Internet Engineering Task Force (IETF)
Request for Comments: 6555
Category: Standards Track
ISSN: 2070-1721

D. Wing
A. Yourtchenko
Cisco

April 2012

Happy Eyeballs: Success with Dual-Stack Hosts

Abstract

When a server's IPv4 path and protocol are working, but the server's IPv6 path and protocol are not working, a dual-stack client application experiences significant connection delay compared to an IPv4-only client. This is undesirable because it causes the dual-stack client to have a worse user experience. This document specifies requirements for algorithms that reduce this user-visible delay and provides an algorithm.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6555.

Copyright Notice

Table of Contents

1. Introduction

In order to use applications over IPv6, it is necessary that users enjoy nearly identical performance as compared to IPv4. A combination of today's applications, IPv6 tunneling, 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 does not scale well (to the number of DNS servers worldwide or the number of content providers worldwide) and does react to intermittent network path outages.

Instead, applications reduce connection setup delays themselves, by more aggressively making connections on IPv6 and IPv4. There are a variety of algorithms that can be envisioned. This document specifies requirements for any such algorithm, with the goals that the network and servers not be inordinately harmed with a simple doubling of traffic on IPv6 and IPv4 and the host's address preference be honored (e.g., [RFC3484]).

1.1. Additional Network and Host Traffic

Additional network traffic and additional server load is created due to the recommendations in this document, especially when connections to the preferred address family (usually IPv6) are not completing quickly.

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 and, most importantly, to reduce the load on IPv4 network address translators). The user experience is improved to the slight detriment of the network, DNS server, and server that are serving the user.

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

The basis of the IPv6/IPv4 selection problem was first described in 1994 in [RFC1671]:

> 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.)

As discussed in more detail in Section 3.1, it is important that the same hostname be used for IPv4 and IPv6.

As discussed in more detail in Section 3.2, IPv6 connectivity is broken to specific prefixes or specific hosts or is slower than native IPv4 connectivity.

The mechanism described in this document is directly applicable to connection-oriented transports (e.g., TCP, SCTP), which is the scope of this document. For connectionless transport protocols (e.g., UDP), a similar mechanism can be used if the application has request/response semantics (e.g., as done by Interactive Connectivity Establishment (ICE) to select a working IPv6 or IPv4 media path [RFC6157]).

3.1. Hostnames

Hostnames are often used between users to exchange pointers to content -- such as on social networks, email, instant messaging, or other systems. Using separate namespaces (e.g., "ipv6.example.com"), which are only accessible with certain client technology (e.g., an IPv6 client) and dependencies (e.g., a working IPv6 path), causes namespace fragmentation and reduces the ability for users to share hostnames. It also complicates printed material that includes the hostname.

The algorithm described in this document allows production hostnames to avoid these problematic references to IPv4 or IPv6.

3.2. Delay When IPv6 Is Not Accessible

When IPv6 connectivity is impaired, today’s IPv6-capable applications (e.g., web browsers, email clients, instant messaging clients) incur many seconds of delay before falling back to IPv4. This delays overall application operation, including harming 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 following diagram shows this behavior.

The algorithm described in this document allows clients to connect to servers without significant delay, even if a path or the server is slow or down.

```
        DNS Server                 Client                 Server
             |                        |                       |
      1.     |<--www.example.com A?-----|                     |
      2.     |<--www.example.com AAAA?--|                     |
      3.     |---192.0.2.1------------->|                     |
      4.     |---2001:db8::1----------->|                     |
      5.     |                          |                     |
      6.     |                          |==TCP SYN, IPv6===>X |
      7.     |                          |==TCP SYN, IPv6===>X |
      8.     |                          |==TCP SYN, IPv6===>X |
      9.     |                          |                     |
      10.    |                          |--TCP SYN, IPv4------->|
      11.    |                          |<-TCP SYN+ACK, IPv4----|
      12.    |                          |--TCP ACK, IPv4------->|
```

Figure 1: Existing Behavior Message Flow

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. Algorithm Requirements

A "Happy Eyeballs" algorithm has two primary goals:

1. Provides fast connection for users, by quickly attempting to connect using IPv6 and (if that connection attempt is not quickly successful) to connect using IPv4.

2. Avoids thrashing the network, by not (always) making simultaneous connection attempts on both IPv6 and IPv4.

The basic idea is depicted in the following diagram:

```
           DNS Server                  Client                  Server
               |                          |                       |
         1.    |<--www.example.com A?-----|                       |
         2.    |<--www.example.com AAAA?--|                       |
         3.    |---192.0.2.1------------->|                       |
         4.    |---2001:db8::1----------->|                       |
         5.    |                          |                       |
         6.    |                          |==TCP SYN, IPv6===>X   |
         7.    |                          |--TCP SYN, IPv4------->|
         8.    |                          |<-TCP SYN+ACK, IPv4----|
         9.    |                          |--TCP ACK, IPv4------->|
        10.    |                          |==TCP SYN, IPv6===>X   |
```

Figure 2: Happy Eyeballs Flow 1, IPv6 Broken

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 whether connections to the host’s IPv6 or IPv4 address were successful. The client MUST cache information regarding the outcome of each connection attempt, and it uses that information to avoid thrashing the network with subsequent attempts. In the example above, the cache indicates that the IPv6 connection attempt failed, and therefore the system will prefer IPv4 instead. Cache entries should be flushed when their age exceeds a system-defined maximum on the order of 10 minutes.

```
           DNS Server                  Client                  Server
               |                          |                        |
         1.    |<--www.example.com A?-----|                        |
         2.    |<--www.example.com AAAA?--|                        |
         3.    |---192.0.2.1------------->|                        |
         4.    |---2001:db8::1----------->|                        |
         5.    |                          |                        |
         6.    |                          |==TCP SYN, IPv6=======>|
         7.    |                          |--TCP SYN, IPv4------->|
         8.    |                          |<=TCP SYN+ACK, IPv6====|
         9.    |                          |<-TCP SYN+ACK, IPv4----|
        10.    |                          |==TCP ACK, IPv6=======>|
        11.    |                          |--TCP ACK, IPv4------->|
        12.    |                          |--TCP RST, IPv4------->|
```

Figure 3: Happy Eyeballs Flow 2, IPv6 Working

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

Any Happy Eyeballs algorithm will persist in products for as long as the client host is dual-stacked, which will persist as long as there are IPv4-only servers on the Internet -- the so-called "long tail". Over time, as most content is available via IPv6, the amount of IPv4 traffic will decrease. This means that the IPv4 infrastructure will, over time, be sized to accommodate that decreased (and decreasing) amount of traffic. It is critical that a Happy Eyeballs algorithm not cause a surge of unnecessary traffic on that IPv4 infrastructure. To meet that goal, compliant Happy Eyeballs algorithms must adhere to the requirements in this section.

4.1. Delay IPv4

The transition to IPv6 is likely to produce a mix of different hosts within a subnetwork -- hosts that are IPv4-only, hosts that are IPv6-only (e.g., sensors), and dual-stack hosts. This mix of hosts will exist both within an administrative domain (a single home, enterprise, hotel, or coffee shop) and between administrative domains. For example, a single home might have an IPv4-only television in one room and a dual-stack television in another room. As another example, another subscriber might have hosts that are all capable of dual-stack operation.

Due to IPv4 exhaustion, it is likely that a subscriber’s hosts (both IPv4-only hosts and dual-stack hosts) will be sharing an IPv4 address with other subscribers. The dual-stack hosts have an advantage: they can utilize IPv6 or IPv4, which means they can utilize the technique described in this document. The IPv4-only hosts have a disadvantage:

they can only utilize IPv4. If all hosts (dual-stack and IPv4-only) are using IPv4, there is additional contention for the shared IPv4 address. The IPv4-only hosts cannot avoid that contention (as they can only use IPv4), while the dual-stack hosts can avoid it by using IPv6.

As dual-stack hosts proliferate and content becomes available over IPv6, there will be proportionally less IPv4 traffic. This is true especially for dual-stack hosts that do not implement Happy Eyeballs, because those dual-stack hosts have a very strong preference to use IPv6 (with timeouts in the tens of seconds before they will attempt to use IPv4).

When deploying IPv6, both content providers and Internet Service Providers (who supply mechanisms for IPv4 address sharing such as Carrier-Grade NAT (CGN)) will want to reduce their investment in IPv4 equipment -- load-balancers, peering links, and address sharing devices. If a Happy Eyeballs implementation treats IPv6 and IPv4 equally by connecting to whichever address family is fastest, it will contribute to load on IPv4. This load impacts IPv4-only devices (by increasing contention of IPv4 address sharing and increasing load on IPv4 load-balancers). Because of this, ISPs and content providers will find it impossible to reduce their investment in IPv4 equipment. This means that costs to migrate to IPv6 are increased because the investment in IPv4 cannot be reduced. Furthermore, using only a metric that measures the connection speed ignores the benefits that IPv6 brings when compared with IPv4 address sharing, such as improved geo-location [RFC6269] and the lack of fate-sharing due to traversing a large translator.

Thus, to avoid harming IPv4-only hosts, implementations MUST prefer the first IP address family returned by the host’s address preference policy, unless implementing a stateful algorithm described in Section 4.2. This usually means giving preference to IPv6 over IPv4, although that preference can be overridden by user configuration or by network configuration [ADDR-SELECT]. If the host’s policy is unknown or not attainable, implementations MUST prefer IPv6 over IPv4.

4.2. Stateful Behavior When IPv6 Fails

Some Happy Eyeballs algorithms are stateful -- that is, the algorithm will remember that IPv6 always fails, or that IPv6 to certain prefixes always fails, and so on. This section describes such algorithms. Stateless algorithms, which do not remember the success/failure of previous connections, are not discussed in this section.

After making a connection attempt on the preferred address family (e.g., IPv6) and failing to establish a connection within a certain time period (see Section 5.5), a Happy Eyeballs implementation will decide to initiate a second connection attempt using the same address family or the other address family.

Such an implementation MAY make subsequent connection attempts (to the same host or to other hosts) on the successful address family (e.g., IPv4). So long as new connections are being attempted by the host, such an implementation MUST occasionally make connection attempts using the host's preferred address family, as it may have become functional again, and it SHOULD do so every 10 minutes. The 10-minute delay before retrying a failed address family avoids the simple doubling of connection attempts on both IPv6 and IPv4. Implementation note: this can be achieved by flushing Happy Eyeballs state every 10 minutes, which does not significantly harm the application's subsequent connection setup time. If connections using the preferred address family are again successful, the preferred address family SHOULD be used for subsequent connections. Because this implementation is stateful, it MAY track connection success (or failure) based on IPv6 or IPv4 prefix (e.g., connections to the same prefix assigned to the interface are successful whereas connections to other prefixes are failing).

4.3. Reset on Network (Re-)Initialization

Because every network has different characteristics (e.g., working or broken IPv6 or IPv4 connectivity), a Happy Eyeballs algorithm SHOULD re-initialize when the interface is connected to a new network. Interfaces can determine network (re-)initialization by a variety of mechanisms (e.g., Detecting Network Attachment in IPv4 (DNAv4) [RFC4436], DNAv6 [RFC6059]).

If the client application is a web browser, see also Section 5.6.

4.4. 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.

> Justification: This reduces the load on the server (file descriptors, TCP control blocks) and stateful middleboxes (NAT and firewalls). Also, if the abandoned connection is IPv4, this reduces IPv4 address sharing contention.
>
> HTTP: The design of some sites can break because of HTTP cookies that incorporate the client's IP address and require all connections be from the same IP address. If some connections from

the same client are arriving from different IP addresses (or worse, different IP address families), such applications will break. Additionally, for HTTP, using the non-winning connection can interfere with the browser’s same-origin policy (see Section 5.6).

5. Additional Considerations

This section discusses considerations related to Happy Eyeballs.

5.1. 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, IPv6/IPv4 translators do not need to be deployed on networks with dual-stack clients because dual-stack clients can use their native IP address family.

5.2. 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 a particular address family. To assist in that regard, the implementations MAY also provide a mechanism to disable their Happy Eyeballs behavior via a user setting, and to provide data useful for debugging (e.g., a log or way to review current preferences).

5.3. Three or More Interfaces

A dual-stack host normally has two logical interfaces: an IPv6 interface and an IPv4 interface. However, a dual-stack host might have more than two logical interfaces because of a VPN (where a third interface is the tunnel address, often assigned by the remote corporate network), because of multiple physical interfaces such as wired and wireless Ethernet, because the host belongs to multiple VLANs, or other reasons. The interaction of Happy Eyeballs with more than two logical interfaces is for further study.

5.4. A and AAAA Resource Records

It is possible that a DNS query for an A or AAAA resource record will return more than one A or AAAA address. When this occurs, it is RECOMMENDED that a Happy Eyeballs implementation order the responses following the host’s address preference policy and then try the first address. If that fails after a certain time (see Section 5.5), the next address SHOULD be the IPv4 address.

If that fails to connect after a certain time (see Section 5.5), a Happy Eyeballs implementation SHOULD try the other addresses returned; the order of these connection attempts is not important.

On the Internet today, servers commonly have multiple A records to provide load-balancing across their servers. This same technique would be useful for AAAA records, as well. However, if multiple AAAA records are returned to a client that is not using Happy Eyeballs and that has broken IPv6 connectivity, it will further increase the delay to fall back to IPv4. Thus, web site operators with native IPv6 connectivity SHOULD NOT offer multiple AAAA records. If Happy Eyeballs is widely deployed in the future, this recommendation might be revisited.

5.5. Connection Timeout

The primary purpose of Happy Eyeballs is to reduce the wait time for a dual-stack connection to complete, especially when the IPv6 path is broken and IPv6 is preferred. Aggressive timeouts (on the order of tens of milliseconds) achieve this goal, but at the cost of network traffic. This network traffic may be billable on certain networks, will create state on some middleboxes (e.g., firewalls, intrusion detection systems, NATs), and will consume ports if IPv4 addresses are shared. For these reasons, it is RECOMMENDED that connection attempts be paced to give connections a chance to complete. It is RECOMMENDED that connection attempts be paced 150-250 ms apart to balance human factors against network load. Stateful algorithms are expected to be more aggressive (that is, make connection attempts closer together), as stateful algorithms maintain an estimate of the expected connection completion time.

5.6. Interaction with Same-Origin Policy

Web browsers implement a same-origin policy [RFC6454] that 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, as that would create an additional security exposure.

5.7. Implementation Strategies

The simplest venue for the implementation of Happy Eyeballs is within the application itself. The algorithm specified in this document is relatively simple to implement and would require no specific support from the operating system beyond the commonly available APIs that provide transport service. It could also be added to applications by way of a specific Happy Eyeballs API, replacing or augmenting the transport service APIs.

To improve the IPv6 connectivity experience for legacy applications (e.g., applications that simply rely on the operating system’s address preference order), operating systems may consider more sophisticated approaches. These can include changing default address selection sorting [RFC3484] based on configuration received from the network, or observing connection failures to IPv6 and IPV4 destinations.

6. Example Algorithm

What follows is the algorithm implemented in Google Chrome and Mozilla Firefox.

1. Call getaddinfo(), which returns a list of IP addresses sorted by the host’s address preference policy.

2. Initiate a connection attempt with the first address in that list (e.g., IPv6).

3. If that connection does not complete within a short period of time (Firefox and Chrome use 300 ms), initiate a connection attempt with the first address belonging to the other address family (e.g., IPv4).

4. The first connection that is established is used. The other connection is discarded.

If an algorithm were to cache connection success/failure, the caching would occur after step 4 determined which connection was successful.

Other example algorithms include [Perreault] and [Andrews].

7. Security Considerations

See Sections 4.4 and 5.6.

8. Acknowledgements

The mechanism described in this paper was inspired by Stuart Cheshire’s discussion at the IAB Plenary at IETF 72, the author’s understanding of Safari’s operation with SRV records, ICE [RFC5245], the current IPv4/IPv6 behavior of SMTP mail transfer agents, and the implementation of Happy Eyeballs in Google Chrome and Mozilla Firefox.

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, Dmitry Anipko, Brian Carpenter, and David Crocker for their feedback.

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, and Matthew Palmer.

9. References

9.1. 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.

9.2. Informative References

[ADDR-SELECT] Matsumoto, A., Fujisaki, T., Kato, J., and T. Chown, "Distributing Address Selection Policy using DHCPv6", Work in Progress, February 2012.

[Andrews] Andrews, M., "How to connect to a multi-homed server over TCP", January 2011, <http://www.isc.org/community/blog/201101/how-to-connect-to-a-multi-homed-server-over-tcp>.

[Experiences] Savolainen, T., Miettinen, N., Veikkolainen, S., Chown, T., and J. Morse, "Experiences of host behavior in broken IPv6 networks", March 2011, <http://www.ietf.org/proceedings/80/slides/v6ops-12.pdf>.

[Perreault] Perreault, S., "Happy Eyeballs in Erlang", February 2011, <http://www.viagenie.ca/news/index.html#happy_eyeballs_erlang>.

[RFC1671] Carpenter, B., "IPng White Paper on Transition and Other Considerations", RFC 1671, August 1994.

[RFC4436] Aboba, B., Carlson, J., and S. Cheshire, "Detecting Network Attachment in IPv4 (DNAv4)", RFC 4436, March 2006.

[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.

[RFC6157] Camarillo, G., El Malki, K., and V. Gurbani, "IPv6 Transition in the Session Initiation Protocol (SIP)", RFC 6157, April 2011.

[RFC6269] Ford, M., Boucadair, M., Durand, A., Levis, P., and P. Roberts, "Issues with IP Address Sharing", RFC 6269, June 2011.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, December 2011.

[WHITELIST] Google, "Google over IPv6", <http://www.google.com/intl/en/ipv6>.

Authors' Addresses

Dan Wing
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134
USA

EMail: dwing@cisco.com

Andrew Yourtchenko
Cisco Systems, Inc.
De Kleetlaan, 7
Diegem B-1831
Belgium

EMail: ayourtch@cisco.com