DNSOP Working Group
Internet-Draft
Updates: 1034, 1035 (if approved)
Intended status: Standards Track
Expires: May 3, 2018

D. Lawrence
Akamai Technologies
W. Kumari
Google

October 30, 2017

Serving Stale Data to Improve DNS Resiliency
draft-ietf-dnsop-serve-stale-00

Abstract

This draft defines a method for recursive resolvers to use stale DNS data to avoid outages when authoritative nameservers cannot be reached to refresh expired data.

Ed note

Text inside square brackets ([]) is additional background information, answers to frequently asked questions, general musings, etc. They will be removed before publication. This document is being collaborated on in GitHub at <https://github.com/vttale/serve-stale>. The most recent version of the document, open issues, etc should all be available here. The authors gratefully accept pull requests.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 3, 2018.

Table of Contents

1. Introduction

Traditionally the Time To Live (TTL) of a DNS resource record has been understood to represent the maximum number of seconds that a record can be used before it must be discarded, based on its description and usage in [RFC1035] and clarifications in [RFC2181].

This document proposes that the definition of the TTL be explicitly expanded to allow for expired data to be used in the exceptional circumstance that a recursive resolver is unable to refresh the information. It is predicated on the observation that authoritative server unavailability can cause outages even when the underlying data those servers would return is typically unchanged.

A method is described for this use of stale data, balancing the competing needs of resiliency and freshness. While this intended to

be immediately useful to the installed base of DNS software, an [RFC6891] EDNS option is also proposed for enhanced signalling around the use of stale data by implementations that understand it.

2. Terminology

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 [RFC2119] when, and only when, they appear in all capitals, as shown here.

For a comprehensive treatment of DNS terms, please see [RFC7719].

3. Background

There are a number of reasons why an authoritative server may become unreachable, including Denial of Service (DoS) attacks, network issues, and so on. If the recursive server is unable to contact the authoritative servers for a name but still has relevant data that has aged past its TTL, that information can still be useful for generating an answer under the metaphorical assumption that, "stale bread is better than no bread."

[RFC1035] Section 3.2.1 says that the TTL "specifies the time interval that the resource record may be cached before the source of the information should again be consulted", and Section 4.1.3 further says the TTL, "specifies the time interval (in seconds) that the resource record may be cached before it should be discarded."

A natural English interpretation of these remarks would seem to be clear enough that records past their TTL expiration must not be used, However, [RFC1035] predates the more rigorous terminology of [RFC2119] which softened the interpretation of "may" and "should".

[RFC2181] aimed to provide "the precise definition of the Time to Live" but in Section 8 was mostly concerned with the numeric range of values and the possibility that very large values should be capped. (It also has the curious suggestion that a value in the range 2147483648 to 4294967295 should be treated as zero.) It closes that section by noting, "The TTL specifies a maximum time to live, not a mandatory time to live." This is again not [RFC2119]-normative language, but does convey the natural language connotation that data becomes unusable past TTL expiry.

Several major recursive resolver operations currently use stale data for answers in some way, including Akamai, OpenDNS, Xerocole, and

Nominum. Their collective operational experience is that it provides significant benefit with minimal downside.

4. Standards Action

The definition of TTL in [RFC1035] Sections 3.2.1 and 4.1.3 is amended to read:

TTL a 32 bit unsigned integer number of seconds in the range 0 - 2147483647 that specifies the time interval that the resource record MAY be cached before the source of the information MUST again be consulted. Zero values are interpreted to mean that the RR can only be used for the transaction in progress, and should not be cached. Values with the high order bit set SHOULD be capped at no more than 2147483647. If the authority for the data is unavailable when attempting to refresh the data past the given interval, the record MAY be used as though it has a remaining TTL of 1 second.

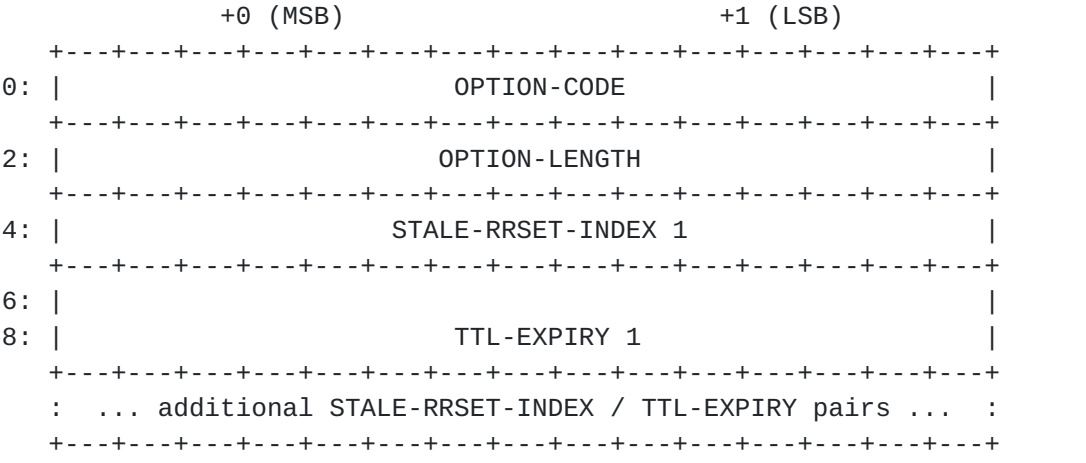
5. EDNS Option

While the basic behaviour of this answer-of-last-resort can be achieved with changes only to resolvers, explicit signalling about the use of stale data can be done with an EDNS [RFC6891] option.

[This section will be fleshed out a bit more thoroughly if there is interest in pursuing the option.]

5.1. Option Format

The option is structured as follows:



OPTION-CODE 2 octets per [RFC6891]. For Serve-Stale the code is TBD by IANA.

OPTION-LENGTH: 2 octets per [RFC6891]. Contains the length of the payload following OPTION-LENGTH, in octets.

STALE-RRSET-INDEX Two octets as a signed integer, indicating the first RRSet in the message which is beyond its TTL, with RRSet counting starting at 1 and spanning message sections.

TTL-EXPIRY Four octets as an unsigned integer, representing the number of seconds that have passed since the TTL for the RRset expired.

5.2. Option Usage

Software making a DNS request can signal that it understands Serve-Stale by including the option with one STALE-RRSET-INDEX initialized to any negative value and TTY-EXPIRY initialized to 0. The index is set to a negative value to detect mere reflection of the option by responders that don't really understand it.

If the request is made to a recursive resolver which used any stale RRsets in its reply, it then fills in the corresponding indices and staleness values. If no records are stale, STALE-RRSET-INDEX and TTL-EXPIRY are set to 0.

If the request is made to an authoritative nameserver, it can use the option in the reply to indicate how the resolver should treat the records in the reply if they are unable to be refreshed later. A default for all RRsets in the message is established by setting the first STALE-RRSET-INDEX to 0, with optional additional STALE-RRSET-INDEX values overriding the default. A TTL-EXPIRY value of 0 means to never serve the RRset as stale, while non-zero values represent the maximum amount of time it can be used before it MUST be evicted. [Does anyone really want to do this? It adds more state into resolvers. Is the idea only for purists, or is there a practical application?]

No facility is made for a client of a resolver to signal that it doesn't want stale answers, because if a client has knowledge of Serve-Stale as an option, it also has enough knowledge to just ignore any records that come back stale. [There is admittedly the issue that the client might just want to wait out the whole attempted resolution, which there's currently no way to indicate. The absolute value of STALE-RRSET-INDEX could be taken as a timer the requester is willing to wait for an answer, but that's kind of gross overloading it like that Shame to burn another field on that though, but on the other hand it would be nice if a client could always signal its impatience level - "I must have an answer within 900 milliseconds!"]

6. Example Method

There is conceivably more than one way a recursive resolver could responsibly implement this resiliency feature while still respecting the intent of the TTL as a signal for when data is to be refreshed.

In this example method three notable timers drive considerations for the use of stale data, as follows:

- A client response timer, which is the maximum amount of time a recursive resolver should allow between the receipt of a resolution request and sending its response.

- A query resolution timer, which caps the total amount of time a recursive resolver spends processing the query.

- A maximum stale timer, which caps the amount of time that records will be kept past their expiration.

Recursive resolvers already have the second timer; the first and third timers are new concepts for this mechanism.

When a request is received by the recursive resolver, it SHOULD start the client response timer. This timer is used to avoid client timeouts. It SHOULD be configurable, with a recommended value of 1.8 seconds.

The resolver then checks its cache for an unexpired answer. If it finds none and the Recursion Desired flag is not set in the request, it SHOULD immediately return the response without consulting the cache for expired records.

If iterative lookups will be done, it SHOULD start the query resolution timer. This timer bounds the work done by the resolver, and is commonly around 10 to 30 seconds.

If the answer has not been completely determined by the time the client response timer has elapsed, the resolver SHOULD then check its cache to see whether there is expired data that would satisfy the request. If so, it adds that data to the response message and SHOULD set the TTL of each expired record in the message to 1 second. The response is then sent to the client while the resolver continues its attempt to refresh the data. 1 second was chosen because historically 0 second TTLs have been problematic for some implementations. It not only sidesteps those potential problems with no practical negative consequence, it would also rate limit further queries from any client that is honoring the TTL, such as a forwarding resolver.

The maximum stale timer is used for cache management and is independent of the query resolution process. This timer is conceptually different from the maximum cache TTL that exists in many resolvers, the latter being a clamp on the value of TTLs as received from authoritative servers. The maximum stale timer SHOULD be configurable, and defines the length of time after a record expires that it SHOULD be retained in the cache. The suggested value is 7 days, which gives time to notice the resolution problem and for human intervention to fix it.

This same basic technique MAY be used to handle stale data associated with delegations. If authoritative server addresses are not able to be refreshed, resolution can possibly still be successful if the authoritative servers themselves are still up.

7. Implementation Caveats

Answers from authoritative servers that have a DNS Response Code of either 0 (NOERROR) or 3 (NXDOMAIN) MUST be considered to have refreshed the data at the resolver. In particular, this means that this method is not meant to protect against operator error at the authoritative server that turns a name that is intended to be valid into one that is non-existent, because there is no way for a resolver to know intent.

Resolution is given a chance to succeed before stale data is used to adhere to the original intent of the design of the DNS. This mechanism is only intended to add robustness to failures, and to be enabled all the time. If stale data were used immediately and then a cache refresh attempted after the client response has been sent, the resolver would frequently be sending data that it would have had no trouble refreshing.

It is important to continue the resolution attempt after the stale response has been sent, until the query resolution timeout, because some pathological resolutions can take many seconds to succeed as they cope with unavailable servers, bad networks, and other problems. Stopping the resolution attempt when the response with expired data has been sent would mean that answers in these pathological cases would never be refreshed.

Canonical Name (CNAME) records mingled in the expired cache with other records at the same owner name can cause surprising results. This was observed with an initial implementation in BIND when a hostname changed from having an IPv4 Address (A) record to a CNAME. The version of BIND being used did not evict other types in the cache when a CNAME was received, which in normal operations is not a significant issue. However, after both records expired and the

authorities became unavailable, the fallback to stale answers returned the older A instead of the newer CNAME.

[This probably applies to other occluding types, so more thought should be given to the overall issue.]

Keeping records around after their normal expiration will of course cause caches to grow larger than if records were removed at their TTL. Specific guidance on managing cache sizes is outside the scope of this document. Some areas for consideration include whether to track the popularity of names in client requests versus evicting by maximum age, and whether to provide a feature for manually flushing only stale records.

8. Implementation Status

[RFC Editor: per RFC 6982 this section should be removed prior to publication.]

The algorithm described in the Section 6 section was originally implemented as a patch to BIND 9.7.0. It has been in production on Akamai's production network since 2011, and effectively smoothed over transient failures and longer outages that would have resulted in major incidents. The patch was contributed to the Internet Systems Consortium and is now distributed with BIND 9.12.

Unbound has a similar feature for serving stale answers, but it works in a very different way by returning whatever cached answer it has before trying to refresh expired records. This is unfortunately not faithful to the ideal that data past expiry should attempt to be refreshed before being served.

9. Security Considerations

The most obvious security issue is the increased likelihood of DNSSEC validation failures when using stale data because signatures could be returned outside their validity period. This would only be an issue if the authoritative servers are unreachable, the only time the techniques in this document are used, and thus does not introduce a new failure in place of what would have otherwise been success.

Additionally, bad actors have been known to use DNS caches to keep records alive even after their authorities have gone away. This potentially makes that easier, although without introducing a new risk.

10. Privacy Considerations

This document does not add any practical new privacy issues.

11. NAT Considerations

The method described here is not affected by the use of NAT devices.

12. IANA Considerations

This document contains no actions for IANA. This section will be removed during conversion into an RFC by the RFC editor.

13. Acknowledgements

The authors wish to thank Matti Klock, Mukund Sivaraman, Jean Roy, and Jason Moreau for initial review. Feedback from Robert Edmonds and Davey Song has also been incorporated.

14. References

14.1. Normative References

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/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/info/rfc2119>.

[RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, DOI 10.17487/RFC2181, July 1997, <https://www.rfc-editor.org/info/rfc2181>.

[RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, DOI 10.17487/RFC6891, April 2013, <https://www.rfc-editor.org/info/rfc6891>.

14.2. Informative References

[RFC7719] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", RFC 7719, DOI 10.17487/RFC7719, December 2015, <https://www.rfc-editor.org/info/rfc7719>.

Authors' Addresses

David C Lawrence
Akamai Technologies
150 Broadway
Cambridge MA 02142-1054
USA

Email: tale@akamai.com

Warren Kumari
Google
1600 Amphitheatre Parkway
Mountain View CA 94043
USA

Email: warren@kumari.net