended, which causes the initial IPv6 connection to be attempted 100ms before the IPv4 connection.

MAXWAIT: Maximum wait time for a connection to complete, before trying additional IP addresses. This is RECOMMENDED to be 10 seconds.

5.2. Initialization, Cache Flush, and Resetting Smoothed P

Because every network has different characteristics (e.g., working or broken IPv6 or IPv4 connectivity) the Smoothed P variable SHOULD be set to its default value (Smoothed P = Initial Headstart) and the exception cache SHOULD be emptied whenever the host is connected to a new network (e.g., DNAv4 *[RFC4436]*, DNAv6 *[RFC6059]*, [cx-osx], [cx-win]).
If there are IPv6 failures to specific hosts or prefixes, the exception cache will build up exception entries preferring IPv4, and if there are significant IPv6 failures to many hosts or prefixes, Smoothed P will become negative. When this occurs, IPv6 will not be attempted at all. To avoid this problem, it is strongly RECOMMENDED to occasionally flush the exception cache of all entries and reset Smoothed P to Initial Offset. This SHOULD be done every 10 minutes. In so doing, IPv6 and IPv4 are tried again so that if the IPv6 is working again, it will quickly be preferred again.

5.3. Connecting to a Server

The steps when connecting to a server are as follows:

1. query DNS using getaddrinfo(). This will return addresses sorted by the host's default address selection ordering *[RFC3484]*, its updates, or the address selection as chosen by the network administrator [I-D.ietf-6man-addr-select-opt].

2. If this returns both an IPv6 and IPv4 address, continue processing to the next stop. Otherwise, Happy Eyeballs processing stops here.

3. Of the addresses returned in step (1), look up the first IPv6 address and first IPv4 address in the Happy Eyeballs exception

cache. Matching entries in the exception cache influence the P value for this connection attempt by setting P to the sum of Smoothed_P and of the P values from the matching IPv6 entry (if it exists) and the matching IPv4 entry (if it exists).

4. If P>=0, initiate a connection attempt using the first IPv6 address returned by step (1). If that connection has not completed after P milliseconds, initiate a connection attempt using IPv4.

5. If P<=0, initiate a connection attempt using the first IPv4 address returned by getaddrinfo. If that connection has not completed after absolute value(P) milliseconds, initiate a connection attempt using IPv6.

6. If neither connection has completed after MAXWAIT seconds, repeat the procedure at step (3) until the addresses are exhausted.

After performing the above steps, there will be no connection at all or one connection will complete first. If no connection was successful, it should be treated as a failure for both IPv6 and IPv4.

5.4. Adjusting Address Family Preferences

If the preferred address family completed first, Smoothed P is adjusted towards that address family. If the non-preferred address family completed, we wait an additional Tolerance Interval milliseconds for the preferred address family to complete. If the expected address family succeeded, we increment the absolute value of the Smoothed P; if the expected address family failed - we create an exception entry that will make an adjustment to the future value of P for the attempt on this pair in the direction opposite to the current sign of Smoothed P. The table below summarizes the adjustments:

	Connection completed within Tolerance Interval		
	v6 and v4 ok	v6 ok, v4 failed	v6 failed, v4 ok
P > 0	SP=SP+10	SP=SP+10	SP=SP/2 or cache
P < 0	SP=SP+10	SP=SP/2 or cache	SP=SP-10
P = 0	SP=big(10,IH)	SP=IH	SP=(-IH)

The the above table is described in textual form:

*If P > 0 (indicating IPv6 is preferred over IPv4):

-and both the IPv6 and IPv4 connection attempts completed within the Tolerance Interval, it means the IPv6 preference

was accurate or we should gently prefer IPv6, so Smoothed P is increased by 10 milliseconds (Smoothed P = Smoothed P + 10).

- If the IPv6 connection completed but the IPv4 connection failed within the tolerance interval, it means future connections should prefer IPv6, so Smoothed P is increased by 10 milliseconds (Smoothed_P = Smoothed_P + 10).

- If the IPv6 connection failed but the IPv4 connection completed within the tolerance interval, it means the IPv6 preference is inaccurate. If no exception cache entry exists for the IPv6 and IPv4 prefixes, the entries are created and their P value set to to the connection setup time * -1, and Smoothed P is halved and rounded towards zero (Smoothed_P = Smoothed_P * 0.5). If an exception cache entry already existed, its P value is doubled and Smoothed_P is not adjusted.

*If P < 0 (indicating IPv4 is preferred over IPv6):

- and both the IPv6 and IPv4 connection attempts completed within the tolerance interval, we should gently prefer IPv6, so Smoothed P is increased by 10 milliseconds (Smoothed_P = Smoothed_P + 10).

- If the IPv6 connection completed but the IPv4 connection failed within within the tolerance interval, it means the IPv4 preference is inaccurate. If no exception cache entry exists for the IPv6 and IPv4 prefixes, they are created and their P values set to the connection setup time and Smoothed P is halved and rounded towards 0 (Smoothed_P = Smoothed_P * 0.5). If an exception cahe entry already existed, its P value is doubled and Smoothed_P is not adjusted.

- If the IPv4 connection completed but the IPv6 connection failed within the tolerance interval, it means future connections should prefer IPv4, so Smoothed P is decreased by 10 milliseconds (Smoothed_P = Smoothed_P - 10).

*If P = 0 (indicating IPv4 and IPv6 are equally preferred):

- and both the IPv6 and IPv4 connection attempts completed within the tolerance interval, we should prefer IPv6 significantly, so Smoothed P is set to the larger of Initial Headstart or 10 (Smoothed_P = larger(Initial Headstart, 10)).

- if the IPv6 connection completed but the IPv4 connection failed within the Tolerance Interval, it means we need to prefer IPv6, so Smoothed P is increased by 10 (Smoothed_P = Smoothed_P + 10).

-if the IPv4 connection completed but the IPv6 connection failed within the Tolerance Interval, it means we need to prefer IPv4, so P is decreased by 10 (Smoothed_P = Smoothed_P - 10).

5.5. Exception Cache

An exception cache is maintained of IPv6 prefixes and IPv4 prefixes, which are exceptions to the Smoothed P value at the time a connection was made. For IPv6 prefixes, the default prefix length is 64. For IPv4, the default prefix length is /32.
The exception cache MAY be a fixed size, removing entires using a least-frequently used algorithm. This works because the network path is likely to change over time (thus old entries aren't valuable anyway), and if an entry does not exist the Smoothed P value will still provide some avoidance of user-noticable connection setup delay.

6. Implementation Details: SRV

*[[Editor's Note: SRV processing needs to be incorporated into the above section, rather than described separately. This will be done in a future update to this document.]]

For the purposes of this section, "client" is defined as the entity initiating the connection.
For protocols which support DNS SRV [RFC2782], the client performs the IN SRV query (e.g. IN SRV _xmpp-client._tcp.example.com) as normal. The client MUST perform the following steps:

1. Sort all SRV records according to priority (lowest priority first)

2. Process all of the SRV targets of the same priority with a weight greater than 0:

   a. Perform A/AAAA queries for each SRV target in parallel, as described in the A/AAAA processing section

   b. Connect to the IPv4/IPv6 addresses

   c. If at least one connection succeeds, stop processing SRV records

3. If there is no connection, process all of the SRV targets of the same priority with a weight of 0, as per steps 2.1 through 2.3 above

4. Repeat steps 2.1 through 2.3 for the next priority, until a connection is established or all SRV records have been exhausted

5. If there is still no connection, fallback to using the domain (e.g., example.com), following steps 2.1 through 2.3 above

7. Additional Considerations

This section discusses considerations and requirements that are common to new technology deployment.

7.1. Additional Network and Host Traffic

Additional network traffic and additional server load is created due to the recommendations in this document. This additional load is mitigated by the P value, especially the exception cache P value.
The procedures described in this document retain a quality user experience while transitioning from IPv4-only to dual stack, while still giving IPv6 a slight preference over IPv4 (in order to remove load from IPv4 networks, most importantly to reduce the load on IPv4 network address translators). The improvement in the user experience benefits the user to only a small detriment of the network, DNS server, and server that are serving the user.

7.2. Abandon Non-Winning Connections

It is RECOMMENDED that the non-winning connections be abandoned, even though they could -- in some cases -- be put to reasonable use. To take HTTP as an example, the design of some sites can break because of HTTP cookies that incorporate the client's IP address, require all connections be from the same IP address. If some connections from the same client are arriving from different IP addresses, such applications will break. It is also important to abandon connections to avoid consuming server resources (file descriptors, TCP control blocks) or middlebox resources (e.g., NAPT). Using the non-winning connection can also interfere with the browser's Same Origin Policy (see Section 7.8).

7.3. Determining Address Type

For some transitional technologies such as a dual-stack host, it is easy for the application to recognize the native IPv6 address (learned via a AAAA query) and the native IPv4 address (learned via an A query). While IPv6/IPv4 translation makes that difficult, fortunately IPv6/IPv4 translators are not deployed on networks with dual stack clients, which is the scope of this document.

7.4. Debugging and Troubleshooting

This mechanism is aimed at ensuring a reliable user experience regardless of connectivity problems affecting any single transport. However, this naturally means that applications employing these techniques are by default less useful for diagnosing issues with any particular transport. To assist in that regard, the applications

implementing the proposal in this document SHOULD also provide a mechanism to revert the behavior to that of a default provided by the operating system - the [RFC3484].

7.5. DNS Behavior

Unique to DNS AAAA queries are the problems described in [RFC4074] which, if they still persist, require applications to perform an A query before the AAAA query.

*[[Editor's Note 03: It is believed these defective DNS servers have long since been upgraded. If so, we can remove this section.]]

7.6. Middlebox Issues

Some devices are known to exhibit what amounts to a bug, when the A and AAAA requests are sent back-to-back over the same 4-tuple, and drop one of the requests or replies [DNS-middlebox]. However, in some cases fixing this behaviour may not be possible either due to the architectural limitations or due to the administrative constraints (location of the faulty device is unknown to the end hosts or not controlled by the end hosts). The algorithm described in this draft, in the case of this erroneous behaviour will eventually pace the queries such that this middlebox issue is avoided. The algorithm described in this draft also avoids calling the operating system's getaddrinfo() with "any", which should prevent the operating system from sending the A and AAAA queries from the same port.
For the large part, these issues with simultaneous DNS requests are believed to be fixed.

7.7. Multiple Interfaces

Interaction of the suggestions in this document with multiple interfaces, and interaction with the MIF working group, is for further study ([I-D.chen-mif-happy-eyeballs-extension] is devoted to this).

7.8. Interaction with Same Origin Policy

Web browsers implement same origin policy (SOP, [sop], [I-D.abarth-origin]), which causes subsequent connections to the same hostname to go to the same IPv4 (or IPv6) address as the previous successful connection. This is done to prevent certain types of attacks.
The same-origin policy harms user-visible responsiveness if a new connection fails (e.g., due to a transient event such as router failure or load balancer failure). While it is tempting to use Happy Eyeballs to maintain responsiveness, web browsers MUST NOT change their same origin policy because of Happy Eyeballs

8. Content Provider Recommendations

Content providers SHOULD provide both AAAA and A records for servers using the same DNS name for both IPv4 and IPv6.

9. Security Considerations

[[Placeholder.]]
See Section 7.2 and Section 7.8.

10. Acknowledgements

The mechanism described in this paper was inspired by Stuart Cheshire's discussion at the IAB Plenary at IETF72, the author's understanding of Safari's operation with SRV records, Interactive Connectivity Establishment (ICE *[RFC5245]*), and the current IPv4/IPv6 behavior of SMTP mail transfer agents.
Thanks to Fred Baker, Jeff Kinzli, Christian Kuhtz, and Iljitsch van Beijnum for fostering the creation of this document.
Thanks to Scott Brim, Rick Jones, Stig Venaas, Erik Kline, Bjoern Zeeb, Matt Miller, Dave Thaler, and Dmitry Anipko for providing feedback on the document.
Thanks to Javier Ubillos, Simon Perreault and Mark Andrews for the active feedback and the experimental work on the independent practical implementations that they created.
Also the authors would like to thank the following individuals who participated in various email discussions on this topic: Mohacsi Janos, Pekka Savola, Ted Lemon, Carlos Martinez-Cagnazzo, Simon Perreault, Jack Bates, Jeroen Massar, Fred Baker, Javier Ubillos, Teemu Savolainen, Scott Brim, Erik Kline, Cameron Byrne, Daniel Roesen, Guillaume Leclanche, Mark Smith, Gert Doering, Martin Millnert, Tim Durack, Matthew Palmer.

11. IANA Considerations

This document has no IANA actions.

12. References

12.1. Normative References

[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
[RFC3484]	Draves, R., "Default Address Selection for Internet Protocol version 6 (IPv6)", RFC 3484, February 2003.
[RFC2782]	Gulbrandsen, A., Vixie, P. and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

12.2. Informational References

[RFC1671]	Carpenter, B., "IPng White Paper on Transition and Other Considerations", RFC 1671, August 1994.
[RFC4074]	Morishita, Y. and T. Jinmei, "Common Misbehavior Against DNS Queries for IPv6 Addresses", RFC 4074, May 2005.
[RFC5245]	Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/ Answer Protocols", RFC 5245, April 2010.
[RFC6059]	Krishnan, S. and G. Daley, "Simple Procedures for Detecting Network Attachment in IPv6", RFC 6059, November 2010.
[RFC4436]	Aboba, B., Carlson, J. and S. Cheshire, "Detecting Network Attachment in IPv4 (DNAv4)", RFC 4436, March 2006.
[I-D.chen-mif-happy-eyeballs-extension]	Chen, G, Williams, C, Wing, D and A Yourtchenko, "Happy Eyeballs Extension for Multiple Interfaces", Internet-Draft draft-chen-mif-happy-eyeballs-extension-03, October 2011.
[I-D.ietf-6man-addr-select-opt]	Matsumoto, A, Fujisaki, T, Kato, J and T Chown, "Distributing Address Selection Policy using DHCPv6", Internet-Draft draft-ietf-6man-addr-select-opt-01, June 2011.
[whitelist]	Google, , "Google IPv6 DNS Whitelist", January 2009.
[DNS-middlebox]	Various, , "DNS middlebox behavior with multiple queries over same source port", June 2009.
[cx-osx]	Adium, , "AIHostReachabilityMonitor", June 2009.
[cx-win]	Microsoft, , "NetworkChange.NetworkAvailabilityChanged Event", June 2009.
[Perreault]	Perreault, S, "Happy Eyeballs in Erlang", February 2011.
[Andrews]	Andrews, M, "How to connect to a multi-homed server over TCP", January 2011.
[I-D.abarth-origin]	Barth, A, "The Web Origin Concept", Internet-Draft draft-abarth-origin-09, November 2010.
[sop]	W3C, , "Same Origin Policy", January 2010.
[Experiences]	Savolainen, T., Miettinen, N., Veikkolainen, S., Chown, T. and J. Morse, "Experiences of host behavior in broken IPv6 networks", March 2011.

Appendix A. Changes

Appendix A.1. changes from -01 to -02

*Now honors host's address preference (RFC3484 and friends)

*No longer requires thread-safe DNS library. It uses getaddrinfo()

*No longer describes threading.

*IPv6 is given a 200ms head start (Initial Headstart variable).

*If the IPv6 and IPv4 connection attempts were made at nearly the same time, wait Tolerance Interval milliseconds for both to complete before deciding which one wins.

*Renamed "global P" to "Smoothed P", and better described how it is calculated.

*introduced the exception cache. This contains the set of networks that only work with IPv4 (or only with IPv6), so that subsequent connection attempts use that address family without them causing serious affect to Smoothed P.

*encourages that every 10 minutes the exception cache and Smoothed P be reset. This allows IPv6 to be attempted again, so we don't get 'stuck' on IPv4.

*If we didn't get both A and AAAA, abandon all Happy Eyeballs processing (thanks to Simon Perreault).

*added discussion of Same Origin Policy

*Removed discussion of NAT-PT and address learning; those are only used with IPv6-only hosts whereas this document is about dual-stack hosts contacting dual-stack servers.

Appendix A.2. changes from -00 to -01

*added SRV section (thanks to Matt Miller)

Authors' Addresses

Dan Wing Wing Cisco Systems, Inc. 170 West Tasman Drive
San Jose, CA 95134 USA EMail: dwing@cisco.com

Andrew Yourtchenko Yourtchenko Cisco Systems, Inc. De Kleetlaan, 7
San Jose, Diegem B-1831 Belgium EMail: ayourtch@cisco.com