| | |
|---|---|
| Stream: | Internet Engineering Task Force (IETF) |
| RFC: | 8767 |
| Updates: | 1034, 1035, 2181 |
| Category: | Standards Track |
| Published: | March 2020 |
| ISSN: | 2070-1721 |
| Authors: | D. Lawrence | W. Kumari | P. Sood |
|---|---|---|---|
| | *Oracle* | *Google* | *Google* |

# RFC 8767
# Serving Stale Data to Improve DNS Resiliency

## Abstract

This document defines a method (serve-stale) for recursive resolvers to use stale DNS data to avoid outages when authoritative nameservers cannot be reached to refresh expired data. One of the motivations for serve-stale is to make the DNS more resilient to DoS attacks and thereby make them less attractive as an attack vector. This document updates the definitions of TTL from RFCs 1034 and 1035 so that data can be kept in the cache beyond the TTL expiry; it also updates RFC 2181 by interpreting values with the high-order bit set as being positive, rather than 0, and suggests a cap of 7 days.

## 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 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8767.

## Copyright Notice

## Table of Contents



# 1. Introduction

Traditionally, the Time To Live (TTL) of a DNS Resource Record (RR) 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 expands the definition of the TTL to explicitly 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 answer unavailability can cause outages even when the underlying data those servers would return is typically unchanged.

We describe a method below for this use of stale data, balancing the competing needs of resiliency and freshness.

This document updates the definitions of TTL from [RFC1034] and [RFC1035] so that data can be kept in the cache beyond the TTL expiry; it also updates [RFC2181] by interpreting values with the high-order bit set as being positive, rather than 0, and also suggests a cap of 7 days.

## 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

For a glossary of DNS terms, please see [RFC8499].

## 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 a recursive server is unable to contact the authoritative servers for a query 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." [RFC1035], Section 4.1.3 further says that 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 Section 8 of [RFC2181] was mostly concerned with the numeric range of values rather than data expiration behavior. It does, however, close that section by noting, "The TTL specifies a maximum time to live, not a mandatory time to live." This wording again does not contain BCP 14 key words [RFC2119], but it does convey the natural language connotation that data becomes unusable past TTL expiry.

As of the time of this writing, several large-scale operators use stale data for answers in some way. A number of recursive resolver packages, including BIND, Knot Resolver, OpenDNS, and Unbound, provide options to use stale data. Apple macOS can also use stale data as part of the Happy Eyeballs algorithms in mDNSResponder. The collective operational experience is that using stale data can provide significant benefit with minimal downside.

# 4. Standards Action

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

TTL a 32-bit unsigned integer number of seconds that specifies the duration 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 **SHOULD** be capped on the order of days to weeks, with a recommended cap of 604,800 seconds (7 days). If the data is unable to be authoritatively refreshed when the TTL expires, the record **MAY** be used as though it is unexpired. See Sections 5 and 6 of [RFC8767] for details.

Interpreting values that have the high-order bit set as being positive, rather than 0, is a change from [RFC2181], the rationale for which is explained in Section 6. Suggesting a cap of 7 days, rather than the 68 years allowed by the full 31 bits of Section 8 of [RFC2181], reflects the current practice of major modern DNS resolvers.

When returning a response containing stale records, a recursive resolver **MUST** set the TTL of each expired record in the message to a value greater than 0, with a **RECOMMENDED** value of 30 seconds. See Section 6 for explanation.

Answers from authoritative servers that have a DNS response code of either 0 (NoError) or 3 (NXDomain) and the Authoritative Answer (AA) bit set **MUST** be considered to have refreshed the data at the resolver. Answers from authoritative servers that have any other response code **SHOULD** be considered a failure to refresh the data and therefore leave any previous state intact. See Section 6 for a discussion.

# 5. Example Method

There is 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, four notable timers drive considerations for the use of stale data:

- 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 failure recheck timer, which limits the frequency at which a failed lookup will be attempted again.
- A maximum stale timer, which caps the amount of time that records will be kept past their expiration.

Most recursive resolvers already have the query resolution timer and, effectively, some kind of failure recheck timer. The client response timer and maximum stale timer are new concepts for this mechanism.

When a recursive resolver receives a request, 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 as being just under a common timeout value of 2 seconds while still giving the resolver a fair shot at resolving the name.

The resolver then checks its cache for any unexpired records that satisfy the request and returns them if available. If it finds no relevant unexpired data and the Recursion Desired flag is not set in the request, it should immediately return the response without consulting the cache for expired records. Typically, this response would be a referral to authoritative nameservers covering the zone, but the specifics are implementation dependent.

If iterative lookups will be done, then the failure recheck timer is consulted. Attempts to refresh from non-responsive or otherwise failing authoritative nameservers are recommended to be done no more frequently than every 30 seconds. If this request was received within this period, the cache may be immediately consulted for stale data to satisfy the request.

Outside the period of the failure recheck timer, the resolver should start the query resolution timer and begin the iterative resolution process. This timer bounds the work done by the resolver when contacting external authorities and is commonly around 10 to 30 seconds. If this timer expires on an attempted lookup that is still being processed, the resolution effort is abandoned.

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 with a TTL greater than 0 (as specified in Section 4). The response is then sent to the client while the resolver continues its attempt to refresh the data.

When no authorities are able to be reached during a resolution attempt, the resolver should attempt to refresh the delegation and restart the iterative lookup process with the remaining time on the query resolution timer. This resumption should be done only once per resolution effort.

Outside the resolution process, 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 and recommended to be 7 days in the TTL definition in Section 4. The maximum stale timer should be configurable. It defines the length of time after a record expires that it should be retained in the cache. The suggested value is between 1 and 3 days.

# 6. Implementation Considerations

This document mainly describes the issues behind serving stale data and intentionally does not provide a formal algorithm. The concept is not overly complex, and the details are best left to resolver authors to implement in their codebases. The processing of serve-stale is a local operation, and consistent variables between deployments are not needed for interoperability. However, we would like to highlight the impact of various implementation choices, starting with the timers involved.

The most obvious of these is the maximum stale timer. If this variable is too large, it could cause excessive cache memory usage, but if it is too small, the serve-stale technique becomes less effective, as the record may not be in the cache to be used if needed. Shorter values, even less than a day, can effectively handle the vast majority of outages. Longer values, as much as a week, give time for monitoring systems to notice a resolution problem and for human intervention to fix it; operational experience has been that sometimes the right people can be hard to track down and unfortunately slow to remedy the situation.

Increased memory consumption could be mitigated by prioritizing removal of stale records over non-expired records during cache exhaustion. Eviction strategies could consider additional factors, including the last time of use or the popularity of a record, to retain active but stale records. A feature to manually flush only stale records could also be useful.

The client response timer is another variable that deserves consideration. If this value is too short, there exists the risk that stale answers may be used even when the authoritative server is actually reachable but slow; this may result in undesirable answers being returned. Conversely, waiting too long will negatively impact user experience.

The balance for the failure recheck timer is responsiveness in detecting the renewed availability of authorities versus the extra resource use for resolution. If this variable is set too large, stale answers may continue to be returned even after the authoritative server is reachable; per [RFC2308], Section 7, this should be no more than 5 minutes. If this variable is too small, authoritative servers may be targeted with a significant amount of excess traffic.

Regarding the TTL to set on stale records in the response, historically TTLs of 0 seconds have been problematic for some implementations, and negative values can't effectively be communicated to existing software. Other very short TTLs could lead to congestive collapse as TTL-respecting clients rapidly try to refresh. The recommended value of 30 seconds not only sidesteps those potential problems with no practical negative consequences, it also rate-limits further queries from any client that honors the TTL, such as a forwarding resolver.

As for the change to treat a TTL with the high-order bit set as positive and then clamping it, as opposed to [RFC2181] treating it as zero, the rationale here is basically one of engineering simplicity versus an inconsequential operational history. Negative TTLs had no rational intentional meaning that wouldn't have been satisfied by just sending 0 instead, and similarly there was realistically no practical purpose for sending TTLs of 2^25 seconds (1 year) or more. There's also no record of TTLs in the wild having the most significant bit set in the DNS

Operations, Analysis, and Research Center's (DNS-OARC's) "Day in the Life" samples [DITL]. With no apparent reason for operators to use them intentionally, that leaves either errors or non-standard experiments as explanations as to why such TTLs might be encountered, with neither providing an obviously compelling reason as to why having the leading bit set should be treated differently from having any of the next eleven bits set and then capped per Section 4.

Another implementation consideration is the use of stale nameserver addresses for lookups. This is mentioned explicitly because, in some resolvers, getting the addresses for nameservers is a separate path from a normal cache lookup. If authoritative server addresses are not able to be refreshed, resolution can possibly still be successful if the authoritative servers themselves are up. For instance, consider an attack on a top-level domain that takes its nameservers offline; serve-stale resolvers that had expired glue addresses for subdomains within that top-level domain would still be able to resolve names within those subdomains, even those it had not previously looked up.

The directive in Section 4 that only NoError and NXDomain responses should invalidate any previously associated answer stems from the fact that no other RCODEs that a resolver normally encounters make any assertions regarding the name in the question or any data associated with it. This comports with existing resolver behavior where a failed lookup (say, during prefetching) doesn't impact the existing cache state. Some authoritative server operators have said that they would prefer stale answers to be used in the event that their servers are responding with errors like ServFail instead of giving true authoritative answers. Implementers **MAY** decide to return stale answers in this situation.

Since the goal of serve-stale is to provide resiliency for all obvious errors to refresh data, these other RCODEs are treated as though they are equivalent to not getting an authoritative response. Although NXDomain for a previously existing name might well be an error, it is not handled that way because there is no effective way to distinguish operator intent for legitimate cases versus error cases.

During discussion in the IETF, it was suggested that, if all authorities return responses with an RCODE of Refused, it may be an explicit signal to take down the zone from servers that still have the zone's delegation pointed to them. Refused, however, is also overloaded to mean multiple possible failures that could represent transient configuration failures. Operational experience has shown that purposely returning Refused is a poor way to achieve an explicit takedown of a zone compared to either updating the delegation or returning NXDomain with a suitable SOA for extended negative caching. Implementers **MAY** nonetheless consider whether to treat all authorities returning Refused as preempting the use of stale data.

## 7. Implementation Caveats

Stale data is used only when refreshing has failed in order to adhere to the original intent of the design of the DNS and the behavior expected by operators. If stale data were to always be 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. Because modern resolvers use techniques like prefetching and request coalescing for efficiency, it is not

necessary that every client request needs to trigger a new lookup flow in the presence of stale data, but rather that a good-faith effort has been recently made to refresh the stale data before it is delivered to any client.

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.

The continuing prohibition against using data with a 0-second TTL beyond the current transaction explicitly extends to it being unusable even for stale fallback, as it is not to be cached at all.

Be aware that Canonical Name (CNAME) and DNAME records [RFC6672] 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.

## 8. Implementation Status

The algorithm described in Section 5 was originally implemented as a patch to BIND 9.7.0. It has been in use on Akamai's production network since 2011; it 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 the functionality is now available in BIND 9.12 and later via the options stale-answer-enable, stale-answer-ttl, and max-stale-ttl.

Unbound has a similar feature for serving stale answers and will respond with stale data immediately if it has recently tried and failed to refresh the answer by prefetching. Starting from version 1.10.0, Unbound can also be configured to follow the algorithm described in Section 5. Both behaviors can be configured and fine-tuned with the available serve-expired-* options.

Knot Resolver has a demo module here: <https://knot-resolver.readthedocs.io/en/stable/modules-serve_stale.html>.

Apple's system resolvers are also known to use stale answers, but the details are not readily available.

In the research paper "When the Dike Breaks: Dissecting DNS Defenses During DDoS" [DikeBreaks], the authors detected some use of stale answers by resolvers when authorities came under attack. Their research results suggest that more widespread adoption of the technique would significantly improve resiliency for the large number of requests that fail or experience abnormally long resolution times during an attack.

## 9. EDNS Option

During the discussion of serve-stale in the IETF, it was suggested that an EDNS option [RFC6891] should be available. One proposal was to use it to opt in to getting data that is possibly stale, and another was to signal when stale data has been used for a response.

The opt-in use case was rejected, as the technique was meant to be immediately useful in improving DNS resiliency for all clients.

The reporting case was ultimately also rejected because even the simpler version of a proposed option was still too much bother to implement for too little perceived value.

## 10. 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. Stale negative records can increase the time window where newly published TLSA or DS RRs may not be used due to cached NSEC or NSEC3 records. These scenarios would only be an issue if the authoritative servers are unreachable (the only time the techniques in this document are used), and thus serve-stale 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. The serve-stale feature potentially makes the attack easier, although without introducing a new risk. In addition, attackers could combine this with a DDoS attack on authoritative servers with the explicit intent of having stale information cached for a longer period of time. But if attackers have this capacity, they probably could do much worse than prolonging the life of old data.

In [CloudStrife], it was demonstrated how stale DNS data, namely hostnames pointing to addresses that are no longer in use by the owner of the name, can be used to co-opt security -- for example, to get domain-validated certificates fraudulently issued to an attacker. While this document does not create a new vulnerability in this area, it does potentially enlarge the window in which such an attack could be made. A proposed mitigation is that certificate authorities should fully look up each name starting at the DNS root for every name lookup. Alternatively, certificate authorities should use a resolver that is not serving stale data.

## 11. Privacy Considerations

This document does not add any practical new privacy issues.

## 12. NAT Considerations

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

# 13. IANA Considerations

This document has no IANA actions.

# 14. References

## 14.1. Normative References

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

**[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>.

**[RFC2308]** Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", RFC 2308, DOI 10.17487/RFC2308, March 1998, <https://www.rfc-editor.org/info/rfc2308>.

**[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/info/rfc8174>.

## 14.2. Informative References

**[CloudStrife]** Borgolte, K., Fiebig, T., Hao, S., Kruegel, C., and G. Vigna, "Cloud Strife: Mitigating the Security Risks of Domain-Validated Certificates", DOI 10.1145/3232755.3232859, ACM 2018 Applied Networking Research Workshop, July 2018, <https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_06A-4_Borgolte_paper.pdf>.

**[DikeBreaks]** Moura, G.C.M., Heidemann, J., Müller, M., Schmidt, R. de O., and M. Davids, "When the Dike Breaks: Dissecting DNS Defenses During DDoS", DOI 10.1145/3278532.3278534, ACM 2018 Internet Measurement Conference, October 2018, <https://www.isi.edu/~johnh/PAPERS/Moura18b.pdf>.

**[DITL]** DNS-OARC, "DITL Traces and Analysis", January 2018, <https://www.dns-oarc.net/oarc/data/ditl>.

**[RFC6672]** Rose, S. and W. Wijngaards, "DNAME Redirection in the DNS", RFC 6672, DOI 10.17487/RFC6672, June 2012, <https://www.rfc-editor.org/info/rfc6672>.

**[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>.

**[RFC8499]** Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", BCP 219, RFC 8499, DOI 10.17487/RFC8499, January 2019, <https://www.rfc-editor.org/info/rfc8499>.

## Acknowledgements

The authors wish to thank Brian Carpenter, Vladimir Cunat, Robert Edmonds, Tony Finch, Bob Harold, Tatuya Jinmei, Matti Klock, Jason Moreau, Giovane Moura, Jean Roy, Mukund Sivaraman, Davey Song, Paul Vixie, Ralf Weber, and Paul Wouters for their review and feedback. Paul Hoffman deserves special thanks for submitting a number of Pull Requests.

Thank you also to the following members of the IESG for their final review: Roman Danyliw, Benjamin Kaduk, Suresh Krishnan, Mirja Kühlewind, and Adam Roach.

## Authors' Addresses

**David C Lawrence**
Oracle
Email: tale@dd.org

**Warren "Ace" Kumari**
Google
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States of America
Email: warren@kumari.net

**Puneet Sood**
Google
Email: puneets@google.com