Network Working Group
Internet-Draft
Intended status: Informational
Expires: December 29, 2017

W. Kumari
Google
R. Arends
ICANN
S. Woolf

D. Migault
Ericsson

June 27, 2017

Highly Automated Method for Maintaining Expiring Records
draft-wkumari-dnsop-hammer-03

Abstract

This document describes a simple DNS cache optimization which keeps the most popular Resource Records set (RRset) in the DNS cache: Highly Automated Method for Maintaining Expiring Records (HAMMER).

The principle is that popular RRset in the cache are fetched, that is to say resolved before their TTL expires and flushed. By fetching RRset before they are being queried by an end user, that is to say prefetched, HAMMER is expected to improve the quality of experience of the end users as well as to optimize the resources involved in large DNSSEC resolving platforms.

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 http://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 December 29, 2017.

Table of Contents

1. Introduction

A recursive DNS resolver may cache a Resource Record set (RRset) for, at most, the Time To Live (TTL) associated with that RRset. While the TTL is greater than zero, the resolver may respond to queries from its cache; but once the TTL has reached zero, the resolver flushes the RRset. When the resolver gets another query for that RRset, the RRset is not anymore in the cache, thus the resolver need

to proceed to a new resolution for that RRset with its associated latency and processing. The resolved RRset are then cached and returned to the original querying client. This document discusses an optimization (Highly Automated Method for Maintaining Expiring Records -- (HAMMER), also known as "prefetch") to help keep popular responses in the cache, by fetching (or resolving) resources before their TTL expires.

In that document, a resolver implementing HAMMER (HAMMER resolver) prefetches a RRset candidate to HAMMER (HAMMER RRset) when it receives a query and its TTL is lower than HAMMER TIME.

Note that [RFC4035] assumes that all RR of a RRset have the same TTL, while [RFC2181] allows the TTL of the RR of a RRset to be different. When the RRset does not follow [RFC4035], the TTL of the RRset that is considered is the minimum value of the TTL.

1.1. Requirements notation

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

2. Terminology

HAMMER resolver: A DNS resolver that implements HAMMER mechanism.

HAMMER RRset: A RRset that is a candidate for the HAMMER process.

HAMMER TIME: a number of seconds that indicates the period preceding the TTL expiration time. When a query for a HAMMER RRset is received during that period, the HAMMER resolver prefetches the HAMMER RRset by initiating a resolution.

3. Motivations

When a recursive resolver responds to a client, it either responds from cache, or it initiates an iterative query to resolve the answer, caches the answer and then responds with that answer.

3.1. Improving browsing Quality of Experience by reducing response time

Any end user querying a fetched RRset will get the response from the cache of the resolver. This provides faster responses, thus improving the end user experience for browsing and other applications/activities.

Popular RRsets are highly queried, and end users have high expectations in terms of application responsiveness for these RRsets. With regular DNS rules, once the RRset has been flushed from the cache, it waits for the next end user to request the RRset before initiating a resolution for this given RRset with iterative queries. This results in at least one end user waiting for this resolution to be performed over the Internet before the response is sent to them. This may provide a poor user experience since DNS response times over the Internet are unpredictable at best and it provides a response time longer then usual. Note that the response time may also be increased by the use of DNSSEC since DNSSEC may involve additional resolutions, larger payloads, and signature checks.

In addition to that first end user querying RRset after it has been flushed, end users querying the RRset during its resolution or fetching phase are also impacted. As a result, especially for popular RRsets, multiple end users are likely to be impacted and be provided a poor user experience.

The impact on users also depends on the architecture of resolving platform architecture. In some case, a centralized resolver is implemented as a farm of independent resolving nodes and the traffic is split between the nodes according to the IP addresses and ports. In such architectures, the number of affected end users is proportional to the number of resolving nodes as each query is pseudo-randomly associated to one of the resolving node. Similarly, some global resolving platform uses anycast to steer the queries to the resolving node associated with the shortest path. Unless all queries comes from a single region, such architecture are also expected to impact a number of user proportional to the number of resolving nodes.

3.2. Optimize the resources involved in large DNSSEC resolving platforms

As mentioned in Section 3.1, large resolving platforms are often composed of a set of independent resolving nodes in order to distribute the traffic between these nodes. Traffic can be distributed using various forms of load balancing between the resolving nodes. This includes, for example, a pseudo-random distribution when load balancing is based on the hash of the IP addresses and ports or shortest path when anycast is deployed. Such distributions split the traffic independently of the queried RRset. Ignoring the coordination of the resolutions implicitly assumes the resource to perform the resolutions is negligible compared to those necessary to handle the queries of the end users.

As a result, such platforms perform multiple parallel resolutions on their various nodes. With DNS, the necessary resource associated to the resolution were in fact minimal so little effort have been considered to synchronize these nodes in order to reduce the number of resolutions. On the other hand DNSSEC resolutions involve additional resolutions, larger payloads and signature checks. The consequent increase of resource to perform DNSSEC resolutions versus DNS resolution makes parallel resolutions a non negligible lost of resource and leave place for synchronization mechanisms.

One way to reduce the number of DNSSEC resolutions is to prefetch (or provision) the nodes with the most popular RRsets before their TTL expire. Note that in this case, the resolution is not performed by the resolving node. At a node level, prefetching increases the nodes availability. At the platform level, synchronizing the resolving nodes' resolution globally reduce the number of resolution and so the overall resource of the platform.

Synchronization of the resolution may be performed by configuring each node as a forwarder for these RRsets. This avoids parallel resolutions and overall reduces cost, because signature checks are not performed by each resolving node. In this case prefetching enables to still benefit from the already existing load balancing architecture that split the load of the end users' queries traffic between the nodes. Note that the advantages of synchronizing the resolutions between the resolving nodes may depend on the popularity of the RRsets. This architecture takes advantage of the Zipf [ZIPF] distribution of the RRsets' popularity. In fact, a few number of RRsets need to be cached (a few thousands) to address most of the traffic (up to 70%) [PREFETCH].

4. Protocol Description

This section describes HAMMER. This section is not normative and implementation may implement this mechanism with their own flavor.

When a recursive resolver that implements HAMMER receives a query for a HAMMER RRset that it has in the cache, it responds from the cache.

If the queried RRset is a HAMMER RRset, the HAMMER resolver compares the TTL value to the HAMMER TIME, as well as if the RRset is being (pre)fetched.

If the HAMMER RRset has a TTL greater then the HAMMER TIME, nothing is done.

If the HAMMER RRset has a TTL less than the HAMMER TIME, the HAMMER resolver starts a resolution for the RRset in order to fill the

cache, just as if the TTL had expired. The HAMMER RRset is prefetched. Note that during the resolution, the HAMMER RRset is still cached, and queries are responded form the cache until the TTL expires. Once the resolution is performed, the freshly resolved RRset replace the existing cached RRset. This ensures the cache has fresh data for subsequent queries.

Since prefetching is initiated before the existing cached entry expires (and is flushed), responses will come from the cache more often. This decreases the client resolution latency and improves the user experience.

Prefetching is triggered by an incoming query (and only if that query arrives shortly before the record would expire anyway). This effectively keeps the most popular RRsets uniformly queried in the cache, without having to maintain counters in the cache or proactively resolve responses that are not likely to be needed as often. This is purely an implementation optimization - resolvers always have the option to cache records for less than the TTL (for example, when running low on cache space, etc), this simply triggers a refresh of the RRset before it expires.

Note that non-uniformly queried RRsets may be popular and may not benefit from the HAMMER mechanism. For example, a RRset MAY be heavily queried the first 10 minutes of every hour with a 30 minute TTL. In that case DNS queries are not expected to come between TTL - HAMMER TIME and TTL.

HAMMER RRset with small TTL may generate a prefetching process even though they are not so popular. Suppose an end user is setting a specific session which requires multiple DNS resolutions on a given FQDN. These resolutions are necessary for a short period of time, i.e. the necessary time to establish the session. If these RRset have been set with a small TTL - in the order of the time session establishment - the multiple queries to a HAMMER resolver may trigger an unnecessary resolution. As a result HAMMER would not scale thousands of these RRsets. As a result, if the original TTL of the RRset is less than (or close to HAMMER TIME), the described method could cause excessive prefetching queries to occur. In order to prevent this an additional variable named STOP (described below) is introduced. If the original TTL of the RRset is less than STOP * HAMMER TIME then the cache entry should be marked with a "Can't touch this" flag, and the described method should not be used.

5. Configuration Variables

These are the mandatory variables:

HAMMER TIME: a number of seconds that indicates the period preceding the TTL expiration time. When a query for a HAMMER RRset is received during that period, the HAMMER resolver prefetches the HAMMER RRset by initiating a resolution. A default of 2 seconds is RECOMMENDED.

STOP: should be a user configurable variable. A default of 3 is recommended.

Implementations may consider additional variables. These are not mandatory but would address specific use of the HAMMER.

HAMMER MATCH: should be a user configurable variable. It defines RRsets that are expected to implement HAMMER. This rule can be expressed in different ways. It can be a list of RRsets, or a number indicating the number of most popular RRsets that needs to be considered. How HAMMER MATCH is expressed is implementation dependent. Implementations can use a list of FQDNs, others can use a matching rule on the RRsets, or define the HAMMER RRsets as the X most popular RRsets.

HAMMER FORWARDER: should be a user configurable variable. It is optional and designates the DNS server the resolver forwards the request to.

6. IANA Considerations

This document makes no request of the IANA.

7. Security Considerations

This technique leverages existing protocols, and should not introduce any new risks, other than a slight increase in traffic.

By initiating cache fill entries before the existing RR has expired this technique will slightly increase the number of queries seen by authoritative servers. This increase will be inversely proportional to the average TTL of the records that they serve.

It is unlikely, but possible, that this increase could cause a denial of service condition.

8. Acknowledgements

The authors wish to thank Tony Finch and MC Hammer. We also wish to thank Brian Somers and Wouter Wijngaards for telling us that they already do this :-) (They should probably be co-authors, but I left this too close to the draft cutoff time to confirm with them that they are willing to have their names on this).

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://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, <http://www.rfc-editor.org/info/rfc2181>.

[RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, DOI 10.17487/RFC4035, March 2005, <http://www.rfc-editor.org/info/rfc4035>.

9.2. Informative References

[PREFETCH] Migault, D., Francfort, S., Senecal, S., Herbert, E., and M. Laurent, "PREFETCHing to optimize DNSSEC deployment over large Resolving Platforms", Jul 2013, <https://www.researchgate.net/publication/270571591_PREFETCHing_to_optimize_DNSSEC_deployment_over_large_Resolving_Platforms>.

[ZIPF] Powers, D., "Applications and Explanations of Zipf's Law", Jan 1998, <http://aclweb.org/anthology/W98-1218>.

Appendix A. Known implementations

[RFC Editor: Please remove this section before publication]

[Ed: Well, this is kinda embarrassing. This idea occurred to us one day while sitting around a pool in New Hampshire. It then took a while before I wrote it down, mostly because I *really* wanted to get "Stop! Hammer Time!" into a draft. Anyway, we presented it in Berlin, and Wouter Wijngaards stood up and mentioned that Unbound

already does this (they use a percentage of TTL, instead of a number of seconds). Then we heard from OpenDNS that they *also* implement something similar. Then we had a number of discussions, then got sidetracked into other things. Anyway, BIND as of 9.10, around Feb 2014 now implements something like this (https://deepthought.isc.org/article/AA-01122/0/Early-refresh-of-cache-records-cache-prefetch-in-BIND-9.10.html), and enables it by default. Unfortunately, while BIND uses the times based approach, they named their parameters "trigger" and "eligibility" - and shouting "Eligibility! Trigger time!" simply isn't funny (unless you have a very odd sense of humor... So, we are now documenting implementations that existed before this was published and an impl,entation that we think was based on this. We think that this has value to the community. I'm also leaving in the HAMMER TIME bit, because it makes me giggle. This below section should be filled out with more detail, in collaboration with the implementors, but this is being written *just* before the draft cutoff.].

A number of recursive resolvers implement techniques similar to the techniques described in this document. This section documents some of these and tradeoffs they make in picking their techniques.

A.1. Unbound (NLNet Labs)

The Unbound validating, recursive, and caching DNS resolver implements a HAMMER type feature, called "prefetch". This feature can be enabled or disabled though the configuration option "prefetch: <yes or no>". When enabled, Unbound will fetch expiring records when their remaining TTL is less than 10% of their original TTL.

[Ed: Unbound's "prefetch" function was developed independently, before this draft was written. The authors were unaware of it when writing the document.]

A.2. OpenDNS

The public DNS resolver, OpenDNS implements a prefetch like solution.

[Ed: Will work with OpenDNS to get more details.]

A.3. ISC BIND

As of version 9.10, Internet Systems Consortium's BIND implements the HAMMER functionality. This feature is enabled by default.

The functionality is configured using the "prefetch" options statement, with two parameters:

Trigger This is equivalent to the HAMMER_TIME parameter described below.

Eligibility This is equivalent to the STOP parameter described below.

Appendix B. Changes / Author Notes.

[RFC Editor: Please remove this section before publication]

From -01 to -02:

- Readbility / cleanup.
- Tried to make it more clear that most implementations now support this (although they call it "prefetch")

From -00 to 01:

- Fairly large rewrite.
- Added text on the fact that there are implmentations that do this.
- Added the "prefetch" name, cleaned up some readability.
- Daniel's test (Section 3.2) added.

From -template to -00.

- Wrote some text.
- Changed the name.

Authors' Addresses

Warren Kumari
Google
1600 Amphitheatre Parkway
Mountain View, CA 94043
US

Email: warren@kumari.net

Roy Arends
ICANN

Email: roy.arends@icann.org

Suzanne Woolf
39 Dodge St. #317
Beverly, MA 01915
US

Email: suzworldwide@gmail.com

Daniel Migault
Ericsson
2039 Rue Cohen
Saint-Laurent H4R 2A4
Canada

Email: daniel.migaultf@ericsson.com