Thinking about Serve Stale

Cathy Almond

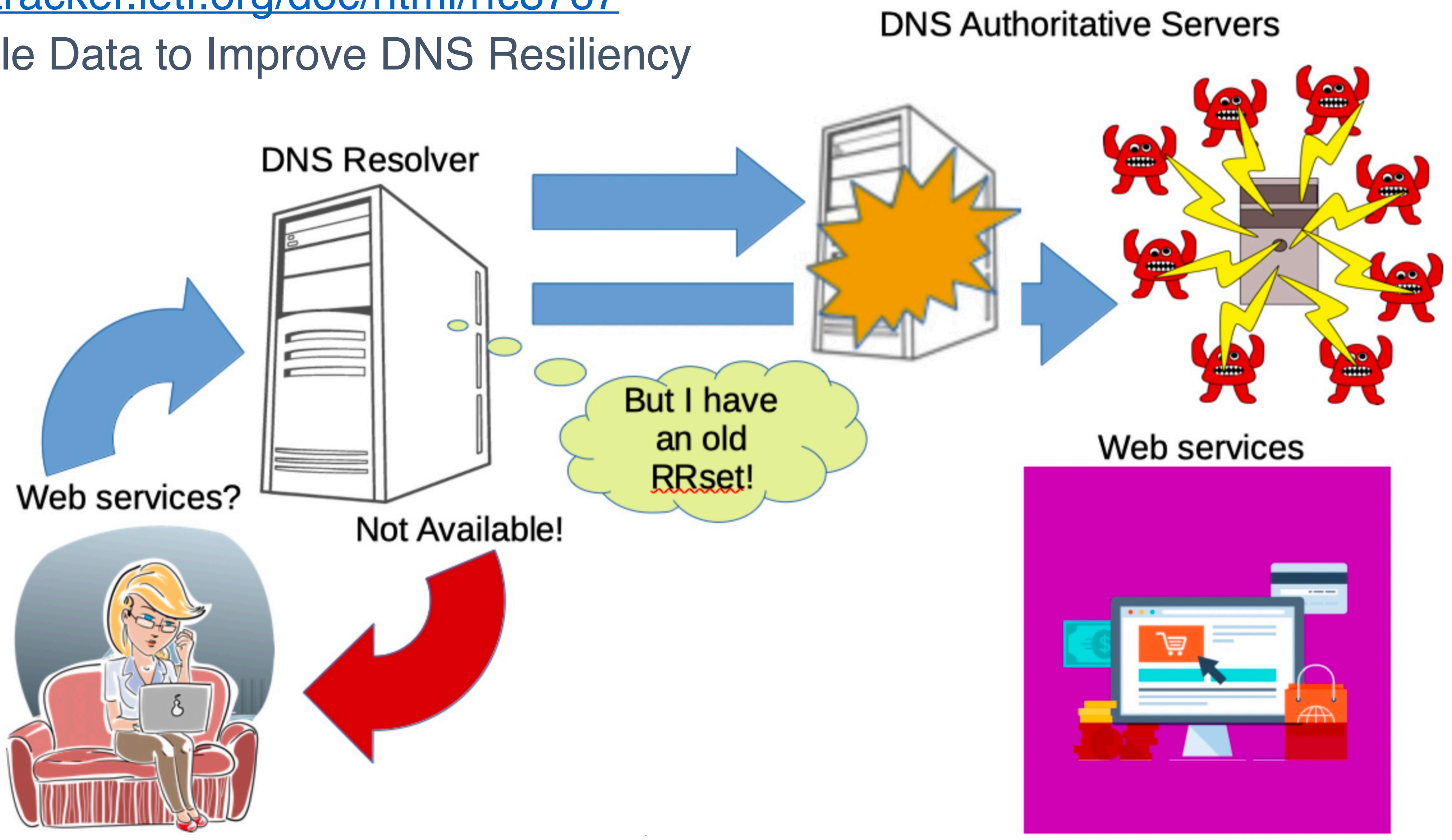
2025-02-06

https://www.isc.org

What is Serve Stale anyway?

https://datatracker.ietf.org/doc/html/rfc8767

Serving Stale Data to Improve DNS Resiliency



How might this work?

- How long do you want to keep ‘stale’ cache content?
- How long do you keep a client waiting before providing a stale answer?
- How often do you try to refresh stale content?
- Should you give up eventually and stop serving stale RRsets?
- Should you indicate to the client that the answer is old data?

RFC 8767 says:

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.

RFC 8767 also talks about:

- What TTL should you put on a stale RRset?
- Should you ever serve stale RRsets that were originally received with TTL=0 ('don't cache')?
- Also use stale content in-path to getting the answers for the client query (such as NS records and addresses?)
- Does receipt of a non-authoritative error (e.g. FORMERR, SERVFAIL, REFUSED) count as a failed refresh?
- Can you signal that stale answers have been provided

Phew! Let’s talk about actually using it!

Two main things for operators to consider when enabling Serve Stale:

1. Cache - how long can/should you keep stale RRsets in cache and will there be any memory consumption implications of this for your server(s)
2. User experience - how long should clients wait before being served with stale content, and how often should they expect their resolver to attempt to refresh the old content

How to enable and configure Serve-Stale

What you need to think about and how to configure Serve Stale in:

- BIND
- Unbound
- Knot Resolver
- PowerDNS Recursor

Enabling Serve Stale in BIND

- Enable the stale cache (off by default in all currently supported versions):

```
stale-cache-enable yes;
```

- Configure how long to retain stale content (default 1 day):

```
max-stale-ttl 24H;
```

- Are you going to enable serving of stale answers now, or turn the feature on/off when required?

```
stale-answer-enable yes;
```

  or

```
rndc serve-stale on|off|reset
```

  Note that you can't enable stale answers without first starting BIND with stale cache enabled - no stale cache means no stale answers available.

(Optional) Configuring stale answers in BIND

These options are only effective when both stale-cache-enable and stale-answers-enable are set to ‘yes’.

- Configure the TTL used for stale RRsets in query responses (default 30s):

```
stale-answer-ttl 30s;
```

- How long to serve stale before reattempting to refresh (default 30s):

```
stale-refresh-time 30s;
```

- How long do clients have to wait for a stale answer instead of the usual recursion timeout and SERVFAIL?

```
stale-answer-client-timeout <duration>;
```

The duration above can be either zero (‘0’) or disabled/off (default off). Recursion duration is controlled by option ‘resolver-query-timeout’, default 10s. ***ISC support recommends not altering this!***

BIND and stale answers - be aware that:

- BIND's cache content management strategy is two-fold. Opportunistic cache cleaning means that in-passing, TTL-expired content is removed. To enable stale cache, most RRsets that would otherwise be candidates for eviction, are now retained (if possible) for the period max-stale-ttl. Therefore your cache memory consumption may increase significantly, up to the point that memory-based (least recently used) cache cleaning is triggered.
- Content received with TTL=0 is not retained for use in stale answers, *unless* option min-cache-ttl has been used to override small or zero TTLs on received RRsets.
- NXDOMAIN RRsets are not retained for use in stale answers (this differs from other implementations).
- BIND automatically adds EDE options 3 (Stale Answer) or 19 (Stale NXDOMAIN) to query responses with stale content (although see above!).
- A SERVFAIL answer due to fetch-limits may instead be replaced with stale content but doesn't trigger stale-refresh-time.

Enabling Serve Stale in Unbound

- Enable the serving of stale content (disabled by default):

```
serve-expired: yes
```

- Enable prefetch (to help keep popular cache up-to-date):

```
prefetch: yes
```

- How long do you want stale RRsets to continue to be used for serving stale answers (default is all content is eligible indefinitely; recommendation is 1 day)?

```
serve-expired-ttl: 86400  # 1 day (in seconds)
```

- Decide whether or not you want automatic extension of use of expired stale RRsets (default no)

```
serve-expired-ttl-reset: yes|no
```

This setting extends/resets the "use this stale until" timestamp added to stale RRsets in cache when a refresh attempt fails. Without it, content is only used stale until the limit specified in serve-expired-ttl.

(Optional) Configuring stale answers in Unbound

- Configure the TTL used for stale RRsets in query responses (default 30s):

```
serve-expired-reply-ttl: 30  # 30 seconds (in seconds)
```

- How long do clients have to wait for a stale answer instead of the usual recursion timeout and SERVFAIL?

```
serve-expired-client-timeout: 1800  #1.8 seconds (in
milliseconds)
```

The above example is deliberately just shorter than most client-side query response timeouts of 2 seconds, but the default is zero - respond stale first and attempt to refresh afterwards.

- Configure whether or not you want to send Extended DNS Errors (EDE) in query responses containing stale RRsets:

```
ede: yes
ede-serve-expired: yes
```

Unbound and stale answers - useful to know:

- Unbound's cache content management strategy is based on eviction of RRsets once cache memory limits are reached and on a Least Recently Used (LRU) basis. Enabling Serve Stale therefore is unlikely to change cache composition or memory consumption.
- Unbound has an additional back-end cache DB feature available which may change how Serve Stale operates - see https://unbound.docs.nlnetlabs.nl/en/latest/manpages/unbound.conf.html#cache-db-module-options
- Content received with TTL=0 is not retained for use in stale answers (because it is never added to cache) *unless* options cache-min-ttl and/or cache-min-negative-ttl have been used to override small or zero TTLs on received RRsets.

Unbound and stale answers - useful to know:

- There are no options to directly control how frequently Unbound attempts to refresh stale content before using it again; client queries will continue to use the stale content without initiating a new refresh until 5 seconds have passed since the last failure.
- The client experience with serve-expired-ttl-reset: yes can vary because although a failed refresh will reset the serve-expired-ttl timer on the RRset, if that RRset was no longer eligible for stale use *before* the refresh, it won't be sent to the client until *after* the refresh attempt has finished processing and has failed (serve-expired-client-timeout isn't used for non-eligible stale content).

Enabling Serve Stale in Knot Resolver

- Enable the serving of stale content (disabled by default):

```
options:
  serve-stale: true
```

This setting enables the use of stale RRsets in order to avoid sending SERVFAIL when cache content can't be refreshed. They are usable for up to 24 hours after TTL expiry. This stale validity period is not directly configurable.

(Optional) Configuring stale answers in Knot Resolver

- Adjust how long to wait before retrying unreachable servers (default 1000 ms):

```
cache:
  ns-timeout: <time ms|s|m|h|d>
```

As long as as all nameservers are marked unreachable and have been for less than ns-timeout, clients will receive immediate stale answers without an attempt to refresh first.

Knot Resolver and stale answers - need to know:

- Knot Resolver’s cache content management strategy is based on eviction of RRsets once cache memory limits are reached. Enabling serve-stale therefore is unlikely to change cache composition or memory consumption.
- Stale RRsets are returned in query responses with a 1 second TTL.
- There is no directly-configurable client query/resolver timeout.
- It’s nevertheless possible to change some timers (e.g. .timeout and .callback) by directly modifying the lua module: https://gitlab.nic.cz/knot/knot-resolver/-/blob/master/modules/serve_stale/serve_stale.lua
- Content received with TTL=0 **is** eligible to be used in stale answers *even if* Knot Resolver option cache/ttl-min (default 5s) is set to zero.
- Knot Resolver (version 6 and up) automatically adds EDE options 3 or 19 to query responses with stale content.

Enabling Serve Stale in PowerDNS Recursor

- Enable the serving of stale content (disabled by default) by choosing how many times to reset a stale RRset's TTL in the configuration or command line:

  ```
  serve-stale-extensions <count>
  ```

  This setting defines how many times a stale RRset can be 'revived' in cache by giving it a TTL that is the smaller of 30s or the original TTL of the RRset.

  The 'revival' only happens after the RRset refresh attempt fails.

  The 'revival' mechanism takes into account how long ago the last extension was done when doing a new extension. RRsets that are only queried once in a while get the same maximum stale period as compared with RRsets queried very frequently (assuming original TTL>= 30s).

PowerDNS and stale answers - useful to know:

- PowerDNS Recursor's cache content management strategy is based both on eviction of expired content and on eviction of RRsets based on LRU once cache memory limits are reached. Enabling Serve Stale defers eviction of expired content until serve-stale-extensions x 30s. Therefore your cache memory consumption will increase and there may be more LRU cleaning occurring.
- Content received with TTL=0 **is** eligible to be used in stale answers *even if* PowerDNS Recursor option minimum-ttl-override (default 1s) is set to zero.
- There is no specific configurable timer for how long to wait for on a refresh attempt before serving stale content; the time to wait is the same, irrespective of whether PowerDNS would respond with SERVFAIL or use stale RRsets.
- The TTL on RRsets used stale is 30s or less, and counts down in cache in the same way as the original RRset's TTL would have done.
- PowerDNS does not (yet) add EDE options to query responses with stale content, but this is planned as a future feature.

Other resolver implementations exist, for example:

- Appliances that use Open Source DNS ‘inside the box’ will almost certainly enable Serve Stale and provide some configuration options.
- Akamai DNS has a switch (default ‘on’) for serving stale content - it is an extension to prefetch functionality for ‘popular’ content
- Cloud-based resolver solutions appear to implement some types of ‘failure mode’ recovery, and this probably includes serving of stale content (*disclaimer - I have not researched this!*)

Not covered in this presentation…

- Logging and statistics (mostly included in software providers' documentation).
- Serve Stale in relation to DNSSEC validation failures and stale DNSSEC material.
- Use of stale cache data when following intermediate referrals.
- Expired cache containing both CNAME/DNAME and other records for the same name (due to different query responses at different times).
- "Fail" responses (as opposed to failure to respond by authoritative servers) - which of these trigger Serve Stale?

It's reasonable to assume that the software implementations handle all of the above sanely - but if you need to know - ask your software provider!

A challenge!

- Is the Serve Stale feature genuinely useful?
- Do operators of resolvers know for certain that it is helping their servers?
- I have never yet heard of a situation where having Serve Stale enabled 'saved the day' - but would love to hear from anyone who has one!
- One 'useful' that is perhaps passing under the radar might be that resolvers are 'smoothing over' short term authoritative server unavailability or slowdowns. *Would we even know that this is happening?*
- What do we think about *'serve stale content first, refresh afterwards'* (available in some implementations) - and does that count as 'useful'?
- Analysis of Resolver logging and statistics from servers in production environments *might* be the starting point for an assessment of the practical benefits of the Serve Stale feature - a talk for OARC45?

Discussion time …

RFC 8767: “Stale bread is better than no bread.”

Photo by Vincent van Zeijst via Wikimedia Commons

References:

- https://kb.isc.org/docs/serve-stale-implementation-details
- https://kb.isc.org/docs/changes-to-serve-stale-option-stale-answer-client-timeout-in-bind-918-and-newer
- https://unbound.docs.nlnetlabs.nl/en/latest/topics/core/serve-stale.html#serving-stale-data
- https://blog.nlnetlabs.nl/some-country-for-old-men/
- https://websites.pages.nic.cz/knot-resolver.cz/documentation/v5.7.4/modules-serve_stale.html
- https://www.knot-resolver.cz/documentation/latest/config-serve-stale.html
- https://gitlab.nic.cz/knot/knot-resolver/-/blob/master/modules/serve_stale/serve_stale.lua
- https://docs.powerdns.com/recursor/appendices/internals.html#serve-stale
- https://docs.powerdns.com/recursor/settings.html#

Thanks to:

- Wouter Wijngaards and Yorgos Thessalonikefs from NLnet Labs for detailed information about Unbound's Serve Stale implementation
- Vladimir Cunat of CZ.NIC for explaining what happens with the different configuration options in Knot Resolver
- Otto Moerbeek at PowerDNS for disentangling in my mind the different approach taken to Serve Stale by PowerDNS Recursor

Also acknowledging the royalty-free image creators contributing to slide 2:

- clickschool of Pixabay
- Megan Rexazin Conde of Pixabay
- GraphicMama-team of Pixabay
- Yayamamo of Wikimedia Commons

Thank you for listening.

- ISC main website: https://www.isc.org
- Software downloads: https://www.isc.org/download or https://downloads.isc.org
- Presentations: https://www.isc.org/presentations
- Main GitLab: https://gitlab.isc.org