Domain Name System Operations
Internet-Draft
Intended status: Standards Track
Expires: 5 December 2026

G. Akiwate
P. Flack
S. Cheshire
Apple Inc.

3 June 2026

Optimistic DNS
draft-gakiwate-dnsop-optimistic-dns-00

Abstract

DNS lookups introduce user-visible delay, particularly when cached records have expired and must be refreshed from the network. This document describes Optimistic DNS, a client-side stub resolver mechanism that immediately returns expired cached DNS records to applications while simultaneously refreshing them with a network query. The application receives an answer in microseconds rather than milliseconds, and if the data has changed receives an updated answer shortly thereafter. Optimistic DNS is complementary to RFC 8767, which addresses serving stale data at recursive resolvers. This document focuses exclusively on client-side stub resolver behavior, including explicit signaling from the application to inform the stub resolver that the application is able to handle old and possibly incorrect information.

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://gakiwate.github.io/draft-gakiwate-dnsop-optimistic-dns/draft-gakiwate-dnsop-optimistic-dns.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-gakiwate-dnsop-optimistic-dns/.

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

Source for this draft and an issue tracker can be found at https://github.com/gakiwate/draft-gakiwate-dnsop-optimistic-dns.

Status of This Memo

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

Copyright Notice

Table of Contents

1. Introduction

In the early days of the Internet, computers and network connections were much slower than they are now. Delays of a few tens or hundreds of milliseconds used to be insignificant compared to the overall time taken to load a web page. Today, increases in computing power and network throughput have dramatically reduced previous major causes of slowness. To continue progress on improving user experience, we need to identify and reduce the remaining sources of delay, which -- as other aspects of computer and network performance improve -- now constitute a larger proportion of the overall web page load time. Computer processing speed and network throughput continue to increase, but the speed of light isn't changing.

When a user views a website, the first step is typically a DNS lookup to translate the hostname into an IP address. If the device's DNS cache contains a record for that hostname, the answer is returned almost instantly and the user perceives no delay. But if the record's TTL has expired by even a single second, the device must perform a fresh DNS lookup. On a wired connection this might take 50 to 200 milliseconds. On a cellular connection, particularly at the edge of coverage, it can take several seconds.

The user experiences this as a brief but noticeable pause. The website appears to hang. The user wonders if something is wrong. And in most cases, the expired record still contained the correct IP address and the website has not moved to a different server. The user waited for confirmation of something the device already knew.

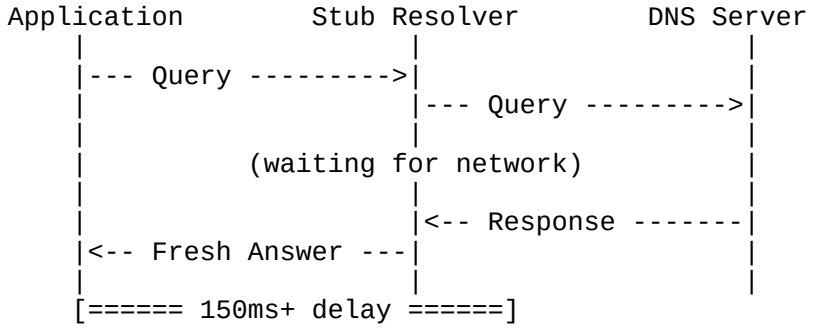
Consider a record for www.example.com with a TTL of 60 seconds. At time T=0 the record is fetched and cached. For the next 60 seconds, any application that asks for www.example.com gets an instant answer. At time T=61, the record has expired. The very next lookup must go to the network. From the user's perspective, the transition from

"instant" to "slow" is a sudden large spike. The record was valid for 60 seconds, giving cached results in microseconds, then invalid for the fraction of a second it took to refresh it, resulting in a delay of tens or hundreds of milliseconds, then valid again, with the delay returning to microseconds. Yet that intermittent slowness will be the experience that the user notices when the rest of their web browsing is usually consistently fast.

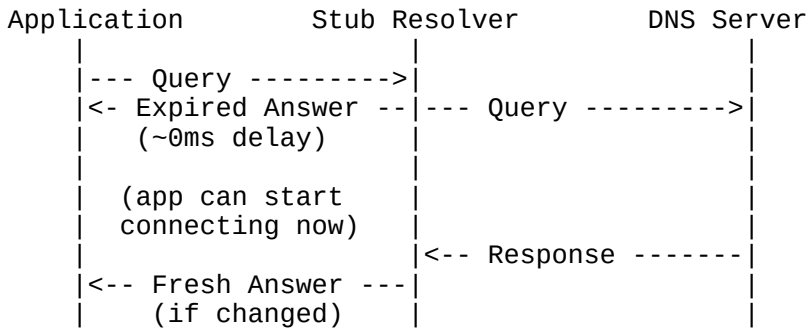
This document describes Optimistic DNS, a stub resolver mechanism that addresses this problem. When the stub resolver has expired cached records that match a query, it returns those expired records to the application immediately while simultaneously issuing a fresh network query in the background. The application first receives the expired (but likely still correct) records, and then shortly afterwards, the authoritative fresh records, if different. Using the (expired, but probably correct) address from the cache, the networking library code that the application is using will start connecting. While connecting, the networking library code MUST continue to pay attention to asynchronous notifications of new addresses as they are learned, and the networking library code MUST react gracefully if, occasionally, some of the candidate addresses do, in fact, prove to be stale and incorrect (Section 5).

The following diagram illustrates the timing difference between conventional DNS resolution and Optimistic DNS:

Conventional DNS (after cache expiry):



Optimistic DNS (after cache expiry):



Optimistic DNS is complementary to Serving Stale Data to Improve DNS Resiliency [RFC8767], which allows recursive resolvers to serve stale data during upstream failures. The two mechanisms differ in their focus. As reflected in the document title, the specification for serving stale data from recursive resolvers was focused on improving resiliency in situations where the authoritative servers are down or unreachable. Optimistic DNS is focused on enhancements to the stub resolver on the end-user's device, to reduce delays even in cases where the authoritative servers are functioning perfectly well. Both can be deployed simultaneously for layered staleness tolerance.

This document describes only the client-side behavior. Optimistic DNS does not define any new DNS wire-protocol messages, opcodes, or EDNS options. The signaling between the stub resolver and the application is purely a local API matter.

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.

Stub Resolver. A DNS resolver that operates on the end-user device. It maintains a local cache of DNS records and forwards queries to configured recursive resolvers when cached answers are unavailable or expired. This is the component that implements Optimistic DNS.

Expired Record. A cached DNS record whose TTL has reached zero. Under conventional DNS caching rules, such a record would be purged from the cache or ignored when answering queries.

Optimistic Answer. An expired cached record that is returned to an

application in response to a query, before the stub resolver has confirmed whether the record's data is still current.

Fresh Answer. A DNS record received from the network in response to a query, as opposed to a record served from the local cache. Fresh answers reflect the data known by the upstream recursive resolver, which may be more recent than the data in the local cache.

Asynchronous DNS Resolution. A DNS resolution model where the application initiates a query and receives results through callbacks or event notifications as they become available, rather than blocking until a single atomic set of answers is returned all at once (Section 5.1).

Happy Eyeballs. A client-side connection establishment algorithm [IETF72] [RFC6555] [RFC8305] [HEv3] that races connection attempts across multiple addresses and address families, using whichever connection succeeds first (Section 5.2).

3. Problem Statement

The DNS TTL mechanism creates an inherent tension between freshness and performance. When a record is cached and its TTL has not expired, lookups are essentially free and the answer is returned from local memory in microseconds. The moment the TTL expires, the cost jumps to a full network round trip. This is not a graceful degradation. It is a dramatic spike.

This problem is compounded by several factors:

Short TTLs. Many content delivery networks and cloud services use TTLs of 60 seconds or less to facilitate rapid failover and load balancing. Short TTLs mean more frequent cache expiration events, which means users hit the delay spike more often.

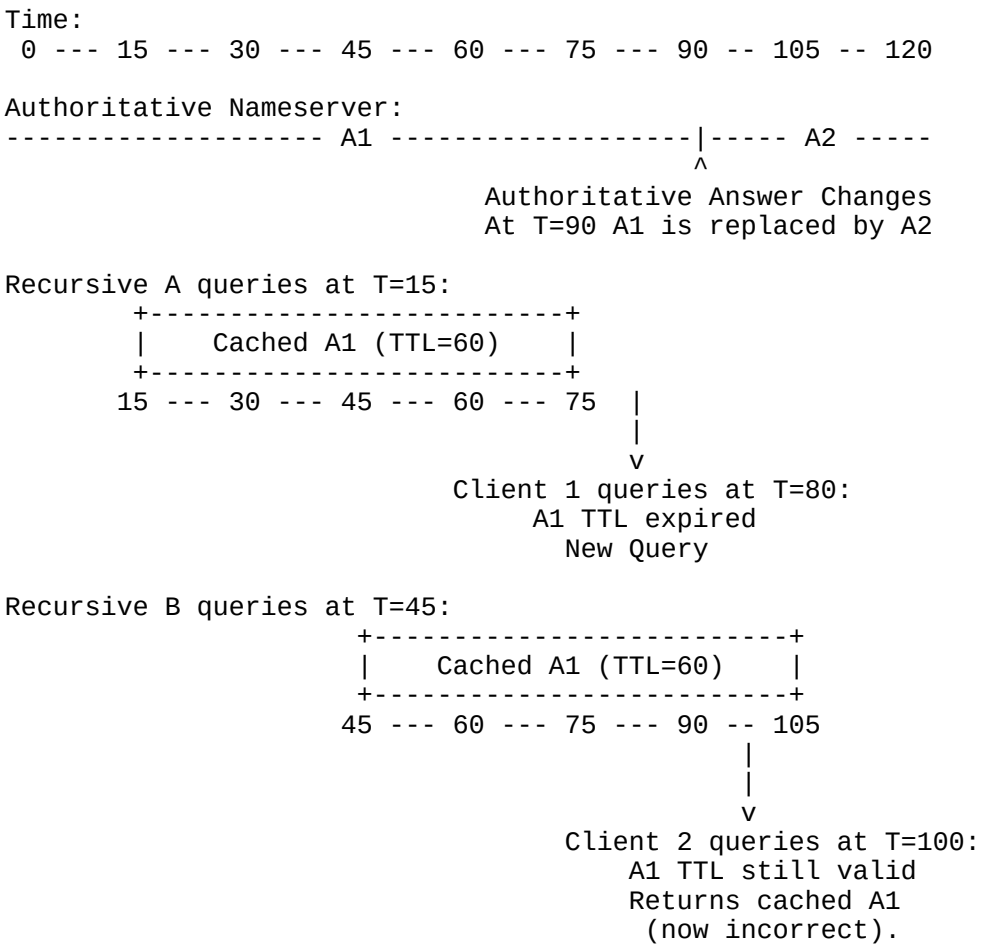
High-latency networks. On cellular networks, satellite links, or congested Wi-Fi, a DNS round trip can take one to five seconds. The penalty for a cache miss is severe.

Multiple queries per page load. A typical web page load involves DNS queries for dozens of hostnames (the page itself, stylesheets, scripts, images, analytics, ads). Often these DNS queries cannot be issued concurrently because some of the hostnames are not known until earlier transactions complete. For example, the hostnames for stylesheets and images are not known until after the base HTML document is fetched. If multiple hostnames have expired address records, the cumulative delay can be substantial.

First query after sleep. While a computer or smartphone is sleeping, normal wall-clock time continues to elapse, and by the time the device awakes, many cached records may have expired. Optimistic DNS avoids the delay penalty when a device is used after sleep or any other extended period of inactivity.

The fundamental observation behind Optimistic DNS is that in most cases, a DNS record that expired a little while ago usually still contains the correct data. Servers do not typically change IP addresses the instant a TTL expires. The TTL is a freshness hint, not a correctness deadline. It marks when the resolver decides to refresh, not when the data becomes incorrect.

The "stale" state is also relative. A user querying the same record at two different resolvers can see entirely different views of staleness. The diagram below illustrates this: both Recursive A and Recursive B cached A1 before the authoritative server replaced it with A2 at T=90. But because each resolver started its TTL clock at a different moment, they present a different view to their clients. A client querying Recursive A at time T=80 will cause a DNS request to the authoritative DNS server to retrieve fresh data. A client querying Recursive B twenty seconds later at time T=100 will receive cached data because Recursive B considers it still valid.



Client 1 issued a query at time T=80 and suffered a delay waiting for “fresh” data, even though, at that time, the data on the authoritative nameserver had not yet changed. In this case, using the expired cached data would have been fine, because it was still correct.

Client 2 issued a query twenty seconds later at time T=100, yet Recursive B happily sent it cached data that was no longer correct.

If the data was already considered "expired" at time T=80, it might be natural to think that 20 seconds later it should be even more expired, yet Client 2 received the opposite behavior. Statistically, averaged over a large set of queries, records that are older relative to their published TTL are more likely to be considered expired, but for a specific individual query the results are much less deterministic.

Thus, clients should not attach too much importance to the absolute value of a record's age relative to its declared TTL. The age of a record is a valuable hint as to whether it should be assumed to be still valid, but it is not a guarantee. Data being younger than its TTL is not a guarantee that it is still correct, and data being older than its TTL is not a guarantee that it is wrong.

The asynchronous connection-racing mechanism known as Happy Eyeballs [IETF72] [RFC6555] [RFC8305] [HEv3] is a key technology that makes Optimistic DNS useful, because it makes applications more robust in occasional situations where they may briefly receive incorrect information (Section 5.2).

4. The Preemptive Refresh Problem - Zeno's Paradox

One seemingly attractive approach to avoiding the delay spike might be to take inspiration from DHCP [RFC2131] [RFC2132], and renew a record before it expires, instead of waiting until after it is already expired before requesting an update. DHCP typically renews an address lease when half of the lease lifetime has elapsed, rather than waiting for the lease to expire and then briefly losing the right to use that IP address while a new request is processed.

Unfortunately, because of the way DNS operates, the DHCP-inspired approach of refreshing records in advance of their expiration does not work.

Suppose a DNS stub resolver makes a request for a name that is not currently in the recursive resolver's cache. Suppose that when the recursive resolver fetches the authoritative record it has a TTL of 24 hours, which it returns to the stub resolver.

If the stub resolver were to request the same record 12 hours later, the recursive resolver will tell the stub resolver that the record has a remaining TTL of 12 hours.

If the stub resolver were to request the same record 6 hours after that, the recursive resolver will tell the stub resolver that the record now has a remaining TTL of 6 hours.

Using this algorithm would result in an ever-accelerating query rate as the remaining TTL continues to count down towards zero, without the stub resolver learning anything new.

Eventually, the record finally expires from the recursive resolver's cache, and the stub resolver then suffers a delay spike waiting for the recursive resolver to fetch the authoritative record.

Currently, recursive resolvers will continue to return a cached record down to the last second of its lifetime. Even for a record that is the subject of a large volume of queries, recursive resolvers will not take the initiative of refreshing that record prior to its inevitable and entirely predictable expiration. Instead, recursive resolvers will let the record expire, and then suffer a delay spike on the very next query.

With the current behavior of recursive resolvers, strictly respecting a record's TTL and avoiding predictable delay spikes are incompatible goals. It is impossible to do both.

5. Enabling Technologies

When an application receives an expired address and immediately begins connecting to it, two things need to happen:

* When the fresh answer arrives moments later, the application needs a way to receive it -- which means the DNS API has to support delivering asynchronous results as they arrive.

* If the expired address is found to be wrong, the application needs to continue gracefully.

5.1. Asynchronous DNS Resolution

Traditional synchronous DNS APIs typically block until the complete set of answers is available. The caller issues a query, waits, receives one set of results, and the call is complete. Optimistic DNS cannot function with this model since there is no mechanism to deliver an expired answer now and update it with newer answers shortly afterwards.

Asynchronous DNS APIs support receiving multiple answers over time, including updated answers that supersede earlier ones. The operational model of asynchronous DNS resolution is that it is a performance optimization over rapid polling. If an application were to call an API like getaddrinfo() as fast as it can in a tight loop, then the application would always have the latest information available from the local stub resolver, but it would be inefficient

and wasteful of CPU time. Most consecutive calls to getaddrinfo() would return unchanged information. The guiding principle of an asynchronous DNS resolution API is that it gives the application the exact same information as repeated calls to the equivalent synchronous API, but more efficiently. If a fresh new query would yield different results, then the client's asynchronous DNS operation MUST be given asynchronous notifications to deliver the new set of results. This avoids the temptation for clients to resort to polling, in the belief that polling gives them better results than trusting the asynchronous notification mechanism. For efficiency, the application is notified via an asynchronous notification only when the set of answers changes. This is similar to the efficiency trade-offs present in other APIs, where asynchronous notification is more efficient than polling. For example, an application can repeatedly poll to find if the content of a directory has changed, but using asynchronous filesystem notifications gives the same effect more efficiently.

This is the resolution model defined by Multicast DNS [RFC6762] and is valuable for unicast DNS too. Asynchronous DNS resolution is supported in Apple's networking APIs [ZC], and in other DNS stub resolver implementations like getdns [getdns].

This model naturally supports the incremental delivery that Optimistic DNS requires. Expired records arrive in the first callback, within microseconds. Fresh records from the network arrive in subsequent callbacks, within milliseconds. The application can act on the expired answer immediately by opening a connection. If the address has changed, the application can start a new connection to the updated address. If the address is the same, the connection is already established and the fresh answer serves as confirmation.

5.2. Happy Eyeballs

Optimistic DNS delivers DNS answers quickly, with the caveat that, on rare occasions, some of those answers might be stale and incorrect, in which case updated answers will be delivered as soon as they are available. To make this optimistic approach viable, it MUST be coupled with networking code that is designed to embrace this uncertainty, and account for the fact that all data received from a network is necessarily at least a little stale by the time it arrives, and may subsequently be found to be incorrect. Without Happy Eyeballs, a wrong expired address would mean a failed connection and user frustration.

Happy Eyeballs [IETF72] [RFC6555] [RFC8305] [HEv3] defines algorithms for racing connection attempts across multiple addresses and address families. When a client has several candidate addresses for a

destination, Happy Eyeballs staggers connection attempts, in order of expected likelihood of success, with short delays, and uses whichever connection succeeds first.

The Happy Eyeballs mechanism pairs naturally with Optimistic DNS. When using Happy Eyeballs, the networking code takes the set of answers immediately available in the local cache and selects the address it predicts is most likely to succeed. If the prediction is correct, which is the common case, the connection succeeds and no additional traffic is generated. The connection may succeed before the fresh DNS answer from the network even arrives.

In the rare cases where the first connection attempt does not succeed within the expected network round-trip time, the TCP SYN (or equivalent connection request packet) is retransmitted, and a second connection attempt is immediately initiated to the next address in order of preference. The first connection attempt is not abandoned when the second one starts -- its usual schedule of retransmissions is still followed -- and the two attempts proceed in parallel. If the second connection attempt does not succeed within the expected time, a third attempt is initiated, and so on.

Running connection attempts in parallel rather than sequentially prevents long stalls waiting for a connection to time out and fail before the next attempt is started. Staggering the start times by the expected response time means that in the majority of cases only the first connection attempt is needed, which avoids generating unnecessary network traffic or extra load on servers.

If new DNS results are received during this procedure, the list of candidate addresses is updated accordingly for use in subsequent connection requests.

Using Optimistic DNS in conjunction with Happy Eyeballs means that, in the common case, a connection to the correct server is made faster, with no additional network traffic. In the rare case when a server address has actually changed, the cost is a small amount of wasted network traffic while waiting for the DNS reply. Once the reply arrives, Happy Eyeballs can immediately connect to the correct server, with no additional delay beyond what would have been experienced had the application just waited for the DNS reply in the first place. The cost of waiting for a fresh answer that simply confirms what the cache already had is a guaranteed delay of the full network round-trip time on every cache expiry. For most applications, the expected value of the optimistic approach is clearly positive.

After the Happy Eyeballs algorithm has succeeded in establishing a connection to an acceptable destination (determined by verifying the TLS certificate, or otherwise, as appropriate) the asynchronous DNS operation MUST be stopped. Once an application has established a successful connection, it should make its future decisions based on the viability of that connection, rather than any subsequent information to the contrary received from asynchronous DNS. Cancelling the asynchronous DNS operation prematurely, before a successful connection has been established, would prevent the reception of future updated answers, which would defeat the purpose of Optimistic DNS, which is to deliver available answers quickly while still ensuring eventual correctness. Failure to cancel after an acceptable connection has been made would be a waste of resources, and in the extreme case would constitute a resource leak on the client device.

Applications using Optimistic DNS and Happy Eyeballs SHOULD follow appropriate security practices (Section 10) to ensure that they are connecting to the intended host or service on the network.

5.3. Combined Effect

Asynchronous DNS resolution makes Optimistic DNS _possible_. It provides the delivery mechanism for multiple waves of results. Happy Eyeballs makes Optimistic DNS _safe_. It ensures that acting on a wrong expired address is not fatal to the overall connection attempt. Together, the application starts connecting instantly with provisional cached addresses, the connection race handles any staleness gracefully, and the fresh DNS answer arrives as an update confirming or correcting the initial result.

6. Optimistic DNS Overview

Building on the asynchronous resolution model and connection racing described above, Optimistic DNS introduces a specific modification to stub resolver behavior. When an application that has opted in issues a DNS query and the stub resolver’s cache contains expired records matching that query, the resolver performs two actions in parallel:

1. It immediately returns the expired cached records to the application.

2. It issues a standard DNS query on the network to obtain fresh records.

As fresh answers arrive from the network, the resolver delivers them to the application through the asynchronous callback mechanism. Table 1 shows both waves for a query where www.example.com has an expired A record (203.0.113.34) in the cache, and the fresh answer returns the same address:

Time	Event	Notes
T+0us	App queries www.example.com	
T+5us	Callback: 203.0.113.34	Expired
T+120ms	(data unchanged, no callback)	Fresh Answer Same

Table 1: Optimistic DNS when data is unchanged

When a fresh positive answer matches the previous Optimistic Positive Answer, no second callback is delivered. The application that began connecting at T+5us saved 120 milliseconds compared to waiting for the fresh answer.

When the data has changed (for example, if the server moved to a new address), or a fresh negative answer confirms a previous Optimistic Negative Answer, the application is notified. Sending confirmations of Optimistic Negative Answers is necessary because the client may be waiting for that negative answer confirmation before proceeding with its next steps.

Time	Event	Notes
T+0us	App queries www.example.com	
T+5us	Callback: 203.0.113.34	Expired
T+120ms	Callback: 198.51.100.42	Fresh Answer Different

Table 2: Optimistic DNS when data has changed

In Table 2 the expired answer contained a stale and incorrect address. An application that connected to 203.0.113.34 may find that the connection fails with an ICMP Host Unreachable error, a TCP RST, a TLS failure, or other unexpected content. But the fresh answer arrives 120 milliseconds later, and the application can retry with the correct address. The total time to a successful connection is still only about 120 milliseconds -- roughly the same as it would have been without Optimistic DNS.

Optimistic DNS never makes things substantially worse for the application. In the common case where the data has not changed, it makes things dramatically faster. In the uncommon case where the data has changed and the server is no longer available at the previous address, the cost is a small amount of wasted traffic attempting to reach the previous address.

Optimistic DNS is an opt-in mechanism. Applications that do not request it receive conventional stub resolver behavior: expired records are ignored, and only fresh network answers are returned. This ensures backward compatibility and allows applications to choose the tradeoff that suits their needs. Applications that do not support both Happy Eyeballs and appropriate application-layer security SHOULD NOT use Optimistic DNS. These applications are very dependent on always getting the right answers from DNS every time, and do not recover gracefully when delivered stale data by Optimistic DNS.

7. Implementation Details

7.1. Query Initiation

To use Optimistic DNS, an application signals its willingness to receive expired answers when it issues a DNS query. This signaling is a local matter between the application and the stub resolver.

One possible mechanism is for the stub resolver API to provide a flag that the application includes with its query. Setting this flag indicates that the application is prepared to handle expired answers and will treat them appropriately.

7.2. Cache Lookup with Expired Records

When the stub resolver processes a query, it iterates through its cache looking for records that match the query’s name, type, and class. For each matching record, it determines whether the record has expired by comparing the time elapsed since the record was received against the record’s original TTL.

If unexpired records are found in the cache, they are returned to the application as a normal cache hit and no network query is needed.

If a matching cache entry (positive or negative) has expired, then this Optimistic Answer is delivered to the Optimistic DNS client.

Optimistic Answers are subject to concurrent verification (Section 7.3) and may subsequently result in an asynchronous update to the client if newer information is discovered from the network.

Optimistic Positive Answers indicate the stub resolver's optimistic best guess of the answer.

Optimistic Negative Answers (resulting from a previous NXDOMAIN or NODATA response) indicate the stub resolver's best guess that the requested record probably still doesn't exist. Optimistic Negative Answers SHOULD NOT be used to generate error messages to the user. Such error messages SHOULD only be generated after a network query has confirmed that the named record still does not exist. Optimistic Negative Answers can be useful in certain search scenarios (like applying DNS domain name search lists) where the nonexistence of a certain record is a prerequisite for using some other record, and the tentatively presumed nonexistence of a certain record can serve as a useful hint that the client should begin work speculatively looking up other names that may be required by the search algorithm (Section 8.3).

7.3. Parallel Network Query

If no unexpired records are found, the stub resolver MUST issue a standard DNS query on the network. This query proceeds through the normal resolution path: contacting configured recursive resolvers, following CNAME chains, and so on.

Fresh answers from the network are delivered to the application through an asynchronous callback mechanism. The application's networking code MUST use these fresh answers to update the list of answers it received earlier.

7.4. Cache Management

Storing expired records forever could consume an unbounded amount of memory, and return egregiously old records.

For this reason, stub resolvers SHOULD NOT store records for more than one week after the TTL has expired.

The time limit of one week is selected so that a user could go home from work on a Thursday night, take a four-day long-weekend break, return to work the next Tuesday, and still benefit from using expired cached records for faster performance. Storing expired records for less time risks eliminating the benefit of Optimistic DNS in many cases; storing expired records for more time is currently considered to offer little additional benefit, and using excessively old data could lead to unexpected undesirable consequences.

7.5. CNAME Handling

CNAME records introduce a complication for Optimistic DNS. When a DNS query encounters a CNAME record, the resolver must follow one or more records in a CNAME chain to find the ultimate answer. If a CNAME record itself is expired, it may no longer be correct.

Consider the following example. An application queries for www.example.com, which has a CNAME record pointing to cdn.example.net. Both records are in the cache but expired:

```
www.example.com.   CNAME  cdn.example.net.  (expired)
cdn.example.net.   A      198.51.100.42     (expired)
```

If the resolver follows the expired CNAME to cdn.example.net and returns the expired A record, the application receives a result. But if the CNAME has changed (www.example.com now points to cdn2.example.net instead), the resolver has followed a stale chain and returned a record for the wrong name.

To handle this correctly, when the stub resolver encounters CNAME records during optimistic resolution, it takes the following steps:

1. If the record is expired, and a background query is not already in progress for this record name, then a background query is initiated, as for any other expired record.

2. The CNAME referral is followed. (If the target of the CNAME referral is another CNAME record, then this process repeats until a non-CNAME answer is found.)

3. If, at any time while an asynchronous DNS query is active, the record data for any of the names in the CNAME chain changes, then the query is re-evaluated starting from the original query name, as if it were a fresh query. This maintains the operational model of asynchronous DNS resolution, that it is simply a more efficient equivalent to repeated polling. If a fresh new query would yield different results, then the client's asynchronous DNS operation receives asynchronous notifications to deliver the new set of results, giving it the exact same information that a fresh query at that time would yield.

This rewind-and-restart approach ensures that the application continues to receive a complete, consistent answer for its query, even as DNS record changes may mean that the CNAME chain for a given name changes over time.

8. Interaction with Other DNS Features

8.1. Uncacheable Records

The DNS specifications require that DNS records with a TTL of zero MUST NOT be cached [RFC1035]. This document does not change that requirement; subsequent queries for these records always result in a new network request.

8.2. DNSSEC

Optimistic DNS extends the cache lifetime for DNS records, as signaled by the TTL values.

Optimistic DNS does not extend the validity periods of cryptographic signatures.

Optimistic DNS does not interpret the content of any records, including cryptographic records like RRSIG, and will deliver DNS records after their cache lifetime has expired. Clients are responsible for checking and respecting the validity periods of cryptographic signatures.

8.3. DNS Domain Name Search Lists

Most DNS stub resolvers support domain name search lists [RFC1034]. Domain name search lists can be configured manually, or learned automatically from the network using DHCP [RFC3397] [RFC3646] or IPv6 Router Advertisements [RFC8106].

Applying domain name search lists can be thought of as an iterative search algorithm that operates at a layer above the stub resolver's query mechanism. Each query made in the course of executing the domain name search algorithm is a normal query for a fully qualified domain name, and is subject to all the usual procedures like following CNAME referrals. Optimistic DNS can be used to speed up the process of handling domain name search lists.

When a client issues a query for a name that is not considered fully qualified, the stub resolver applies each of the names from the domain name search list, in order, and uses the first name that returns a positive answer. The user expectation is that the order of the domain name search list will be respected strictly. This means that results from a later name in the search list cannot be used until negative answers have been confirmed for all the earlier entries in the search list. The network queries do not need to be performed sequentially, one at a time (intelligently performing queries in parallel can result in a faster result for the user), but the final determination of which name to use cannot be made until all the necessary Optimistic Negative Answers have been confirmed.

For this reason, Optimistic DNS can use Optimistic Negative Answers as a performance hint to tell it that it should start a parallel query for the next name in the search list, but the decision to use a particular answer needs to wait until all earlier Optimistic Negative Answers have been confirmed.

If a stub resolver were to issue parallel queries for all names in the domain name search list, this could result in a lot of unnecessary network traffic, particularly for users with many names in their search lists. If a stub resolver were to issue its queries sequentially, this could result in poor performance for the user. Optimistic DNS provides a good balance between these two extremes. Optimistic DNS allows the stub resolver to issue parallel queries for all the names it reasonably expects it will need to check, without generating wasteful traffic for names near the end of the domain name search list that come after a name that is believed to have a positive answer. Optimistic DNS rapidly performs all the queries it needs to do to confirm record nonexistence, and then returns the first positive answer, whether fresh or expired. If an expired positive answer turns out to be wrong when confirmed with a query on the network, then the newly updated answer is delivered asynchronously to the client. If the newly updated answer is negative, then the domain name search algorithm resumes, continuing with the next untried name in the list, and continues until some positive answer, if any, is returned.

8.4. Encrypted DNS Transports

Optimistic DNS is transport-agnostic. It operates entirely within the stub resolver's cache layer, which sits above the transport layer. Whether the stub resolver communicates with recursive resolvers using classic DNS over UDP/TCP [RFC1035], DNS over TLS (DoT) [RFC7858], DNS over HTTPS (DoH) [RFC8484], or DNS over QUIC (DoQ) [RFC9250], the Optimistic DNS mechanism functions identically.

8.5. Serving Stale Data at Recursive Resolvers

Serving Stale Data to Improve DNS Resiliency [RFC8767] describes a mechanism for recursive resolvers to serve stale cached data when they are unable to refresh it from authoritative servers. Optimistic DNS and RFC 8767 address different levels of the DNS resolution chain:

Optimistic DNS. Operates at the stub resolver on the end-user's device. Serves expired cached data proactively, while simultaneously initiating a network query. The primary goal is delay reduction -- eliminating the TTL expiry spike.

RFC 8767. Operates at the recursive resolver. Serves stale data when upstream authoritative servers are unreachable. The primary goal is resiliency -- maintaining DNS service during outages.

The two mechanisms are complementary and can be deployed simultaneously. When both are active, the resolution chain has two layers of staleness tolerance:

1. The stub resolver returns expired records optimistically, eliminating client-perceived delay.

2. If the recursive resolver's cache has also expired and the authoritative server is unreachable, the recursive resolver can serve its own stale data rather than returning SERVFAIL.

Optimistic DNS delivers the data it has immediately, while also delivering updated data the moment it becomes available. It gives the application the benefit of timely data, which could occasionally be wrong, without depriving the application of also receiving the exact data it would have received with a traditional non-optimistic DNS query, at the time it would have received it.

In contrast, RFC 8767 serves stale data without any provision for a future asynchronous notification when the desired data becomes available.

Optimistic DNS is not applicable to recursive resolvers because Optimistic DNS relies on the ability to deliver its best available information immediately, and then asynchronously update that information promptly if newer information becomes available. Stub resolvers that provide Asynchronous DNS resolution APIs to client applications are able to deliver these essential asynchronous updates. Recursive resolvers do not currently have any good way to issue asynchronous updates to correct an earlier answer that is subsequently discovered to be wrong. In principle, stub resolvers could use DNS Push Notifications [RFC8765] to receive asynchronous updates from a recursive resolver, but in practice this might place too much load on recursive resolvers. The current recommended solution for recursive resolvers returning stale data is that the TTL is reported as being 30 seconds [RFC8767]. This has the effect of instructing stub resolvers to poll the recursive resolver once every 30 seconds to ascertain if newer data has become available.

9. Operational Considerations

A consequence of Optimistic DNS is that clients may continue to use an old IP address longer than expected. There are other reasons that stale addresses may persist and cause clients to use an old IP address longer than expected, but Optimistic DNS adds a new way that this can happen.

Suppose a server operator has a DNS address record for website.example.com with a TTL of 24 hours, and the operator updates that address record to point to a new IP address. Depending on when that address record was cached at various recursive resolvers around the world, the operator may expect to see a roughly linear decrease in new incoming connection requests to the old IP address in the 24 hours after the address record was updated, ending with no new connection requests being received after 24 hours have passed. In reality there are various reasons that some clients may still attempt to use a stale address, but in practice there should be very few incoming connection requests to the old IP address after 24 hours. The server operator has to keep the website available at the old address for at least 24 hours, to accommodate clients using the old address.

Optimistic DNS changes this assumption. Using the same 24-hour TTL, the decline in the incoming connection rate will not be quite as rapid, and incoming connection requests to the old IP address may continue to be received for a week after the TTL has expired (Section 7.4).

However, Optimistic DNS does not mean that the server operator is forced to maintain the availability of their website at the old IP address for a week after the address change. Because clients using the expired IP address are using Optimistic DNS, they have the benefit of asynchronous DNS and Happy Eyeballs, which means that they will promptly switch to the new IP address, and the brief failed attempt to use the old IP address is inconsequential.

If a new IP address is established for the server but existing connections to the old IP address continue to work, then there is no reason for clients to abandon their working connections. The decision about if and when to migrate existing clients to the new IP address is in the hands of the server operator. The server operator might choose to allow those connections to remain. If the server operator wishes to have clients reconnect using the new IP address, then that could be achieved using an in-band message on the existing connection, or by simply shutting down the server at the old IP address, so that clients re-establish their connections using the new IP address.

Moreover, Optimistic DNS gives server operators the freedom to use shorter DNS TTLs safely. One of the reasons why server operators may choose to use long TTLs is to avoid their users experiencing the periodic delay spikes caused by short TTLs. When Optimistic DNS eliminates the concerns about periodic delay spikes, it becomes reasonable to use shorter DNS TTLs. With widespread use of Optimistic DNS, server operators are free to use much shorter DNS TTLs, like five minutes. Optimistic DNS clients will suffer no periodic delay spikes because of the short DNS TTLs, and when the time comes that it is necessary to update the server address for website.example.com, the transition to the new server can be completed in minutes, rather than hours.

10. Security Considerations

Optimistic DNS introduces a tradeoff between speed and freshness. Applications that use expired answers must be prepared for those answers to be occasionally incorrect. Applications using Optimistic DNS SHOULD employ application-layer security (e.g., using TLS or QUIC) when connecting to addresses obtained from expired answers, to detect cases where the address has been reassigned.

Networking always entails a degree of uncertainty -- for example, packets may be intercepted in transit and not reach their intended destination -- and for correct operation, secure applications have to guard against this. Optimistic DNS adds a new way that packets may not reach their intended destination, but does not fundamentally alter the importance of application-layer security.

IP Address Reuse. When a DNS record expires, the IP address it contained may no longer be associated with the previous host or service. For HTTPS connections, TLS certificate validation will detect this mismatch and prevent data from being sent to the wrong party. For unencrypted protocols, there is a risk of connecting to an unintended server.

Poisoning Amplification. If an attacker successfully poisons a cache entry, Optimistic DNS could extend the lifetime of the poisoned record beyond its original TTL. However, the record will be purged after the retention period expires. Moreover, the parallel network query will obtain and deliver the correct answer, giving the application an opportunity to detect the discrepancy.

Expired Records Retention Period. The recommended maximum retention period of one week (Section 7.4) bounds the maximum time an expired record can be served. Implementations MAY allow this period to be configured. Shorter periods reduce the window of exposure to stale data but also reduce the effectiveness of Optimistic DNS for infrequently-accessed names.

11. IANA Considerations

This document has no IANA actions.

12. References

12.1. Normative References

[HEv3] Pauly, T., Schinazi, D., Jaju, N., and K. Ishibashi, "Happy Eyeballs Version 3: Better Connectivity Using Concurrency", Work in Progress, Internet-Draft, draft-ietf-happy-happyeyeballs-v3-03, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-happy-happyeyeballs-v3-03>.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/rfc/rfc1034>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/rfc/rfc1035>.

[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>.

[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>.

12.2. Informative References

[getdns] NLnet Labs, Sinodun and No Mountain Software, "Welcome to getdns!", n.d., <https://getdnsapi.net/>.

[IETF72] Cheshire, S., "Stuart Cheshire on IPv6 Adoption", 2008, <https://www.stuartcheshire.org/IETF72/>. This presentation introduced the concept of asynchronous concurrent connection racing, later known as Happy Eyeballs.

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, DOI 10.17487/RFC2131, March 1997, <https://www.rfc-editor.org/rfc/rfc2131>.

[RFC2132] Alexander, S. and R. Droms, "DHCP Options and BOOTP Vendor Extensions", RFC 2132, DOI 10.17487/RFC2132, March 1997, <https://www.rfc-editor.org/rfc/rfc2132>.

[RFC3397] Aboba, B. and S. Cheshire, "Dynamic Host Configuration Protocol (DHCP) Domain Search Option", RFC 3397, DOI 10.17487/RFC3397, November 2002, <https://www.rfc-editor.org/rfc/rfc3397>.

[RFC3646] Droms, R., Ed., "DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3646, DOI 10.17487/RFC3646, December 2003, <https://www.rfc-editor.org/rfc/rfc3646>.

[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>.

[RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <https://www.rfc-editor.org/rfc/rfc6762>.

[RFC7858] 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>.

[RFC8106] Jeong, J., Park, S., Beloeil, L., and S. Madanapalli, "IPv6 Router Advertisement Options for DNS Configuration", RFC 8106, DOI 10.17487/RFC8106, March 2017, <https://www.rfc-editor.org/rfc/rfc8106>.

[RFC8305] 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>.

[RFC8484] 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>.

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

[RFC8767] Lawrence, D., Kumari, W., and P. Sood, "Serving Stale Data to Improve DNS Resiliency", RFC 8767, DOI 10.17487/RFC8767, March 2020, <https://www.rfc-editor.org/rfc/rfc8767>.

[RFC9250] 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>.

[ZC] Cheshire, S. and D. H. Steinberg, "Zero Configuration Networking: The Definitive Guide", O'Reilly Media, ISBN 978-0-596-10100-5, December 2005.

Appendix A. Deployment History

A.1. Optimistic DNS in mDNSResponder

The mDNSResponder project is an open-source system stub resolver, and runs on macOS, iOS, tvOS, watchOS, Microsoft Windows, Android, Linux, and other platforms. This section describes its concrete implementation of the Optimistic DNS mechanism described in this document.

Optimistic DNS in Apple's mDNSResponder code was first shipped enabled by default in macOS 10.14 (Mojave) and iOS 12 in September 2018. It has been active on all Apple platforms since that release, serving as the default stub resolver behavior for all applications that use Apple's recommended networking APIs.

Prior to implementing Optimistic DNS, the mDNSResponder code addressed the Zeno's paradox problem (Section 4) by implementing a modest form of DNS TTL stretching. The mDNSResponder code stretches TTLs by 25%, plus two seconds to account for minor clock variations. This results in the situation where, when the age of the record as viewed by the stub resolver reaches 80% of its stretched TTL, the actual TTL of the record as viewed by the recursive resolver has expired.

If a client performs a new DNS query for a record within the time window of 80-100% of the stretched TTL, then the mDNSResponder stub resolver code will return the available answer to the client immediately, while in parallel sending a query to the recursive resolver.

In addition, if an asynchronous query is active on the client at the time the record age passes 80% of its stretched TTL, then the mDNSResponder stub resolver code will send a DNS query to the recursive resolver to fetch fresh data, and will update the client with fresh information if new data is received.

This DNS query to the recursive resolver happens after the recursive resolver's copy of the record has expired, causing the recursive resolver to fetch a new fresh copy of the authoritative record, but because available answers are delivered to DNS clients immediately, DNS clients do not experience a delay spike while waiting for the recursive resolver to refresh the record. Thus, traditional DNS queries using the mDNSResponder stub resolver may receive answers that are up to 25% beyond their original lifetime.

The development of Happy Eyeballs in 2008 [IETF72] [RFC6555] [RFC8305] [HEv3] made it feasible to increase effective record lifetimes beyond a modest extension of just 25%, and this new capability is what made Optimistic DNS possible.

A.1.1. Signaling

This document describes query initiation as a local signaling matter between the application and the stub resolver. The mDNSResponder code implements this through the following API flags:

kDNSServiceFlagsAllowExpiredAnswers. Set by the application when issuing a query to request delivery of expired cached records.

kDNSServiceFlagsExpiredAnswer. Set by the resolver on individual answer callbacks to indicate that the record being returned is expired. In the case of Optimistic Negative Answers, the flag is a useful hint to the client about how much trust it should place

in this answer. An expired Optimistic Negative Answer should be quickly updated with a fresh confirmation of the negative answer, or a new positive answer. In the case of Optimistic Positive Answers, the flag has little significance, since the application verifies the validity of positive answers by attempting communication with those addresses. Developers should take care in how they interpret this answer property because, as mentioned earlier, the notion of record expiry is subjective, and this flag is easily misunderstood and misused by developers who think it carries more significance than it really does.

kDNSServiceFlagsAnsweredFromCache. Set by the resolver to indicate that the record was served from the local cache rather than from a network response. Set regardless of whether the record is expired, allowing the application to distinguish cached answers (immediate) from network answers (delayed). Providing an equivalent indication in other implementations is not recommended, because this flag is easily misunderstood and misused by developers who think it carries more significance than it really does. If two clients happen to query for the same domain name at almost exactly the same time then one will be told that the answer came from the cache and the other will be told that it did not. Which client gets told which is effectively a coin toss, and usually has little significance. Revealing whether an answer was served from the cache also has privacy implications. A rogue application could issue queries for a large number of popular domain names, and from the replies indicating which domain names were already in the local cache and which were not, the rogue application could infer a list of which services this user has recently accessed.

A.1.2. Record Lifecycle

The main cache lookup operation requires that expired records remain in the cache so that they are available to serve optimistically. Conceptually, the stub resolver code saves all records for up to seven days. Practically, the stub resolver code may want to limit its memory usage. The mDNSResponder code approaches this with a three-state record lifecycle:

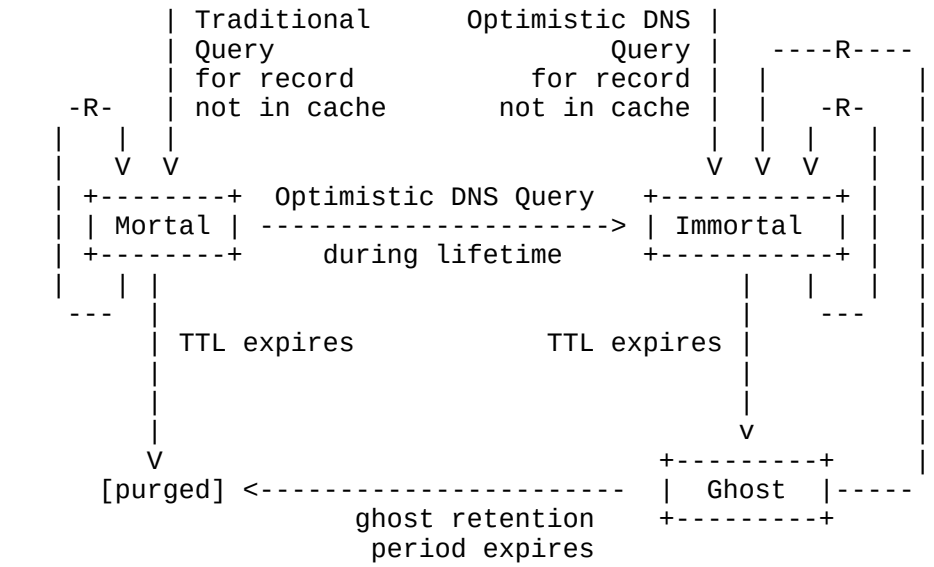
Mortal (default). The normal cache state for a record retrieved to

answer a traditional (not Optimistic DNS) query. If subsequently queried by an Optimistic DNS operation, this record gets promoted to immortal. Mortal records may be refreshed in the course of normal operation, though because of Zeno's paradox (Section 4) such refreshes are likely to be successful only for records in the final 20% of their stretched TTL lifetime. When the (stretched) TTL of a mortal record expires the record is eligible for immediate removal.

Immortal. A record marked to survive past its (stretched) TTL expiry. This is the initial cache state for a record retrieved to answer an Optimistic DNS query. In addition, promotion from mortal to immortal occurs when a cached record is used to answer a query from an application that has opted in to Optimistic DNS. (The logic is that if an application has made an Optimistic DNS query for this DNS name, that is a hint that there may be more such Optimistic DNS queries in the future. If there has not been even one single Optimistic DNS query for this DNS name, then that is a sign that whatever applications are resolving this DNS name do not yet use Optimistic DNS, so saving these records for a long time would be a waste of memory.) Successful subsequent queries for immortal records will refresh their lifetime, as it is with mortal records. When the (stretched) TTL of an immortal record expires the record becomes a ghost record.

Ghost. An immortal record whose (stretched) TTL has expired. Ghost records linger in the cache, available to serve future Optimistic DNS queries. If an Optimistic DNS query is issued that matches a ghost record, then the ghost record data is delivered immediately to the application, and the simultaneous DNS query over the network, if successful, will refresh the ghost record and restore it to immortal state. If not refreshed for the ghost retention period (maximum of one week), ghost records are purged.

The complete lifecycle:



This creates a virtuous cycle: the more frequently a name is queried with Optimistic DNS, the more likely the cache will contain a ghost record for it the next time the TTL expires.

Acknowledgments

TODO: Acknowledgments.

Authors' Addresses

Gautam Akiwate
Apple Inc.
One Apple Park Way
Cupertino, CA 95014
United States of America
Email: gakiwate@apple.com

Phil Flack
Apple Inc.
One Apple Park Way
Cupertino, CA 95014
United States of America
Email: pf-ietf@flacko.com

Stuart Cheshire
Apple Inc.
One Apple Park Way
Cupertino, CA 95014
United States of America
Email: cheshire@apple.com