Thinking about Serve Stale

Cathy Almond

2025-02-06

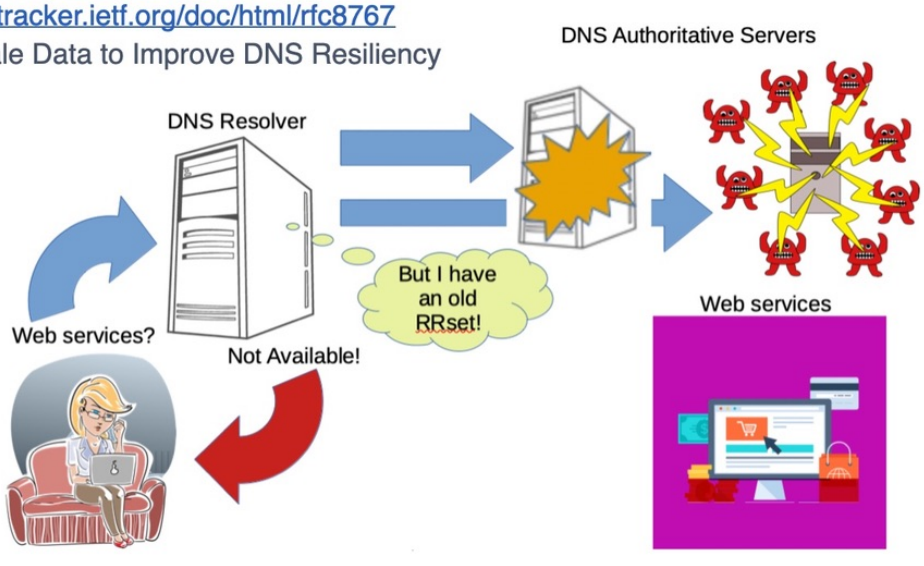
https://www.isc.org

This is a combination of talk and primer covering:
- What is Serve Stale?
- A tutorial on how to configure and use it
- Things you need to be aware of if you're using it
- And a final question to you all

I'm Cathy Almond,
I work for ISC
We make, maintain and distribute Open Source BIND.
But with a bit of help from my friends, I'm going to tell you about Serve Stale in Unbound, Knot Resolver and PowerDNS Recursor too

What is Serve Stale anyway?

https://datatracker.ietf.org/doc/html/rfc8767
Serving Stale Data to Improve DNS Resiliency



Most Internet users rely on DNS without even knowing how important it is
- I chose my T-shirt today specifically for this talk by the way!

If DNS resolution for clients wanting access to services doesn't work, then even if those services are up and running, they're not accessible

DNS is important!

But things can go wrong for authoritative DNS servers
- physical disasters
- DDoS
- People make configuration and operation mistakes

And then then DNS resolvers can't look-up what they need to to provide query responses to their clients.

It started with a big disaster back in 2016
- Successful DDoS against DynDB
- Many popular sites effectively 'down' because Resolvers couldn't reach the authoritative servers
- Management clamoured for a mitigation for this situation

These resolvers - WHAT IF they already had the answers already- DNS resolvers have caches

Resolvers are not supposed to use OLD answers - the DNS protocol includes the Time To Live (think of it as the use-by date on the food you buy).

In fact, think of your resolver as a refrigerator and the records from authoritative servers as the food you bought from the grocery story …

But slightly old answers are very likely (most of the time) going to be correct and better than no answers at all, if something has Gone Wrong.

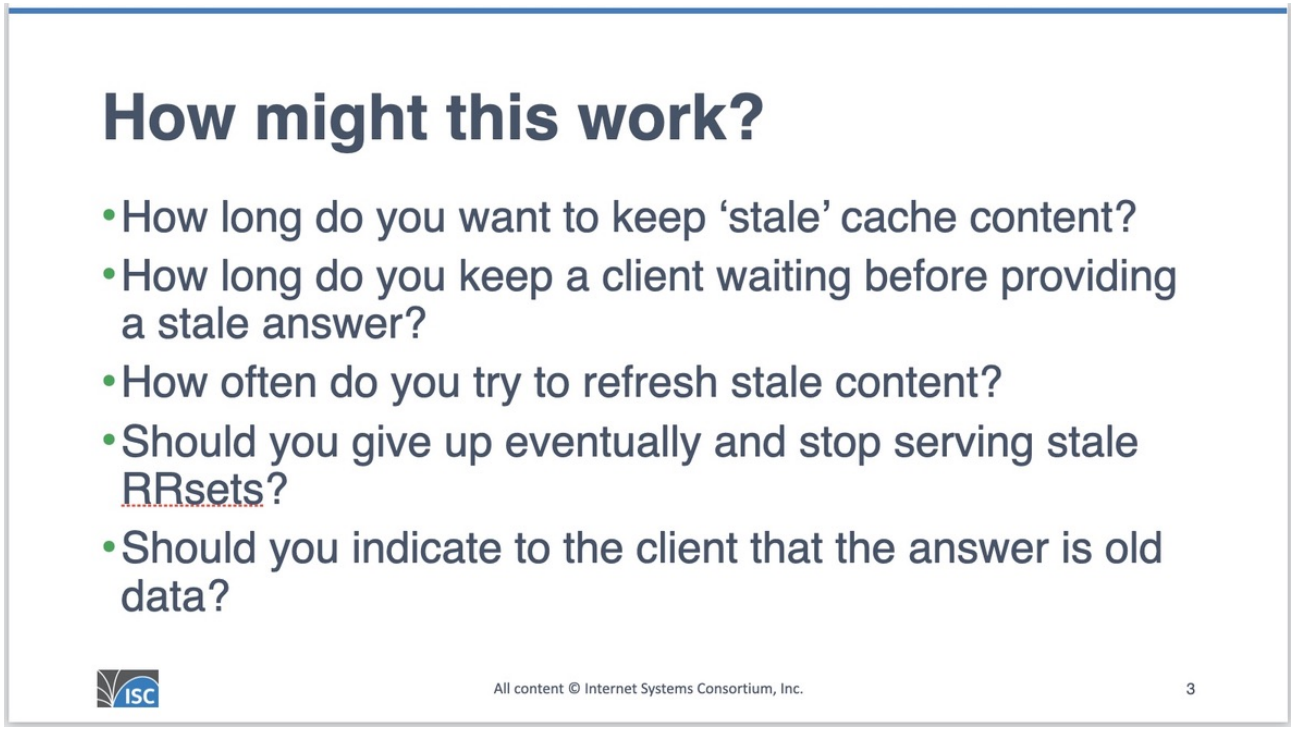
So why not use them as the answer of last resort, and keep the services they point to from becoming inaccessible?

If you're hungry, stale bread is better than no bread at all…

And so, Serve Stale for DNS Resolvers was born…

There's an Internet Standard (RFC) on this - written after several implementations already existed.

It's actually very well put together and easy to read and understand - and I'll cover the general principles in the next couple of slides.



Resolvers have caches

Caches already have retention and eviction strategies for their content
- Ideally keep popular content available all the time if possible
- Include TTL and 'last accessed' information
- Maybe prefetch to refresh RRsets before they expire?

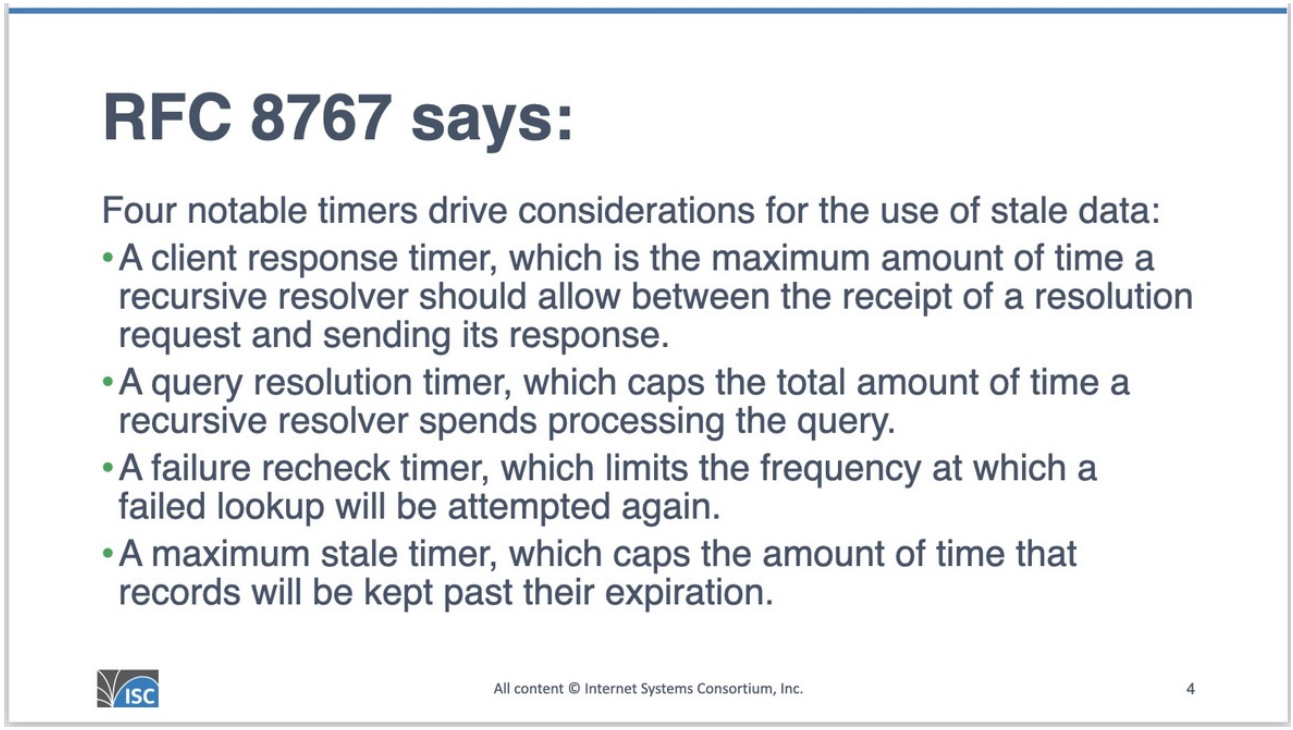
BUT now we're planning to potentially use stale RRsets as a backup
- Should we therefore keep content for longer?
- How much longer?
- What's this going to do for cache memory consumption by the way?

AND how to decide when to use stale RRsets - how long to keep the client waiting?

DO you try to refresh those RRsets on every client query?

And really - when is the old stuff JUST TOO OLD to use?

And … psst! … should you tell the client that you've given them old answers?



RFC 8767 recommends that Resolvers should have these timers …

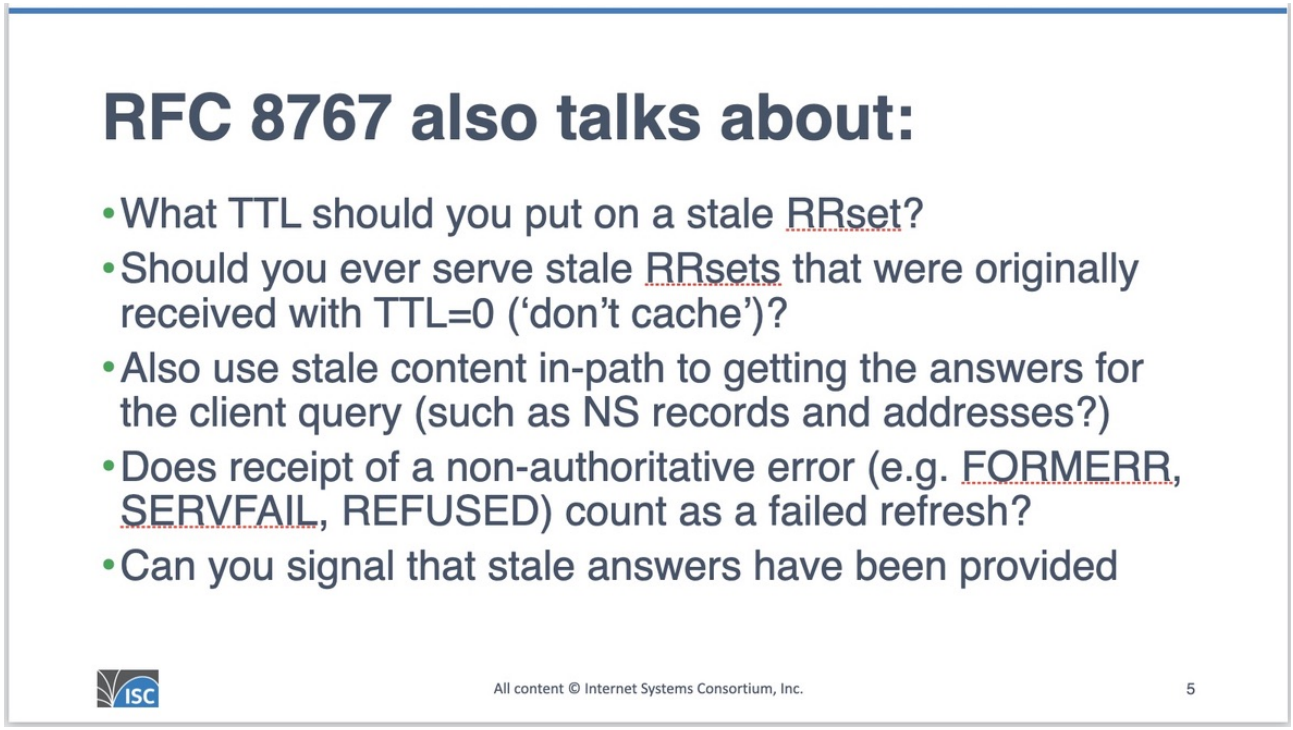
Actually, apart from the last one, they would have had them all anyway in some form or other

- You don't want your clients to have to wait too long (even though they usually retry their queries)
- You don't want your resolver to be doing more work that it needs to

So the practical difference with serve-stale enabled is that the client (if possible) will get old content in a query response instead of SERVFAIL

SERVE STALE instead of SERVFAIL ??

Tempered of course by not serving content so old that it's going to give the clients virtual food poisoning!

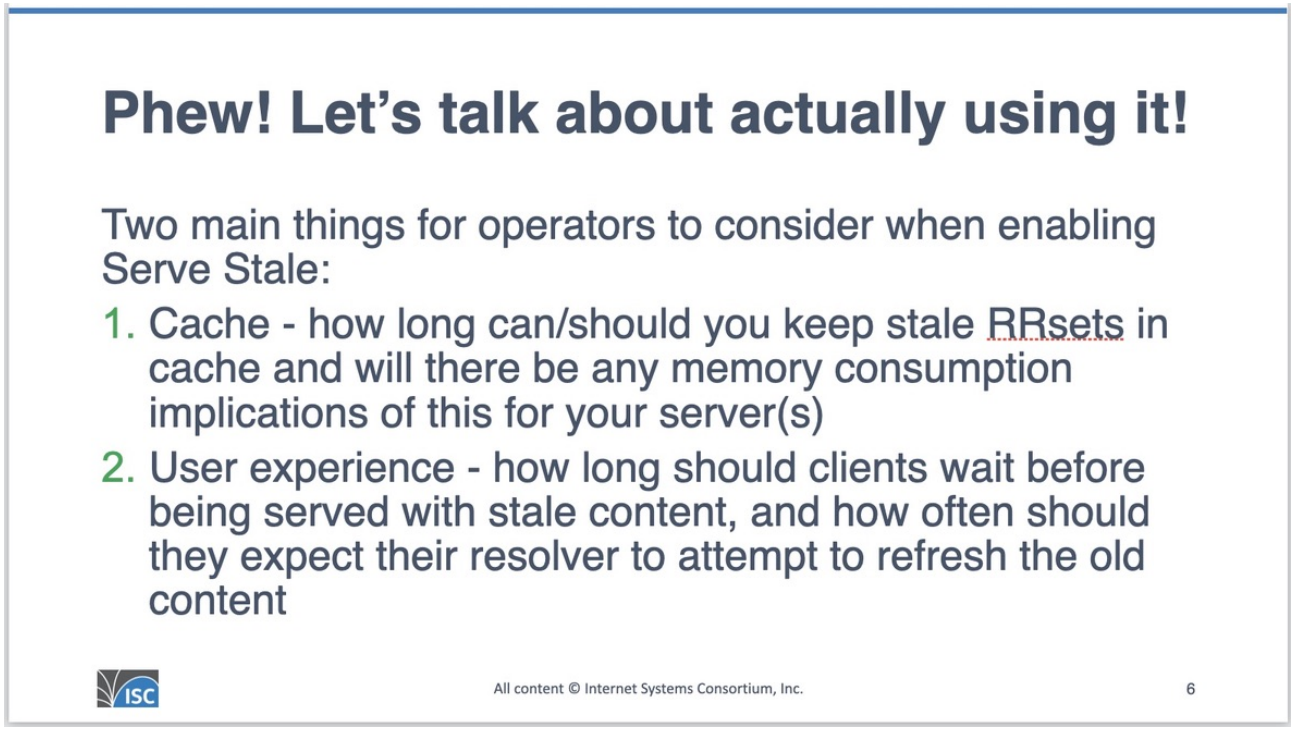


Oooh - some new things to think about we didn't touch upon before
- You're giving your client old expired stuff - already past its TTL. How long do you want the client to use it for before coming back to ask you again?

- Some authoritative servers send 'one time use only' RRsets - what should you do with those?

(ASIDE: Authoritative and Resolver operators alike pay attention to the received TTL=0 scenario later - whether you like it or not, there ARE resolver handling differences!)

- What is a failure anyway (not all auth problems are down to physical disasters and deliberate attack - sometimes people just mess things up…)

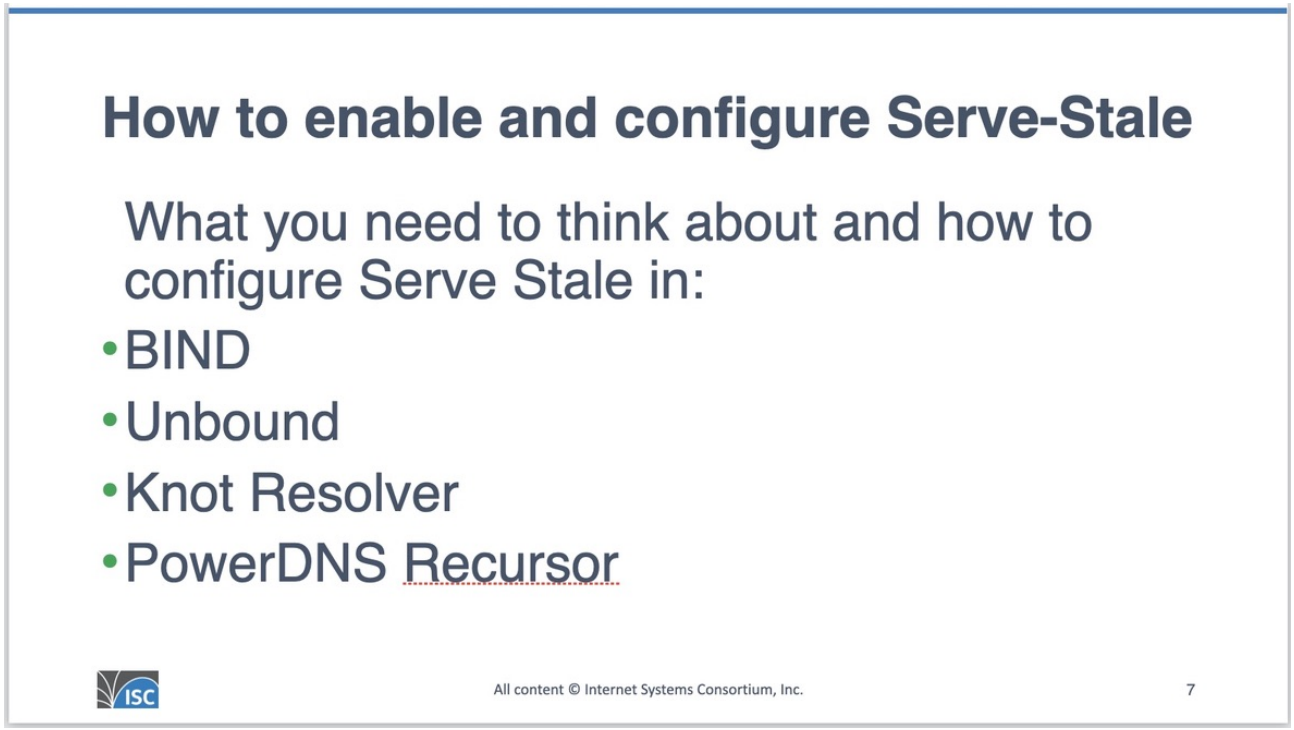


This slide speaks for itself:

1. If you enable Serve Stale, are you going to run into memory consumption problems? Need to plan ahead?

PARTICULARLY on the memory consumption thing - bear in mind that it's not just the popular content that is going to be retained for longer, but the one-time-use content too, and this could cause cache management issues in some implementations.

2. How do you want things to be for your clients/users?



Disclaimer: I know a lot about BIND - obviously!

Hopefully I won't make too many mistakes when explaining how the other three implementations work!

(But they did all approve the slides - so use those afterwards as the source of truth rather than what I say!)

====

NOW, upfront I am going to say 'there is a lot of data in the slides that follow'

I'm NOT going to explain all the options and things to think about down to the tiniest detail

The approach I've taken, is that the slideset is your PRACTICAL TAKEAWAY, to use later.

So instead of bullet points on the slides that the speaker elaborates upon…
I'm elaborating on the slides, but giving you bullet points in my talk over!

So I recommend that you listen rather than read (unless you can multi-task effectively)

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
```

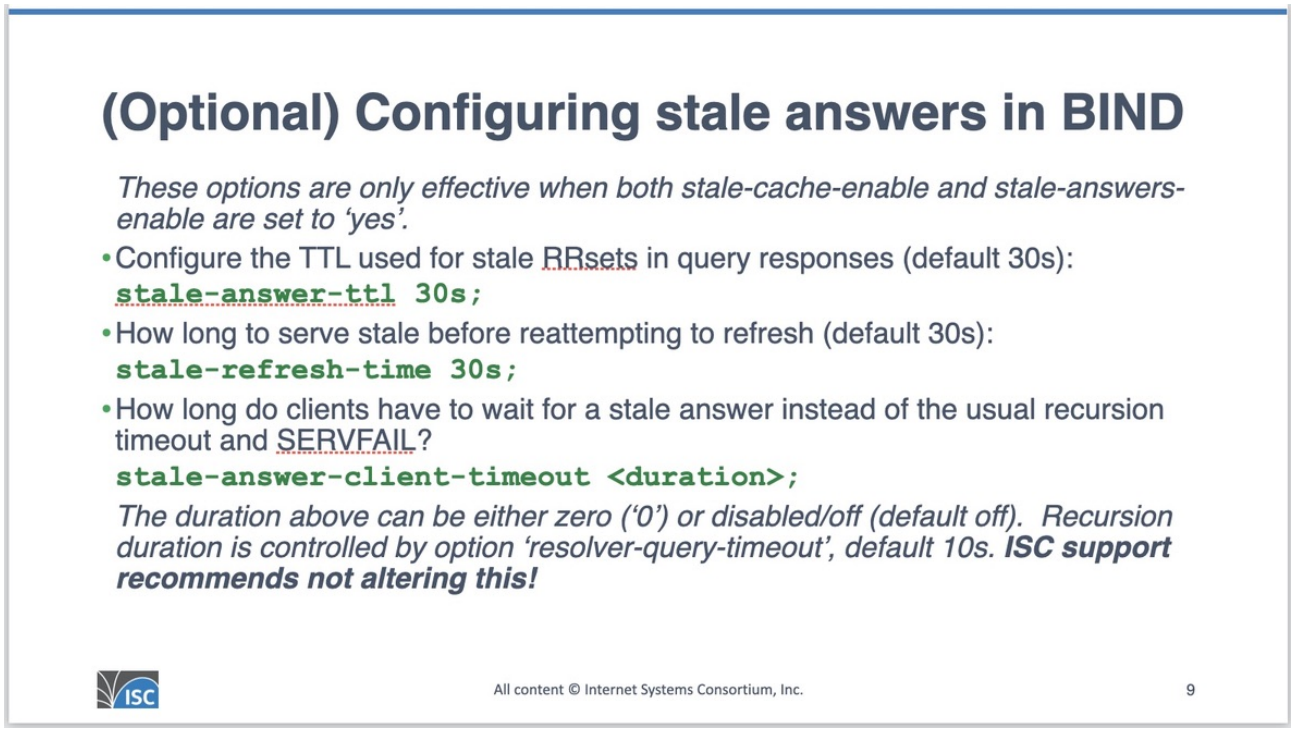
Note that you can't enable stale answers without first starting BIND with stale cache enabled - no stale cache means no stale answers available.

BIND has two separate things to enable
- Retaining of stale content (in case it's needed)
- Activating the processes to use stale content if it can't be refreshed

Neither are enabled by default

It's possible to enable stale cache and then turn on stale answers dynamically afterwards

How would you know you needed to do this though - and probably by the time you realised you needed to, there could be a lot of unhappiness abounding?



You've got the knobs you need (if you want them) for tweaking the standard client experience - but the defaults are sane and most users of Serve Stale won't need to adjust them.

And then there's an interesting knob: stale-answer-client-timeout

Oh we have had some fun with this one …

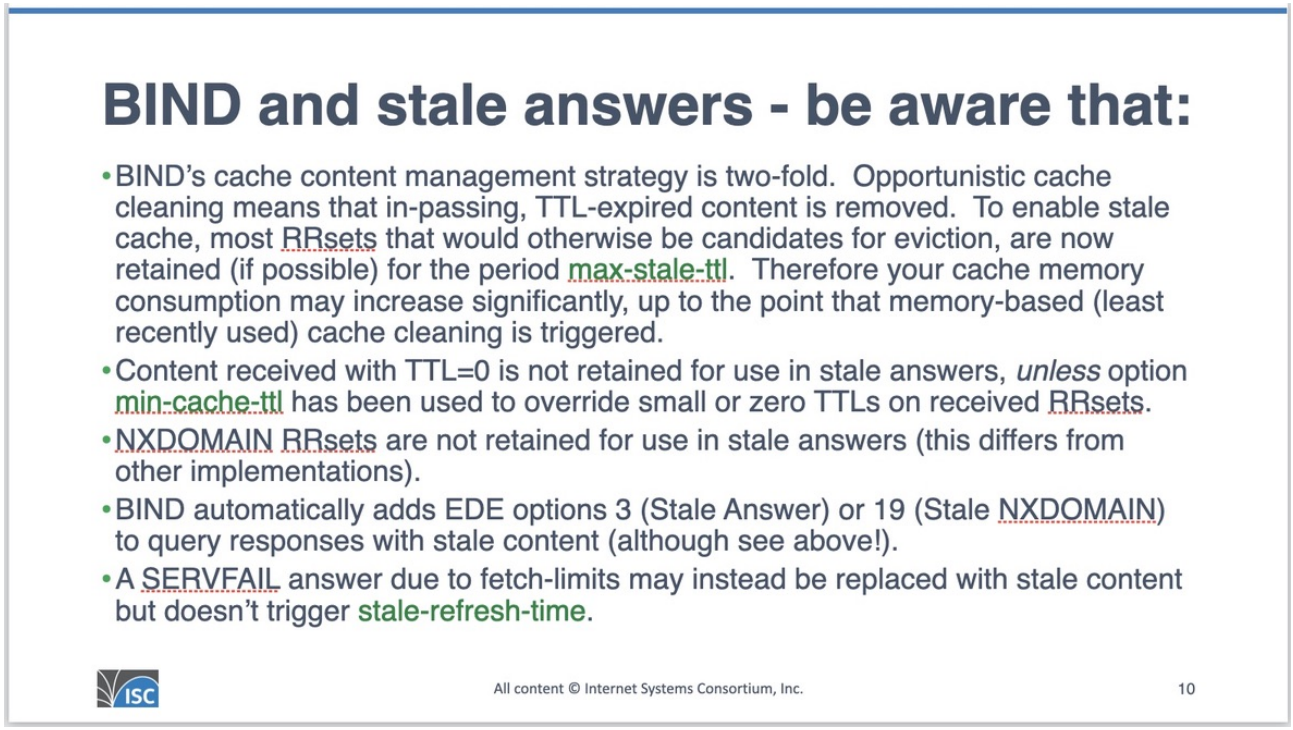
The original idea was that you could decide how long the client had to wait before getting a stale answer (but still leaving the cache refresh to complete as normal - successfully or otherwise).

Where we're at now is that you either just have to let the client wait as usual until we replace SERVFAIL with Serve Stale

OR

If there's a stale cache hit, then serve that *immediately* and leave the refresh to do its thing afterwards.

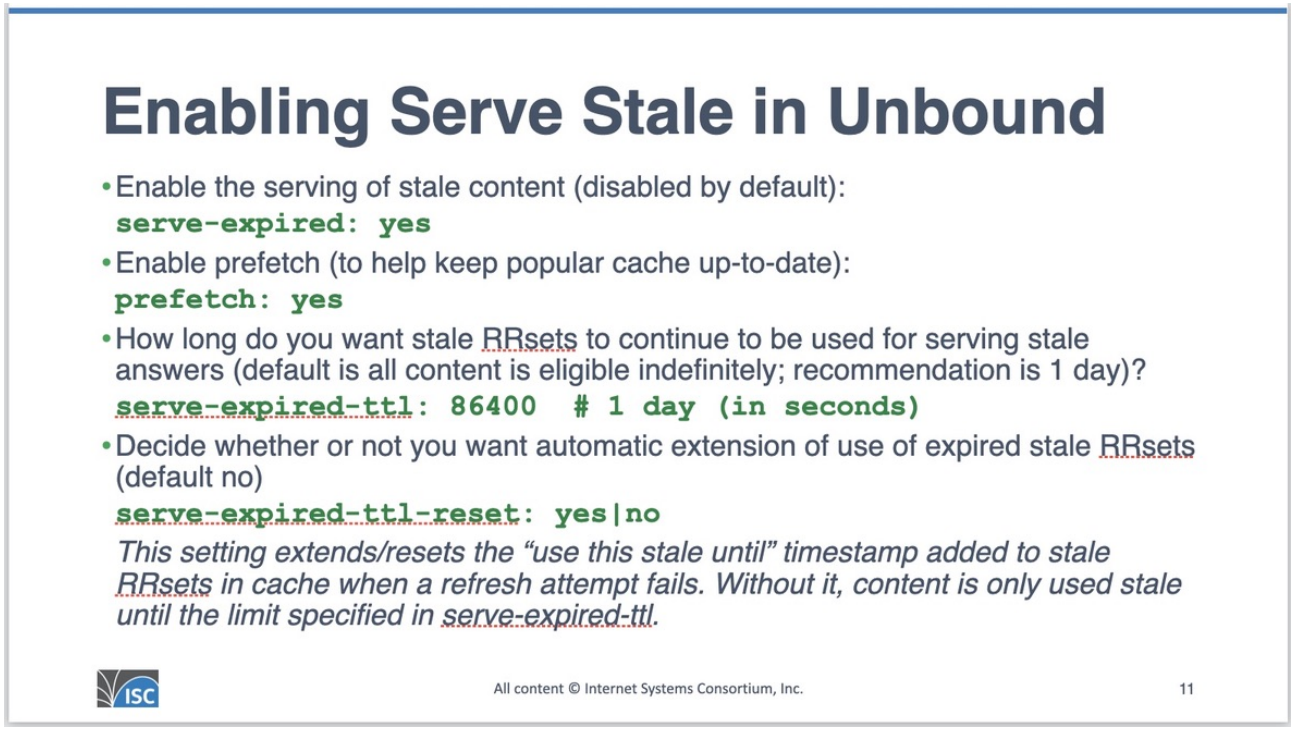
Would you want to do that? (More on that at the end (BIND is not the only implementation that allows this) …



The biggest takeaway point is the cache consumption one. BIND's routine housekeeping of expired content is less processing-intensive than the memory-based mechanism that is triggered when max-cache-size is reached.

Monitor and provision more memory/cache and/or reduce how long you retain stale content for if you need to and you will be happier…

And yes, we do do EDE. Both EDEs - except that we decided later not to apply Serve Stale to NXDOMAINs, so you'll never see EDE 19 in current BIND with Serve Stale enabled.

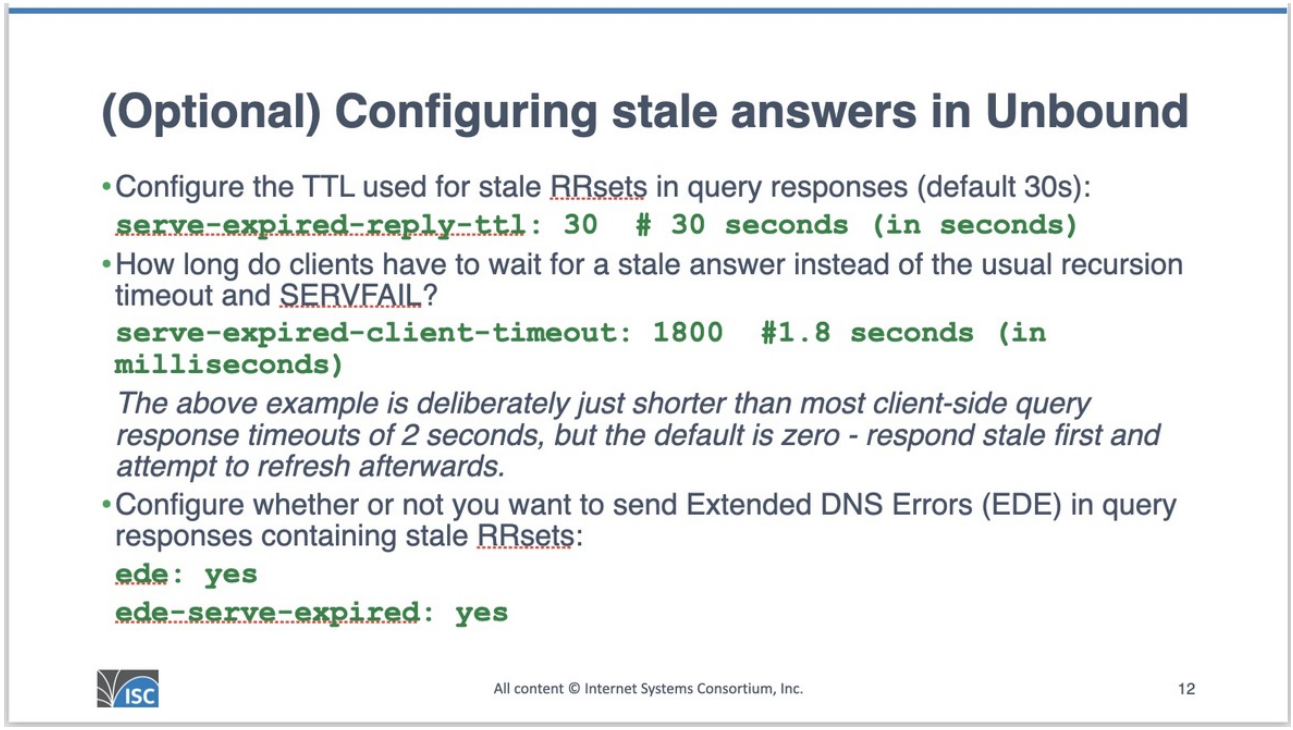


The original default behaviour with “serve-expired: yes” AND NOTHING ELSE CONFIGURED is that all content in cache is eligible to be served stale if needed.

This works because Unbound has no expired-TTL cleaning mechanism, so as long as there is still room in cache, the content remains available.

Alternatively, you can say ‘only use these old RRsets until a given limit and never again thereafter

And on top of that, you can also opt to reset that ‘only for so long’ timer - but that requires a full attempt to refresh before the old content gets sent to the client.

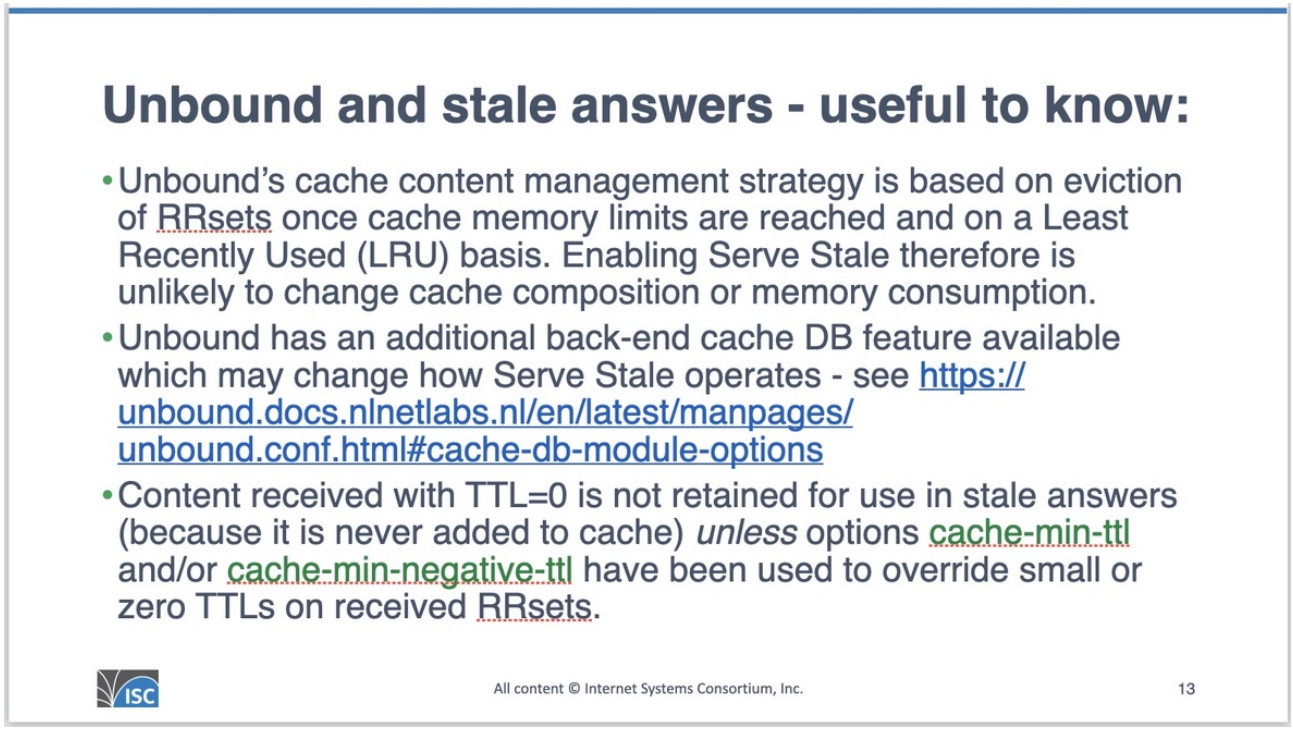


Again, and similar to BIND, in Unbound you've got the knobs you need (if you want them) for tweaking the standard client experience - but the defaults are sane and most users of Serve Stale won't need to adjust them.

… except possibly serve-expired-client-timeout

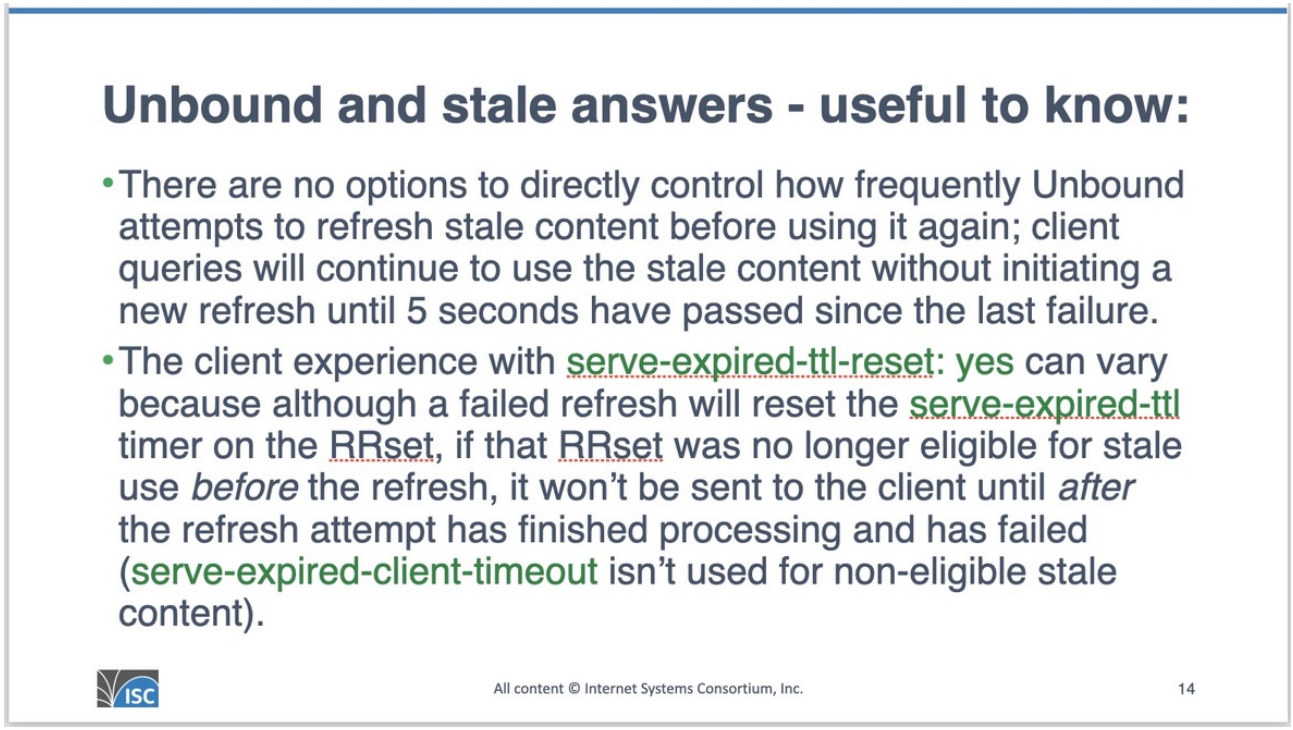
This works similarly to the BIND option stale-answer-client-timeout - it's about how long to wait for refresh recursion to complete for the first time, before opting to use an existing stale RRset for the reply to the client. But in Unbound you can make more use of it, as well as opting to serve stale first and then refresh afterwards.

Also note that you will need to enable both EDE and the serving of EDE when providing stale answers to clients.



The takeaway from this slide is that you shouldn't need to worry overly about cache consumption in Unbound if you enable stale answers - it's not going to change cache consumption. Cache content that was least recently used will still be evicted once cache memory limits have been reached.

Also note that back-end cache DB feature - I did not dig into this and I leave its functionality for the reader to research!



What this means is that after the first 'serve-expired-client-timeout' and then use of stale RRsets, those same stale RRsets will continue to be used immediately for subsequent client queries until 5 seconds have passed, at which point, the next client query will cause another refresh attempt to be initiated.

And 'serve-expired-ttl-reset' is complicated. What it essentially means is that IF you decide to set the limit on 'this stale content is so stale that it would now be poisonous if used' (serve-expired-ttl) instead of leaving all stale content eligible until it's evicted from cache, that you DO have the possibility of using it in desperation. HOWEVER, a thorough attempt to refresh it will take place first (serve-expired-client-timeout does NOT apply here). But at least there is the possibility of 'answers of last resort' - should you want them?

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.

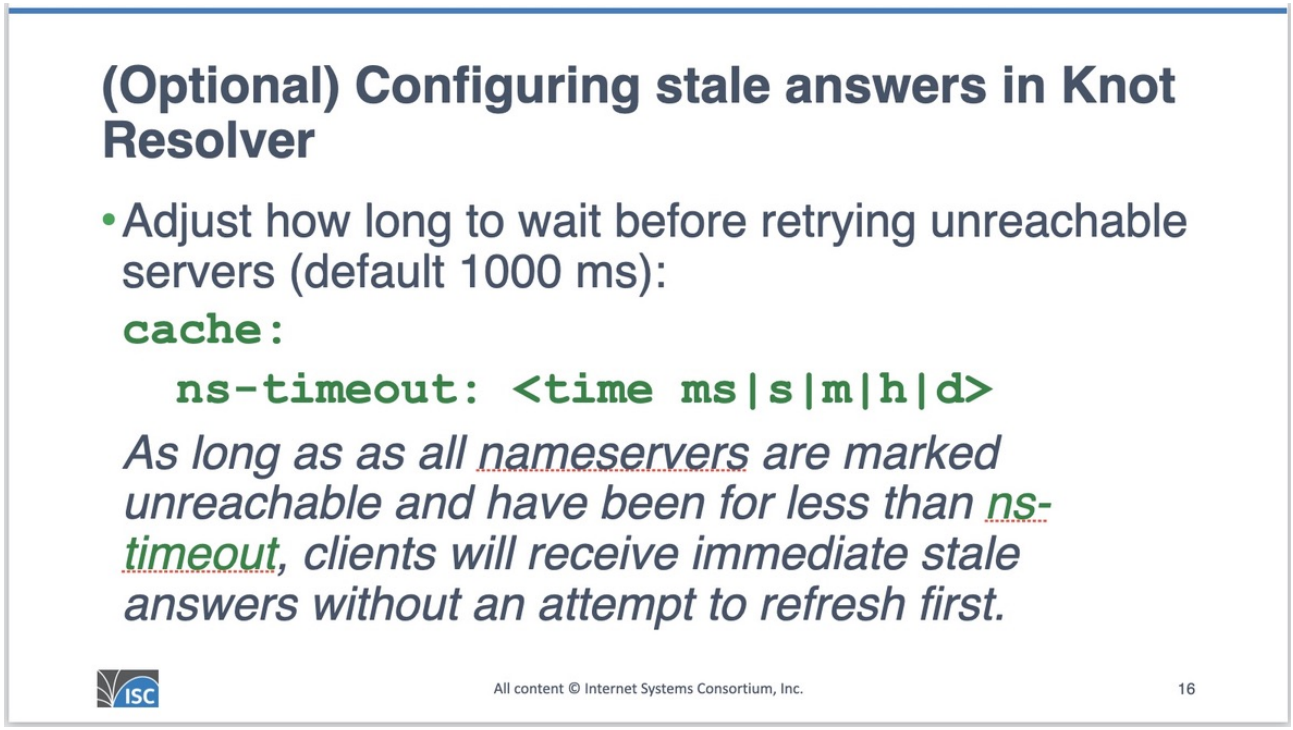
Knot Resolver has a simplistic approach to Serve Stale, with few options. Basically, just enable it and it's 'on'!

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.

This option is not specifically for Serve Stale - it is about setting timeouts during recursion anyway. If "serve-stale: true" had not been set, then the client would receive SERVFAIL instead of stale content.

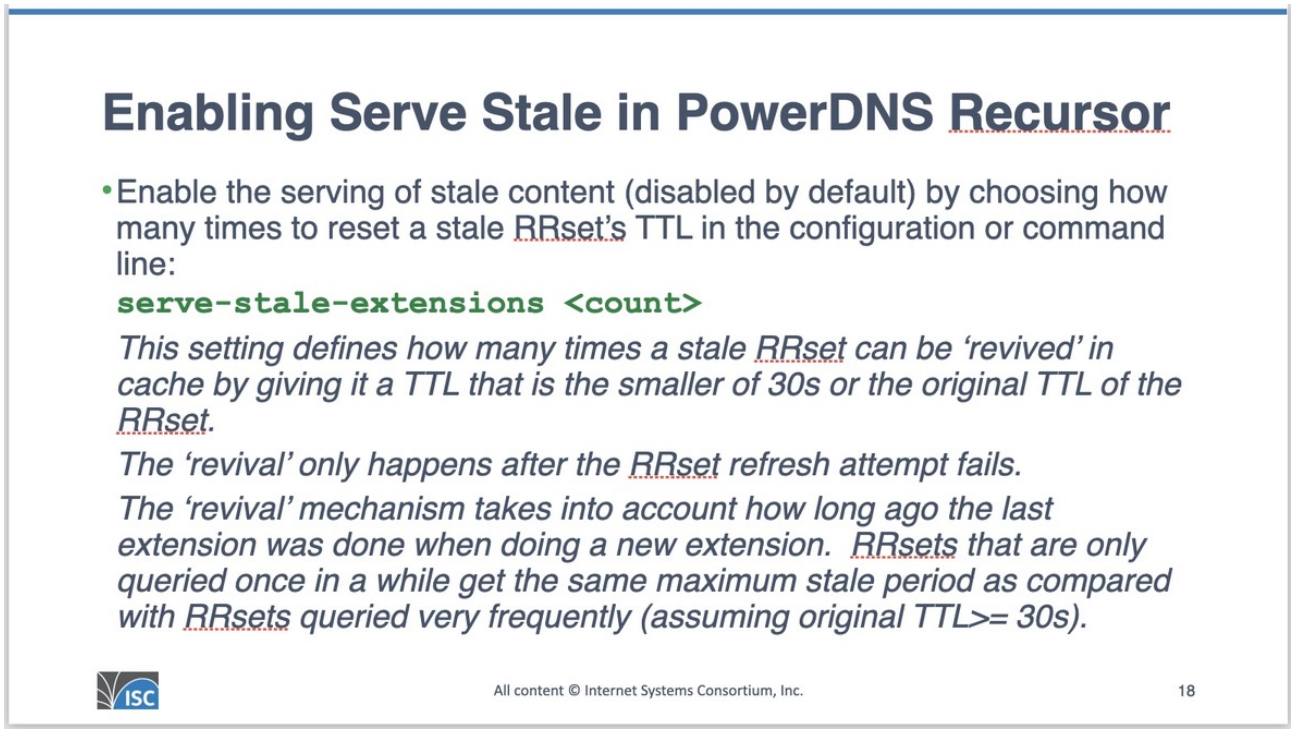
You probably don't want to adjust this.



Knot Resolver's cache management strategy is similar to Unbound's - so enabling Serve Stale isn't going to change much about cache consumption or cache composition.

As noted before, there's not much that you can tweak in the Knot Resolver configuration to change Serve Stale behaviour, but you could modify some timers in the lua module instead.

But please note one BIG DIFFERENCE in relation to TTL=0 content - Knot resolver already defaults to overriding the cache TTL on RRsets received with TTL < 5s. These RRsets are added to cache regardless - and they will still be added to cache if this override is disabled using cache/ttl-min. There is nothing in Knot Resolver's Serve Stale handing to exempt 'one time use' TTL=0 RRsets from use when stale, if they cannot be refreshed.



PowerDNS Recursor has taken a slightly different approach to implementing Serve Stale.

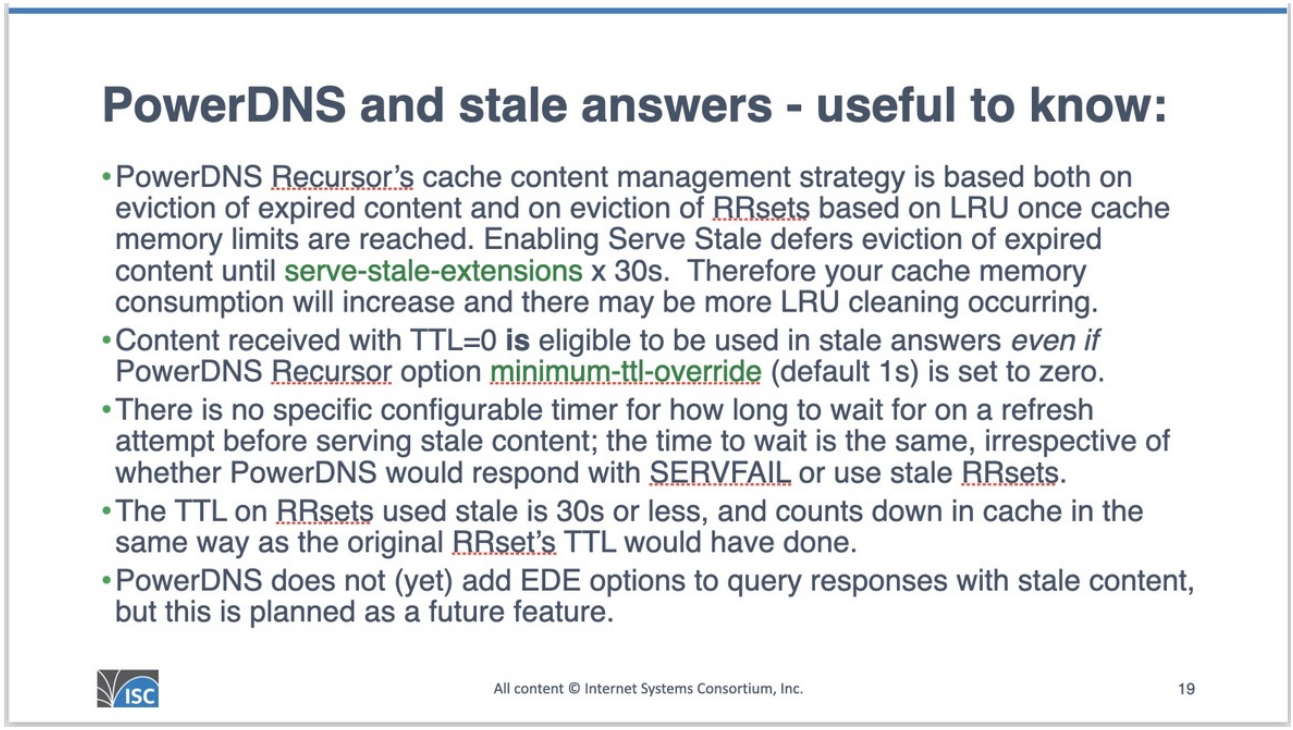
Note that all implementations have to consider:
- What TTL to put on stale answers served to clients
- How long to continue to use stale RRsets from cache before attempting to refresh them again

PowerDNS Recursor handles this when it 'revives' stale content by giving a new TTL of 30s in cache, which then 'counts down' in the same way as the originally received TTL from the authoritative servers did - after which it can't be served again without attempting to refresh it.

There is also a 'count' that you can configured - which is how many times this 'add another 30s TTL' can be allowed to happen.

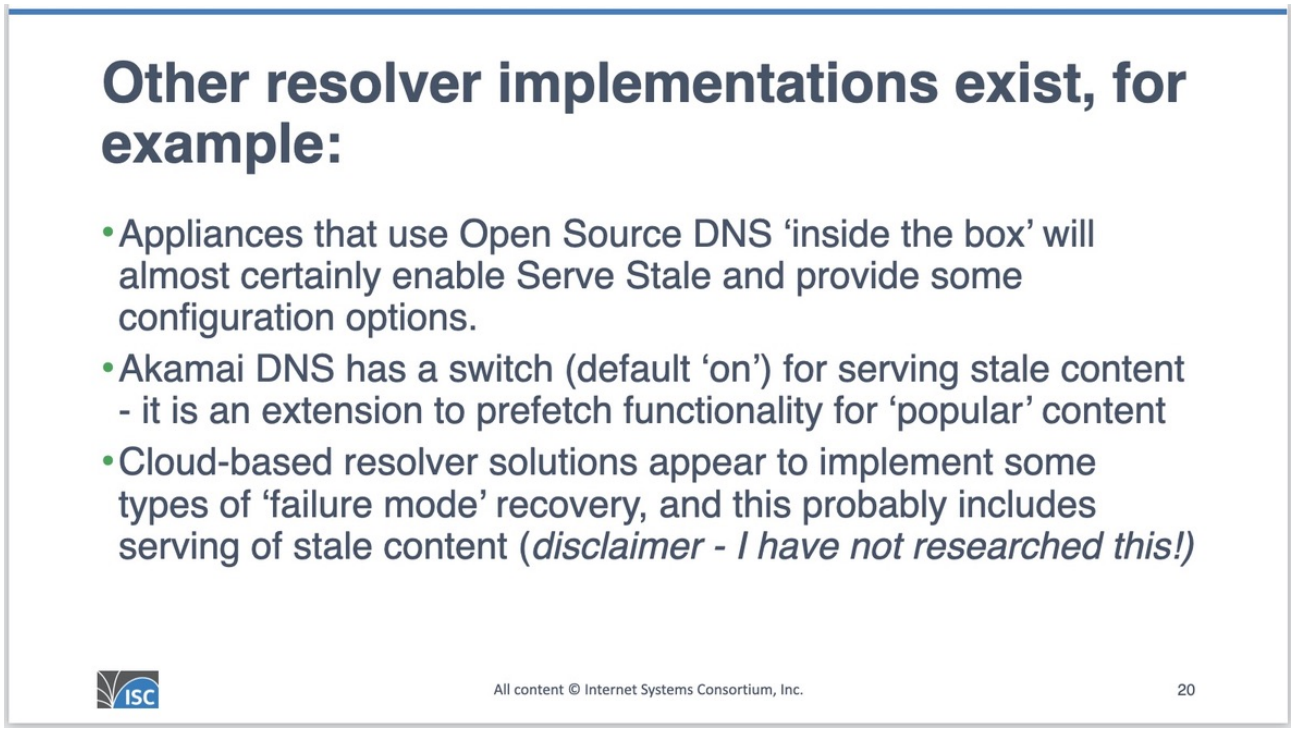
Ah but (you might say), "what if the RRset is only queried once every 5 minutes, does that mean it can survive in cache to be reused much longer than an RRset that is queried every second?" I asked. Apparently not - the <count> and calculation is done on the basis that the refresh attempt was done after each 30s, so old content that is infrequently queried doesn't get reused for longer than popular content.

Also note that the 30s TTL used when 'reviving' stale content might not be 30s if the original authoritative TTL was shorter than 30s. (But the calculations on how long this content is available are still done on <count> x 30s).

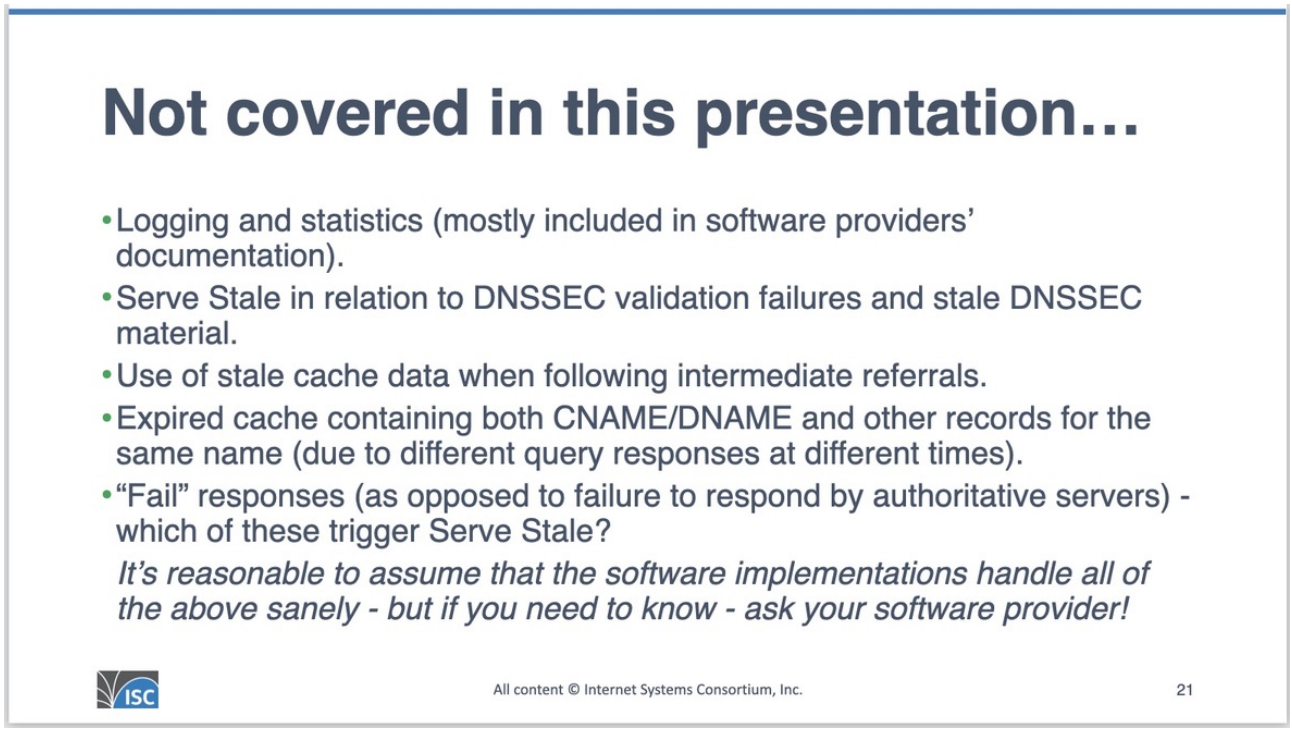


PowerDNS Recursor is akin to BIND in that there are two cache eviction strategies, one based on TTL (content expiry) and the other on reaching cache memory limits, so you will need to think about cache content and cache memory consumption if you enable Serve Stale.

And coming soon - EDE options.



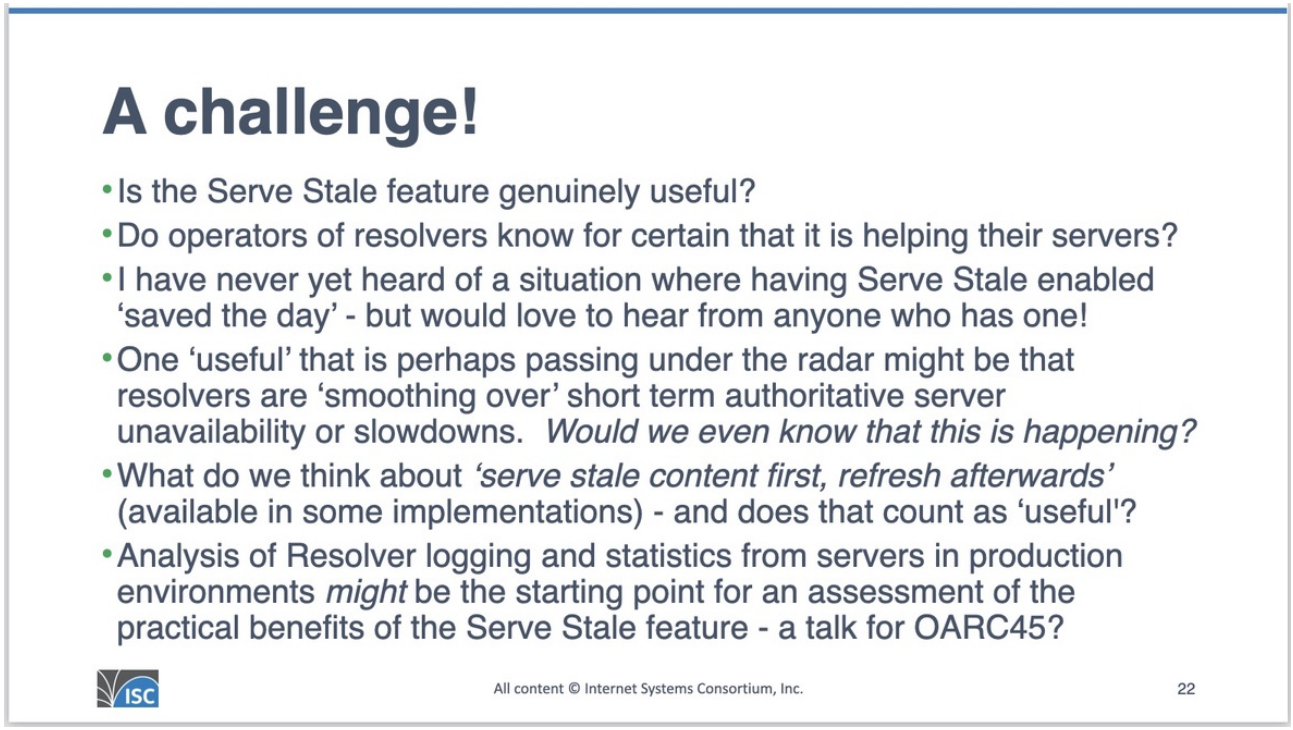
If you're using something other than the DNS resolver implementations I covered then please ask your provider for advice on whether or not they provided this feature and if they do, how it works and what configurable options are available.



Please refer to the documentation for logging and statistics - and also take a look at both of them if you enable Serve Stale in your environment . Is it being triggered? Is it helping you?

There are a lot of nuances and edge scenarios that I didn't dig into (well, I dug into some of them while preparing this talk, mainly to make sure I understood the options correctly, but also because I was fascinated by how the underlying resolver architecture drove the Serve Stale implementation choices made by the developers).

If you have any specific "what if…?" questions, then I would encourage you to ask your software provider directly.



I asked the questions above at the end of my talk. Please watch the recording if you want to hear the discussion, but essentially:

- Several operators confirmed my hypothesis that it does get triggered, but on a small scale ("smoothing over the cracks…")
- No-one has yet seen it 'save the day' on a big DNS authoritative services outage
- One hypothesis on "smoothing over the cracks" is that the cracks are BGP and other routing 'flaps'.

So the conclusion we came to, is that this feature IS genuinely useful, but not a knight in shining armour riding in to save the day - more it's a series of small bandages being applied just to hold things together from time to time on an ad-hoc basis to improve the 'client experience'.

(We didn't touch on 'serve stale content first, refresh afterwards' – which sounds tempting if you are being measured on time to respond to client queries – but do you really want to serve a stale answer when there may be a different answer available from the authoritative servers once that refresh has completed?).

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