Otto Moerbeek, Open-XChange/PowerDNS

OARC 40 Feb 16-17 2023

Aggressive cache (RFC8198) effectiveness, the NSEC3 case

Stay Open.

Agenda

Aggressive caching

A better form of negative caching, employed by recursive resolvers

- “Normal” negative caching: only for **names/types** queried before (some optimisations possible, e.g. RFC8020)
- Main observation: an NSEC or NSEC3 record denies a **subset** of names
- If a query comes in first check to see if you already have an NSEC(3) record that denies it.
- If so: no query to auth needed, response can be *synthesised*
- You actually need to do a bit more work, details in RFC8198
- Implemented in major resolvers
    - unbound and bind: NSEC only
    - Knot Resolver and PowerDNS Recursor: also NSEC3
- NSEC3 *with* opt-out does *not* allow for aggressive caching

Nov 2022: a large zone switched

- **.nl** has > 6M names, > 60% are DNSSEC signed
- Switched from NSEC3 with opt-out to NSEC3 without opt-out (including changes to salt and iterations count)
- Since opt-out was switched off, there is now an opportunity for aggressive caching
- Expected: drop of NxDomain queries as many ISP resolvers in **.nl** validate and are running software capable of NSEC3 aggressive caching

Observations and basic tests

- After some time, no drop in queries resulting in NxDomain seen
- This prompted an investigation
- Only PowerDNS Recursor and Knot actually do the NSEC3 aggressive caching, but we know a few large ISPs in the Netherlands run PowerDNS Recursor
- First check CI tests: they do cover NSEC3 aggressive cases
- Then did a few simple test wit a local PowerDNS Recursor
- NSEC zone: quick success in hitting the aggressive cache, even for a pretty large zone: **.se**
- NSEC3 zone: no such luck, *until I tried a small zone*
- So likely not a bug but something else: a more systematic test needed

Test setup

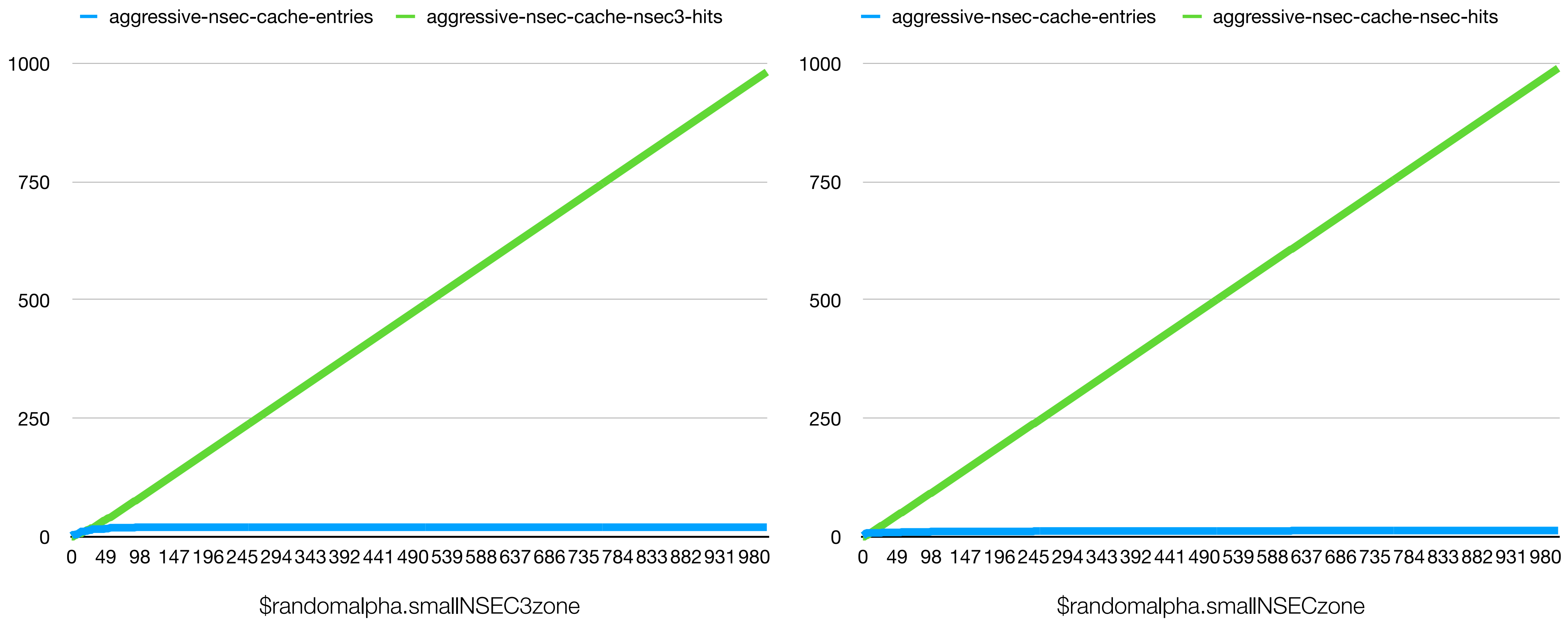
PowerDNS Recursor development version ~ 4.8.0

- Start with clean cache
- Data files with X random strings of various patterns
- In a loop
    1. Query name read from file
    2. Record aggressive cache statistics

- Aggressive cache gets filled during the test (PowerDNS Recursor uses a separate aggressive cache data structure)
- We graph aggressive cache entries and hits
- `aggressive-nsec-cache-entries` is sum of NSEC and NSEC3 entries
- Four test zones, somewhat similar in size:
    - Small NSEC zone
    - Small NSEC3 zone
    - **.cz** (NSEC3 with no opt-out)
    - **.nu** (NSEC)

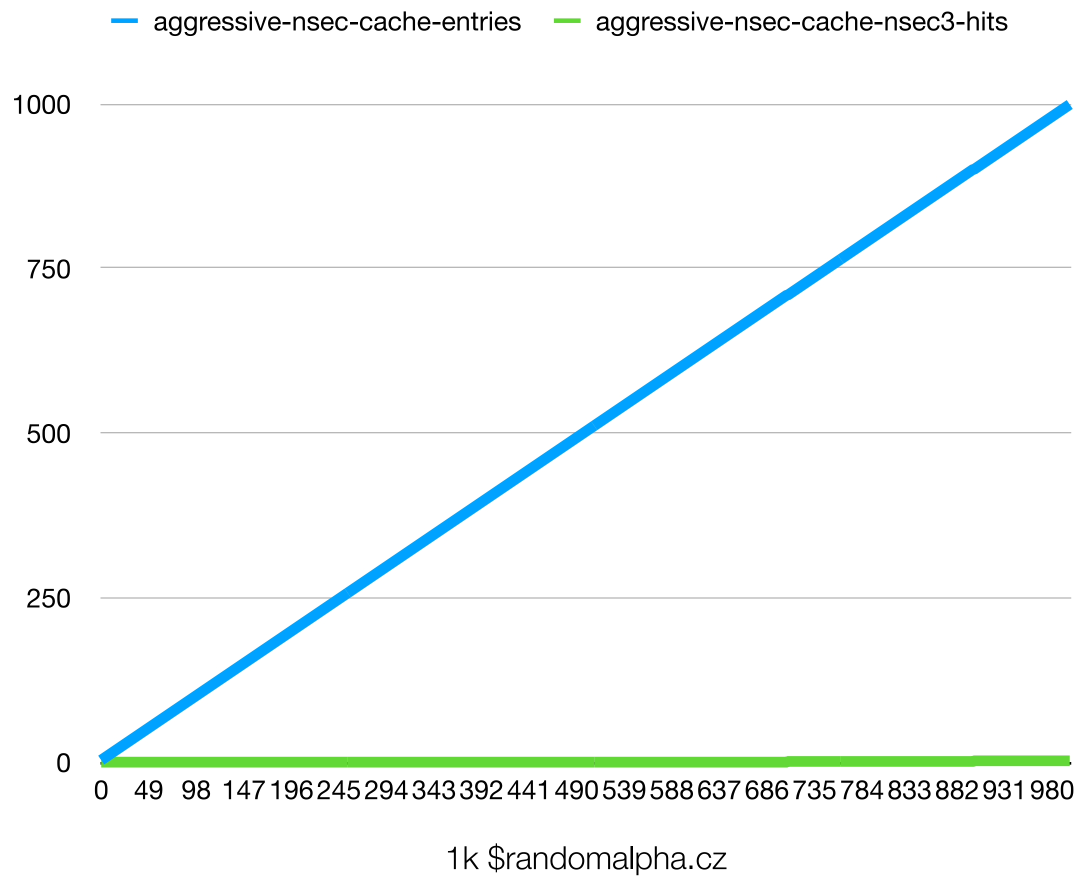
Two small zones, 1000 random alphanumeric names

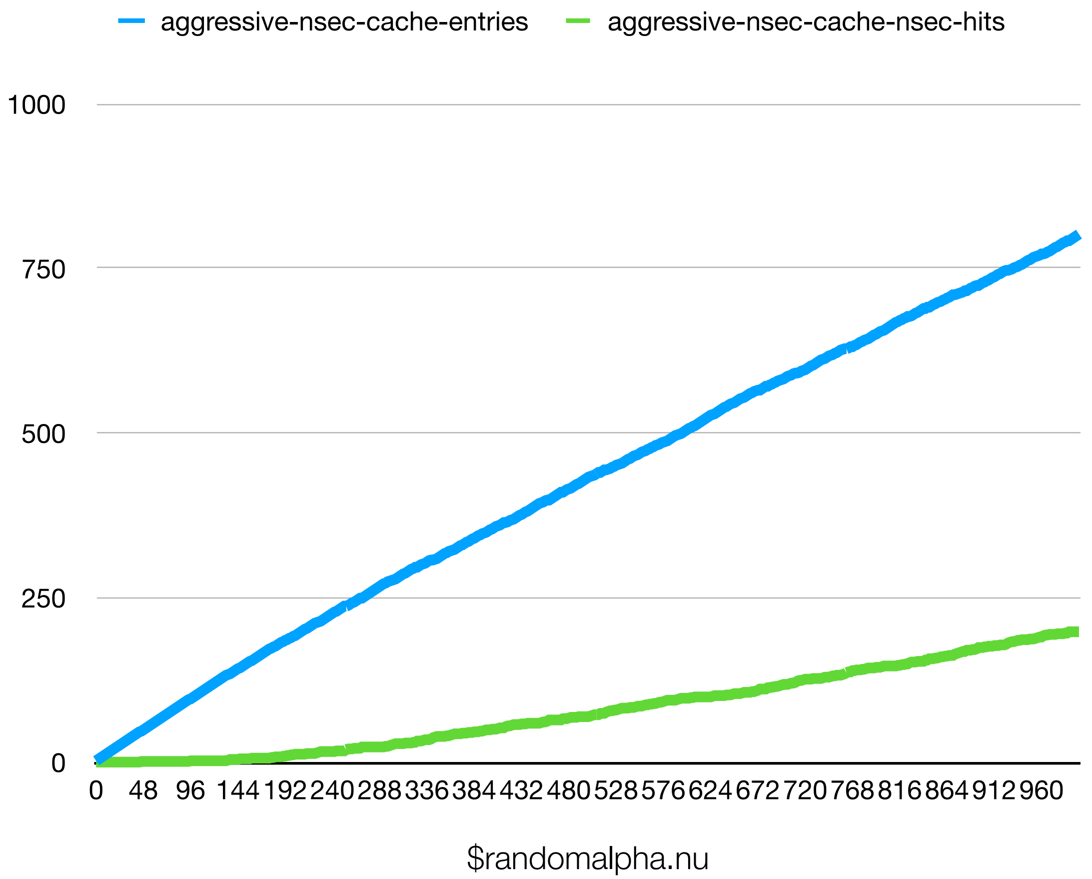
left chart NSEC3 and right chart .NSEC



1000 random alphanumeric names

.cz (NSEC3) and .nu (NSEC)

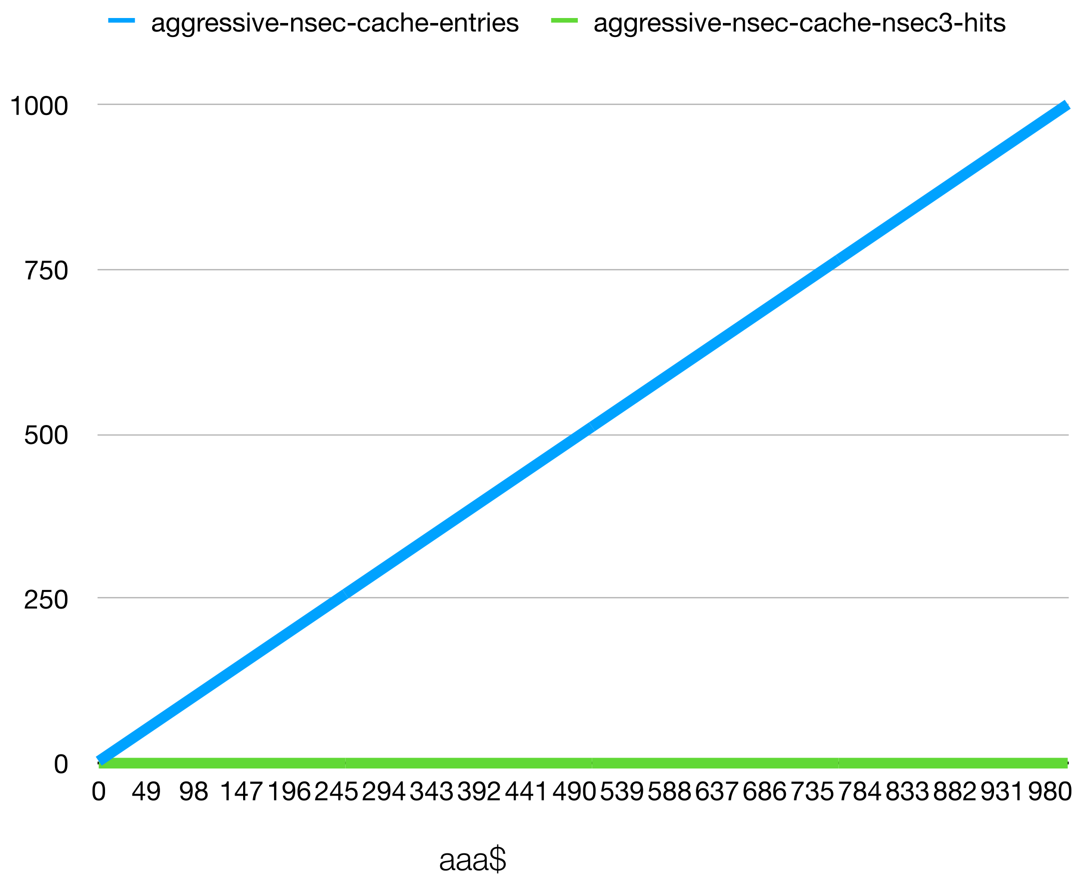


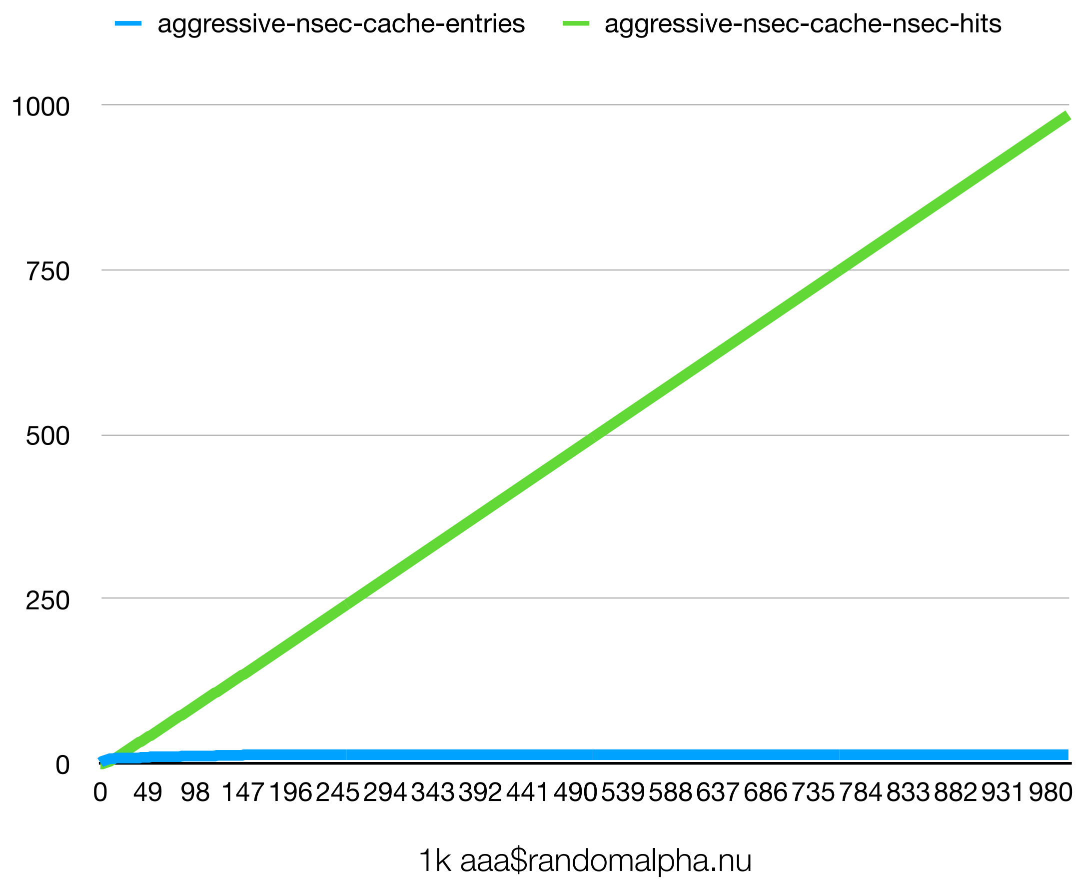


Stay Open.

1000 random alphanumeric names but with ‘aaa’ prefix

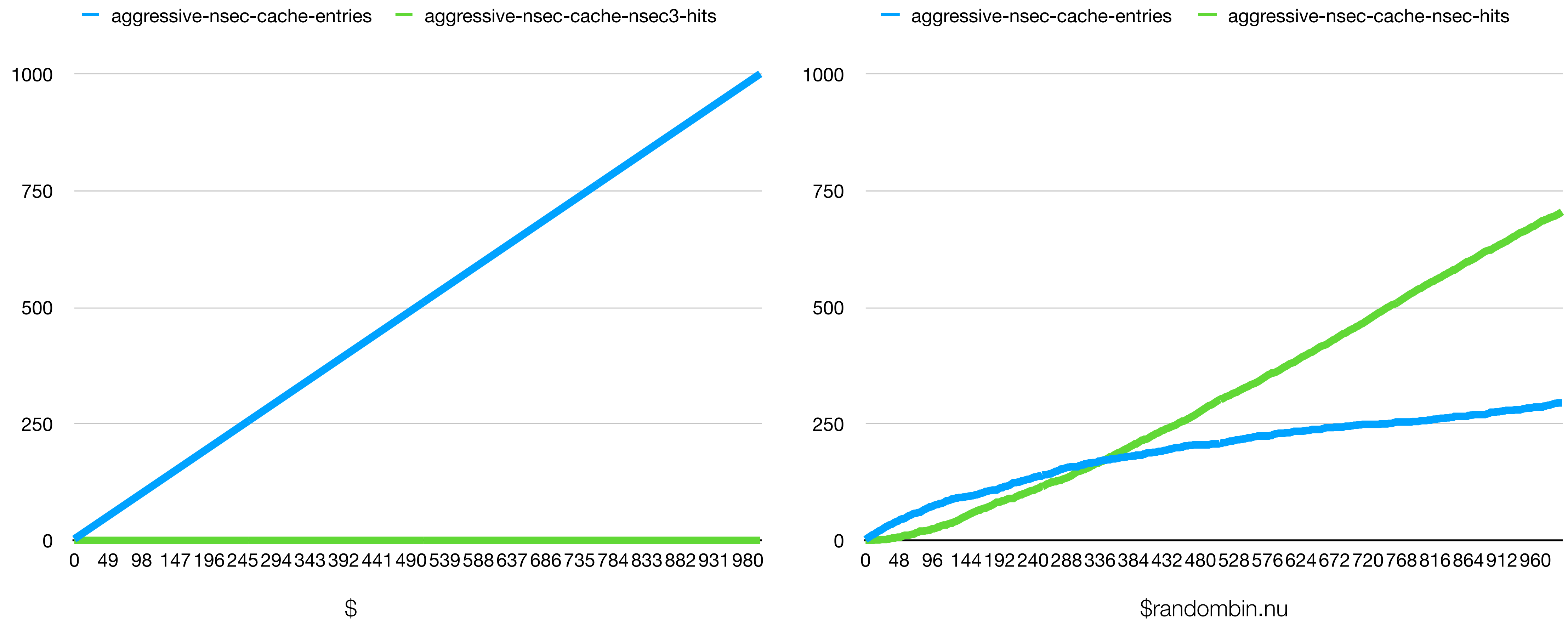
.cz (NSEC3) and .nu (NSEC)





1000 random binary names (full label namespace)

.cz (NSEC3) and .nu (NSEC)

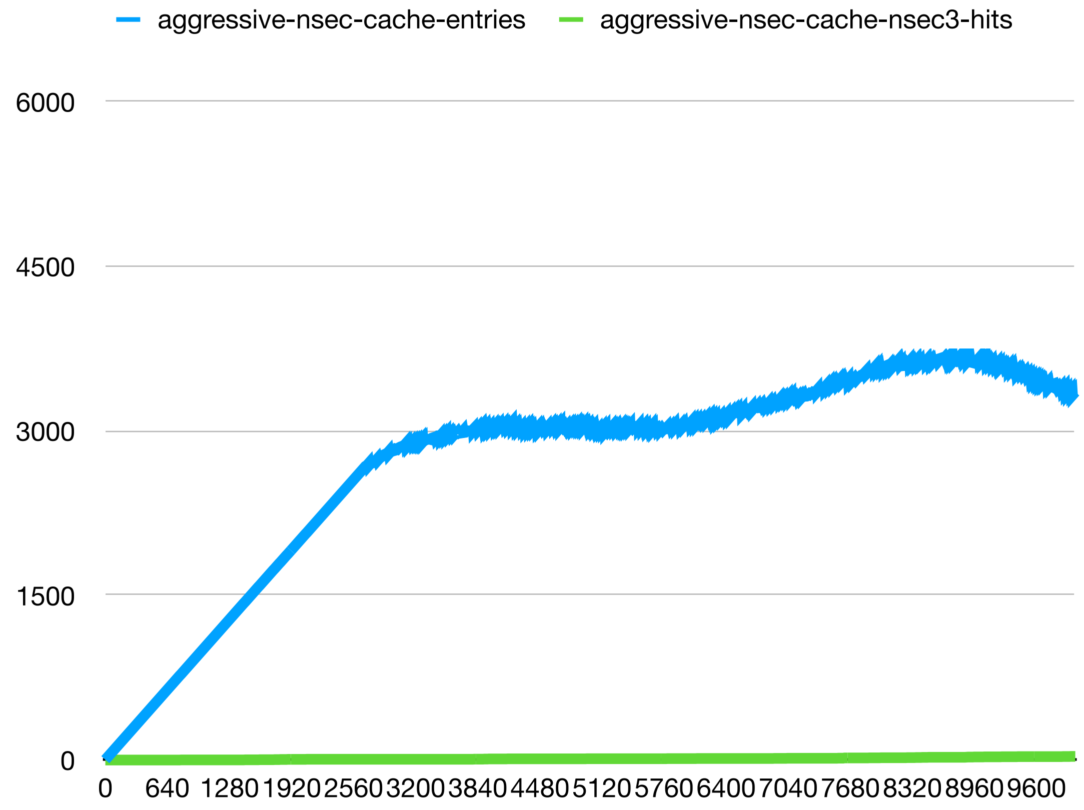


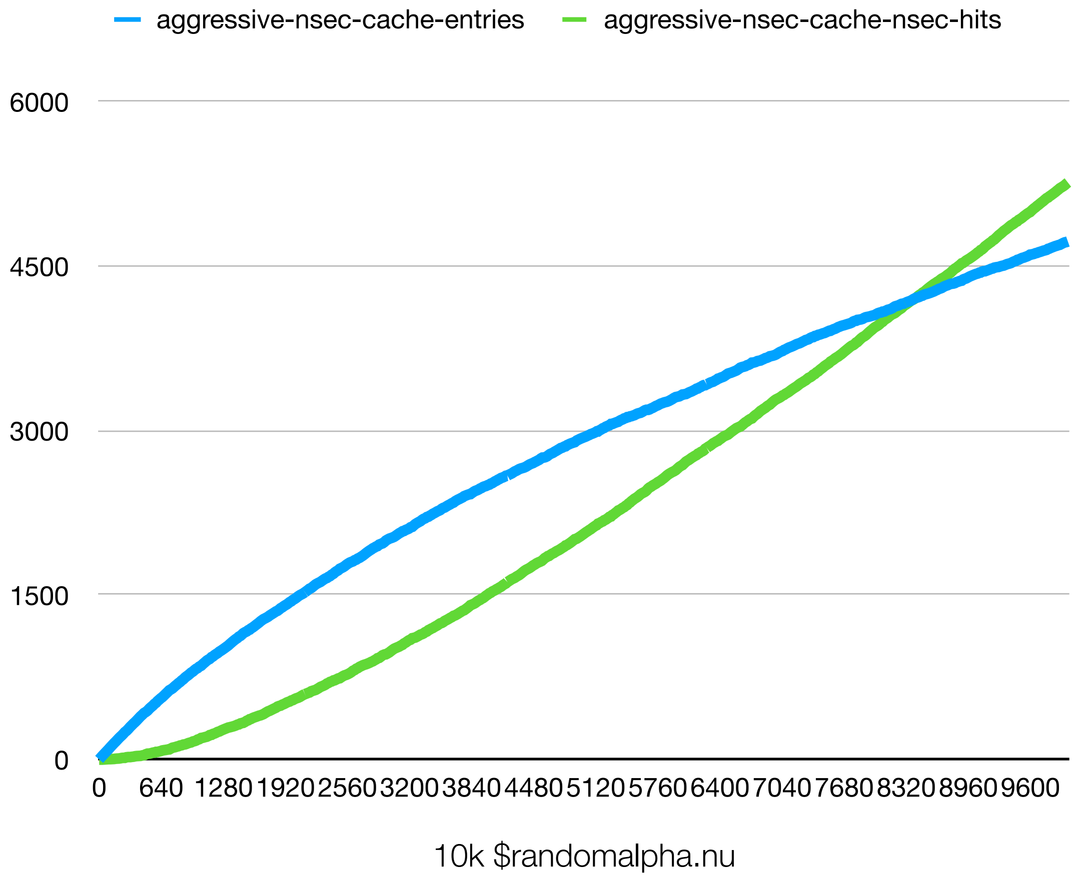
Stay Open. OX

10000 random alphanumeric names

.cz (NSEC3) and .nu (NSEC)

Left hand chart shows cache cleaning in effect due to long running time and somewhat short TTLs





NSEC results not surprising and in line with earlier results

Petr Špaček: *Measuring Efficiency of Aggressive Use of DNSSEC- Validated Cache (RFC 8198)* (OARC 28)

So lets look at the NSEC3 case more closely

A few aggressive cache lines after 10k run

NSEC:

```
zwop.nu. 6311 IN NSEC zx.nu. NS DS RRSIG NSEC
zx.nu.   6202 IN NSEC zxc.nu. NS RRSIG NSEC
zxkj.nu. 6318 IN NSEC zxni.nu. NS RRSIG NSEC
zxni.nu. 6211 IN NSEC zxspectrum.nu. NS RRSIG NSEC
```

NSEC3:

```
vtrjob5hmi00l01qlijfl1um17o5o609.cz. 255 IN NSEC3 1 0 0 611fc037b2816e2c VTRLC9D19Q811JPDE7FA1AR9AL8ITFDC NS
vtuq1dpl70p4kad0vsnpf40d5gbej7gm.cz. 72  IN NSEC3 1 0 0 611fc037b2816e2c VTURM0AVDRNSTV7C09JT3MGFDJBRDNDK NS DS RRSIG
vu1adol0churlaehdblbr7p87dvno6q0.cz. 198 IN NSEC3 1 0 0 611fc037b2816e2c VU1DKVPSHPRVTPS6DSF7SOCP7V5IUFM1 NS
vu2g1g610kroqgm0pu2k4sdj4o00d12s.cz. 687 IN NSEC3 1 0 0 611fc037b2816e2c VU2G34IGPH51ANFS7DURS1EI7N3S5RFF NS
```

```
vtrjob5hmi00l01qlijfl1um17o5o609.cz. vtrlc9d19q811jpde7fa1ar9al8itfdc
vtuq1dpl70p4kad0vsnpf40d5gbej7gm.cz. vturm0avdrnstv7c09jt3mgfdjbrdndk
vu1adol0churlaehdblbr7p87dvno6q0.cz. vu1dkvpshprvtps6dsf7socp7v5iufm1
vu2g1g610kroqgm0pu2k4sdj4o00d12s.cz. vu2g34igph51anfs7durs1ei7n3s5rff
```

Each NSEC(3) denies a subset of all possible names

NSEC: a contiguous canonically ordered subset is denied

- A handful of NSEC records can deny many (all?) names that the registry does not allow
- Typos are close (in a canonically ordered way), so one typo might cache an NSEC that also covers another typo
- Names with a suffix appended: idem

Each NSEC(3) denies a set of all possible names

NSEC3: a random subset of labels is denied, the random subset is scattered all over the place!

What is the size of the subset? Warning: rough calculation/estimate ahead!

SHA-1 hash length 160 bits, encoded as 32 chars, 5 bits per char

For **.cz** we saw common prefixes between 3 and 4 chars, close to 4.

If the NSEC3 range has a common prefix of length 4, that means that 4 x 5 = 20 bits fixed

So 160 - 20 = 140 bits can vary.

2^{140} is pretty large, but still covers only $1/(2^{20})$ = 1 millionth of the possible hashes.

Suppose we have 3500 of such NSEC3 records cached

That results in about 1 in 300 chance of hitting a cached NSEC3 record for that zone

NSEC3 aggressive caching: is it effective?

Observations, comparing to the NSEC case

- No small set of NSEC3 record denies a large portion of names the registry considers illegal
- A hash of a name with a typo has no relation with the hash of the original
- A hash of a name wit a suffix has no relation with the hash of the original
- We rely on *luck* to have the NSEC3 record cached
- Many names means many NSEC3 records, so our luck runs out quickly

NSEC3 aggressive cache is only effective:

- No opt-out
- Only if a *substantial* fraction of NSEC3 records are cached
- Is this a surprise?
- Potential approach: only put NSEC3 record into aggressive cache if it covers a substantial number of records
- Easy way to estimate that number: if *owner hash* and *next owner hash* do not have a long common prefix.
- A guess for a useful maximum common prefix length would be 8-12 bits, subject to further analysis
- Special handling of wildcard and closest encloser?
- Under attack: allow more to be cached, needs very generous cache sizes for large zones

Thank you!
Questions?

Contact info:

Otto Moerbeek
Email: otto.moerbeek@open-xchange.com
OARC Mattermost: @ottom
Fediverse: @otto@bsd.network