# Cache Me If You Can: Effects of DNS Time-to-Live (extended)

USC/ISI Technical Report ISI-TR-734

May 2018

Giovane C. M. Moura
SIDN Labs

John Heidemann
USC/Information Sciences Institute

Ricardo de O. Schmidt
University of Passo Fundo

Wes Hardaker
USC/Information Sciences Institute

## ABSTRACT

DNS depends on extensive caching for good performance, and every DNS zone owner must set *Time-to-Live* (TTL) values to control their DNS caching. Today there is relatively little guidance backed by research about how to set TTLs, and operators must balance conflicting demands of caching against agility of configuration. Exactly how TTL value choices affect operational networks is quite challenging to understand for several reasons: DNS is a distributed service, DNS resolution is security-sensitive, and resolvers require multiple types of information as they traverse the DNS hierarchy. These complications mean there are multiple frequently interacting, places TTLs can be specified. This paper provides the first careful evaluation of how these factors affect the effective cache lifetimes of DNS records, and provides recommendations for how to configure DNS TTLs based on our findings. We provide recommendations in TTL choice for different situations, and for where they must be configured. We show that longer TTLs have significant promise, reducing median latency from 183 ms to 28.7 ms for one country-code TLD.

## KEYWORDS

DNS, recursive DNS servers, caching

## 1 INTRODUCTION

The Domain Name System (DNS) [29] is a core component of the Internet. Every web page and e-mail message requires DNS information, and a complex web page can easily require information from a dozen or more DNS lookups. The DNS provides a low-latency, distributed database that is used to map domain names to IP addresses, perform service location lookups, link distributed potions of the DNS together, including in-protocol integrity protection using in-protocol DNS key storage, linking and verification algorithms.

With this central position, often serving as the initial transaction for every network connection, it is not surprising that DNS performance and reliability is critical. For example, DNS performance is seen as a component of web browsing that must be optimized (for example, [48]), and DNS service providers compete to provide consistent, low latency service around the world. Even in less-latency sensitive services, such as the authoritative service for the Root DNS, reducing latency is still a desired goal [46]. DNS *must* always work, and failures of major DNS resolution systems frequently makes public newspaper headlines. In 2016, when a Distributed Denial-of-Service (DDoS) attack led to problems at a DNS provider, it resulted in disruptions to multiple popular public services (including Github, Twitter, Netflix, and the New York Times) [39].

DNS is also often used to associate clients with nearby servers by large content providers [9] and in Content-Delivery Networks (CDNs) [10]. In this role, DNS helps both performance and reliability, associating clients to nearby sites [46, 51], and implementing load balancing, both to reduce latency, and to control traffic to support site maintenance and react to DDoS attacks [33].

It is not surprising that DNS has developed a complex infrastructure, with client software (the *stub resolver*, provided by OS libraries) that contacts *recursive resolvers* (a type of DNS server that can iterate through the DNS tree for answers), which in turn contact *authoritative servers* (which hold the answers being sought). Recursive and authoritative resolvers are often carefully engineered, with pools of servers operating behind load balancers, sometimes in multiple layers [47], often employing IP anycast [1].

Caching is the cornerstone of good DNS performance and reliability. A 15 ms response to a new DNS query is fast, but a 1 ms cache hit to a repeat query is far faster. Caching also protects users from short outages and can mute even significant DDoS attacks [33].

DNS record TTLs (time-to-live values) directly control cache durations [30, 31] and, therefore, affect latency, resilience, and the role of DNS in CDN server selection. While caching DNS servers and anycast have all been extensively studied, surprisingly, *to date there has been little evaluation of*

*TTLs*. Some early work modeled caches as a function of their time-to-live values (the *TTL*) [22], and recent work examined their interaction with DNS [33], but no research provides *recommendations* about what TTL values are good.

Determining good TTL values for DNS is surprisingly challenging. A fundamental tension exists between short and longer TTL values. Short TTLs allow operators to change services quickly, as part of regular operation for load balancing in CDNs, or perhaps to redirect traffic through a DDoS scrubber. Yet, long TTLs reduce latency seen by clients, reduce server load, and provide resilience against longer DDoS attacks.

But not only is there no "easy" optimal setting, but performance of modern DNS is affected by *many* TTLs, since full resolution of a DNS name may require dozens of lookups across several organizations, all potentially using different TTLs. As a distributed database, TTLs are given in both the parent and child at a delegation boundary, and these may differ. In addition, responses come in different flavors, with some values labeled as authoritative, and others labeled as hints ("additional"). Finally, DNS records sometimes depend on the freshness of other records, which can be used as the basis of multiple points of attack. These concerns have been exploited as part of sophisticated DNS hijacking to open user accounts [24].

With only limited academic study, differing operational requirements, and effects that emerge from different components run by multiple parties, it is not surprising that, to our knowledge, *there is no operational consensus for what TTL values are reasonable.* With this lack of consensus and understanding, and a general operational attitude of "if it ain't broke, don't fix it", we see a large range of values in practice (§5), and new deployments have little consistent expertise to rely upon when making TTL choices.

The goal of this paper is to fill this gap, exploring how these many factors influence TTL effectiveness (§2), and provide a recommendation about good TTL values for different scenarios. We use both controlled experiments and analysis of real-world data to make informed recommendations to operators.

Our first contribution shows what the *effective TTL* is as a result of TTLs stored in different places (§3) of across multiple, cooperating records (§4). Second, we examine real-world DNS traffic and deployments to see how current use compares to our evaluation, and how operators choose TTL values and how their choices between short and long TTLs affect latency and operator flexibility (§5).

Finally, we show that DNS TTLs matter, since longer TTLs allow caching, reducing latency and traffic (§6.2). We outline the trade-offs and provide recommendations (§6): those using CDNs, load balancers may require short TTLs (5 or 15 minutes), but most others should prefer longer TTLs (a few hours).

Discussion of our early results with operators prompted increase in their TTLs, and we show that the median latency drops from 183 ms with their earlier short TTLs, to only 28.7 ms now that longer TTLs enable better caching.

We will make the majority of datasets available at no charge. The DITL data is held by DNS-OARC for semi-public use, and the Ripe Atlas datasets are public. Only data from `.nl` cannot be released. Our measurements are all about public network infrastructure and pose no ethical or privacy issues.

## 2 OUR QUESTION: WHICH TTLS MATTER?

While DNS caching seems simple on its face, with each record cached up to a given time-to-live, the reality is more complex: DNS records come from several places and resolution requires traversing multiple names and types. We next look systematically at each source of information and determine which, in practice, takes priority.

First, *records are duplicated in multiple places*, sometimes with different TTLs. Specifically, DNS records that cross delegation boundaries are in both the parent and the child zone and can have different TTLs. In §3 we examine if recursives in the wild prefer TTL values provided by the parent or child.

Second, *resolution of a fully qualified domain name (FQDN) requires identifying authoritative servers (NS records) and their IP addresses (A or AAAA records) for each part of the FQDN*. FQDN traversal raises two factors. First, communicating with an authoritative server requires knowing its IP address(es), but the NS and A/AAAA records for it may also have different TTLs. Second, records for it may be *in bailiwick* (when they are under the domain being served, so `ns.example.org` is in bailiwick of `example.org` [20]) or *out of bailiwick* (`ns.example.com` would not be in bailiwick of `example.org`). These factors interact: some recursive resolvers discard in-bailiwick A/AAAA records when the NS record expires, as we show in §4.

The answer to these questions should be given in the DNS specifications. Unfortunately early specifications were somewhat informal, and implementations varied in practice. The original DNS specifications give the child's TTL precedence [29, 32], and later clarifications explicitly ranked Authoritative Answers over glue [14]. DNSSEC [7, 8] confirms that authoritative TTL values must be enclosed in and verified by the signature record, which must come from the child zone. Thus our question is: *Do wild resolvers follow these specifications for TTL priorities?*

Answering these questions is also important to understand *who ultimately controls a zone's caching?*

| Q / Type | Server | Response | TTL | Sec. |
|---|---|---|---|---|
| .cl / NS | k.root-servers.net | a.nic.cl/NS | 172800 | Auth. |
| | | a.nic.cl/A | 172800 | Add. |
| | | a.nic.cl/AAAA | 172800 | Add. |
| .cl/NS | a.nic.cl | a.nic.cl/NS | 3600★ | Ans. |
| | | a.nic.cl/A | 43200 | Add. |
| | | a.nic.cl/AAAA | 43200 | Add. |
| a.nic.cl/A | a.nic.cl | 190.124.27.10/A | 43200★ | Ans. |

**Table 1: a.nic.cl. TTL values in parent and child (★ indicates an authoritative answer), on 2019-02-12.**

## 3 ARE RESOLVERS PARENT- OR CHILD-CENTRIC?

We first examine how DNS handles *records that are served from multiple places*. The DNS is a distributed database with portions of the hierarchy (*zones*) managed by different organizations through *delegation*. *Glue records* duplicate content from a child zone in the parent, either for convenience or out of necessity, if the authoritative server for the child zone is named only in that zone (in bailiwick).

We examine this question with a case-study and wild traffic observed from the edge and from authoritative servers for a country code TLD. We reach two key results of cross-zone TTLs: first, **most recursive resolvers are child-centric**, trusting the TTL in the child zone's authoritative server over the glue in the parent zone. Depending on the measurement technique, just 52% to 90% of queries are child-centric. However, our second finding is that **enough queries are parent-centric that they cannot be ignored**. The other queries are are parent-centric, and although they are not the majority, there are enough (10 to 48%) that one *must* set TTLs the same in both parent and child to get consistent results. In cases where operator is *without control of the parent zone's TTL, resolvers will see a mix of TTLs for that zone.*

### 3.1 Case Study: Chile's .cl

To explore this question of whether the parent or child's TTL in the hierarchy is "believed more frequently", we first look at Chile's country-code TLD, .cl. Resolving this name involves three authoritative servers as shown in Table 1.

We see three *different* TTLs: 172800 s at the root, 3600 and 43200 s at the .cl authoritative servers, and 43200 s when we explicitly ask for the address record for the name server's IP address. Which TTL is used depends implementation choices of the recursive resolver, which may prefer either the parent or child's TTL.

A second factor is that response components are returned in different DNS message sections [29], and may be treated differently by different implementations. Records are marked *authoritative* (.cl's NS record at the root), as *answer* (.cl's NS record at .cl), or *additional* (A records attached to the NS response at .cl).

Though answers from the child, when the AA flag is set, have higher priority, when it resolves the domain example.cl, it may choose to never contact the child, and simply relies on the authority and additional records returned by the parent. (For example, to resolve example.cl, a resolver can use the A record of a.nic.cl as provided by the Roots in Table 1.) This question has been previously defined as resolvers' centricity [6, 11, 16, 40]: resolvers using the TTL provided by the parent authoritative (such as the Roots for .cl) servers are defined as *parent-centric*, while *child-centric* resolvers will use the child authoritative.

Resolvers employing in-resolver authoritative mirroring technologies, such as RFC7706 [25] or LocalRoot [18], or serving stale content [26] (*i.e.,* answering queries past TTL expiration only when the NS records for the given domain are unresponsive) will exhibit different perceived TTL caching behaviors. In the former case, resolvers implementing RFC7706 or LocalRoot, entire zones will be transferred into an authoritative server that runs in parallel with a recursive resolver; no queries to these zones will likely be seen exiting the recursive resolver ([18]), though questions to their children will still be sent. For the latter case, resolvers serving stale content, outgoing requests will likely continue to be seen on the wire, but even when unanswered, resolvers will continue serving (expired) answers to clients.

This example illustrates the complexity of what TTLs implementations use. We next look at how these rules work in practice.

### 3.2 Uruguay's .uy as Seen in the Wild

We next consider Uruguay's country-code TLD .uy. We select Uruguay because it's ccTLD has two very different TTL values its NS record: 172800 s at the root, and only 300 s in their own authoritative server (as of 2019-02-14), and 120 s for that server's A record.

These large differences allow us to study their effects on caching from "the wild". We use RIPE Atlas [42, 43], measuring each unique resolver as seen from their ~10k probes physically distributed around the world. Atlas Probes are distributed across 3.3k ASes, with about one third hosting multiple *vantage points* (VPs). Atlas software causes each probe to issue queries to each of its local recursive resolvers, so our VPs are the tuple of probe and recursive. Due to that, we end up with more than 15k VPs.

We make queries first for the NS record of .uy, then the A records of its authoritative server a.nic.uy. In each case, we query from each VP every 10 minutes (twice the shortest TTL of the NS records), for either two or three hours (for the NS or A records). For each query, we look for the TTL

| | .uy-NS | a.nic.uy-A | google.co-NS | .uy-NS-new |
|---|---|---|---|---|
| Frequency | 600s | 600s | 600s | 600 |
| Duration | 2h | 3h | 1h | 2h |
| Query | NS .uy | A a.nic.uy | NS google.co | NS .uy |
| TTL Parent | 172800 s | 172800 s | 900 s | 172800 s |
| TTL Child | 300 s | 120 s | 345600 s | 86,400 |
| Date | 20190214 | 20190215 | 20190304 | 20190304 |
| Probes | 8963 | 8974 | 9127 | 8682 |
| valid | 8863 | 8882 | 9034 | 8536 |
| disc | 100 | 92 | 93 | 96 |
| VPs | 15722 | 15845 | 16078 | 15325 |
| Queries | 189506 | 285555 | 97213 | 184243 |
| Responses | 188307 | 282001 | 96602 | 184243 |
| valid | 188225 | 281931 | 96589 | 184209 |
| disc. | 82 | 70 | 3 | 34 |

**Table 2: Resolver's centricity experiments. Datasets available at [41].**

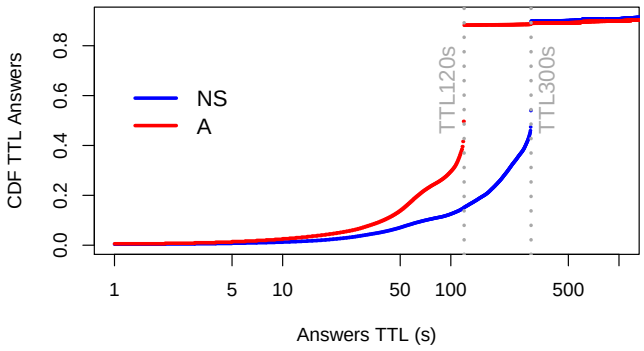



**Figure 1: Observed TTLs from RIPE Atlas VPs for .uy-NS and a.nic.uy-A queries.**

as seen in the answer section of the DNS response, obtaining about 190k and 280k valid responses from .uy-NS and a.nic.uy-A experiments, respectively. Table 2 summarizes the experiments.

Figure 1 shows the CDF of the valid TTLs from all VPs. The vast majority of responses follow the child's value, not the parent's: 90% of .uy-NS are less than 300 s, and 88% of a.nic.uy-A are less than 120 s. We conclude that most resolvers are *child-centric*, preferring the TTL of the authoritative server (following §5.4.1 of RFC2181 [14]).

Roughly 10% of resolvers appear to be *parent-centric*, following the 2-day TTL of the root zone (or these resolvers are manipulating TTLs [17, 33]). In fact, about 2.9% of .uy-NS and 2.2% of a.nic.uy-A show the full 172800 s TTL. Among those resolvers, we found that some probes used OpenDNS public resolvers [37]. We later sent queries to it and confirmed that they are indeed parent-centric, but *only* for the Root zone (probably implementing RFC7706).

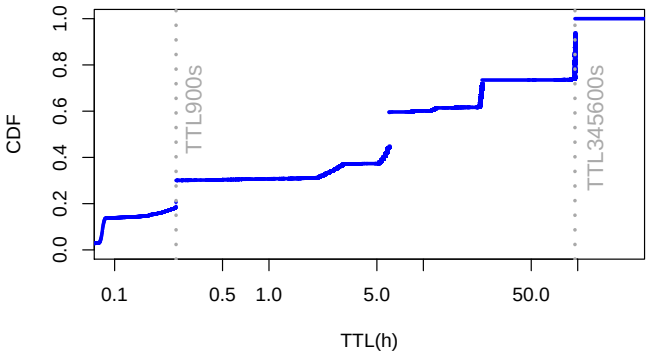



**Figure 2: Observed TTLs from RIPE Atlas VPs for .google.co-NS queries.**

## 3.3 A Second-level Domain in the Wild

To confirm our observations that client-centric TTL preferences extend past top-level domains, we repeat the experiment from §3.2 for google.co, a popular second-level domain. The domain google.co has two TTL values for its NS records: 900 s from the parent servers (.co), and 345600 from the actual authoritative servers ns[1-4].google.com (as of 2019-03-05). We query every 600 s, for one hour (Table 2).

Figure 2 shows the CDF of observed TTLs for this second-level domain, for 16k VPs. About 70% of all answers have TTLs longer than 900 s—results that must come from the child authoritative server. About 15% of all answers, many using Google public DNS, have TTLs of 21,599 s, suggesting TTL capping. About 9% of all answers have a TTL of exactly 900 s, suggesting a fresh value from the parent authoritative server.

This experiment shows that second-level domains are also most often child-centric in selecting TTLs.

## 3.4 Confirming Broadly with Passive Observations of a Country's TLD

Prior sections showed that specific domains are mostly client-centric, observing from the authoritative side, looking at *who is querying* and *what strategy they use* (parent- or child-centric). Here we study passive data for the the Netherlands zone, .nl.

At the time of this experiment (2019-03-06 to -07), the .nl ccTLD had four authoritative servers (sns-pb.isc.org and ns[1-3].dns.nl), each with multiple IP anycast sites [2]. We gather DNS data from ns[1,3].dns.nl servers using *(anonymized for review)*, which saw more than 6.5M queries for the two-day period. The A records for ns[1,3].dns.nl are listed in the parent zone (the root zone) with glue records containing TTL values of 172800 s (2 days). The children's authoritative servers, however, contain only a 3600 s (1 hour) TTL.

We examine query interarrivals for each resolver to classify that resolver as parent- or child centric. We find about

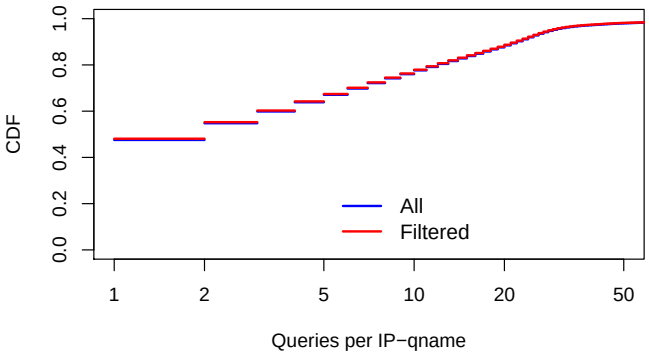



**Figure 3: CDF of A queries per resolver/query name**

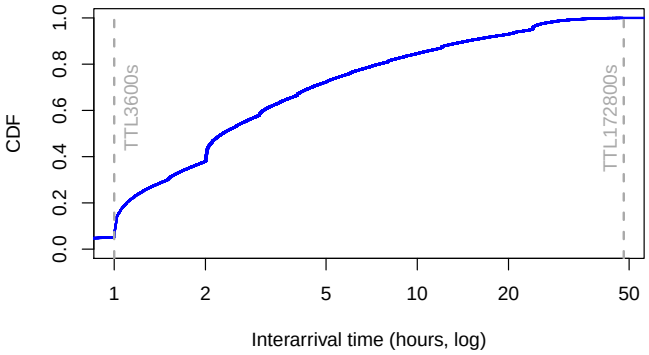



**Figure 4: CDF of interarrival time from each resolver/query-name for `.nl`.**

205k unique resolver IP addresses, providing 13× more VPs than our experiment using RIPE Atlas (§3.2 and §3.3).

We see 368k groups of *resolver, query-name* pairs, in which *query-name* is one of the four NS records for `.nl` pairs, and we compute the interarrival time between queries for each group.

Figure 3 shows the CDF of the number of queries for each group for the aggregate queries (the solid blue "all" line), and for those queries where the interarrival time is more than 2 s (the red "filtered" line). This filtering aims at removing duplicate queries that are retransmissions, but we see that the curves are essentially identical.

More than half of the groups appear to be child-centric, since 52% send more than one query over the two days, suggesting they are following the shorter child TTL. Another possible explanation is that some recursive resolvers cap TTLs to less than 2 days (some versions of BIND [21], however, use one week as the default maximum caching time).

Just less than half (about 48%) send only one query during observation. Since we only observe two of the four authoritative servers, it is possible these clients made queries to non-observed authoritative servers. (It is known that clients tend to rotate betweens servers [34]). Another possibility is that these clients did not need to handle multiple queries for names under `.nl`.

To investigate if these resolvers, which sent only one query per query-name, are indeed parent-centric, we extract the unique source IPs from the groups that sent only one query; which gives us 122562 unique IP addresses. Around 14% of these IPs are also present in groups that sent more than one query for other names. For example, an IP that queries once for `ns1.dns.nl`, but queries 3 times for `ns2.dns.nl`. This suggests that at least 14% of the resolvers in this group behave as child centric as well.

We gain greater confidence how many resolvers are child-centric by looking at the median interarrival time for resolvers that send multiple queries for the same name in Figure 4. Even observing only two of the four authoratives, we can conclude that most resolvers use the child TTL. We also see "bumps" around multiples of one hour. We believe that these bumps are resolvers returning to the same server after the TTL expires. Finally, the small bump at 24 hours suggests resolvers that cap caches at 1 day.

We conclude that, even when observed from the authoritatives, at least half recursive resolvers are child-centric.

# 4 THE MANY TTLS IN RESOLVING A FULLY-QUALIFIED DOMAIN-NAME

We next turn to the second problem from §2: how do the different parts of a FQDN, different records (NS and A or AAAA), different answer types (authoritative answer, authority, and additional), and different server configurations (in and out-of-bailiwick) change the effective TTL lifetime of the originating request? We answer these questions with two controlled experiments: one with an in-bailiwick server, `ns1cachetest.net`, and the other with an out-of-bailiwick server.

The key results of this section are to show that it does matter where the authoritative server is located in the DNS hierarchy. For **in-bailiwick authoritative servers glue records drive cache lifetimes**, and the TTLs of the IP address and authoritative server are frequently linked (a still valid A record will still expire when its covering NS record expires). By contrast, **out-of-bailiwick servers use cached information about authroritative server for the full TTL lifetime**.

## 4.1 Experimental Setup

Our experiments use a test domain (`cachetest.net` from [33]) over which we have complete control. This domain has two authoritative servers: `ns[12]cachetest.net`, as can be seen in Figure 5, both running Debian and BIND 9.1 on EC2 in Frankfurt, Germany.

We add this domain to the parent `.net` zone, which requires adding both NS records in `.net` for our domain and

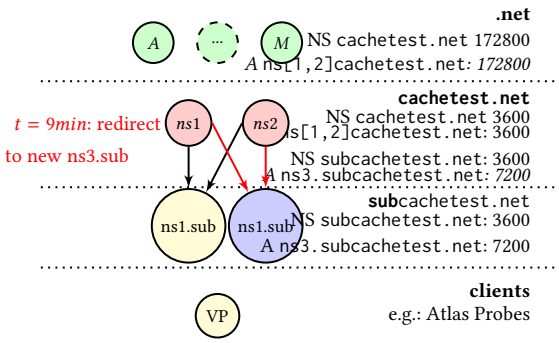



**Figure 5: TTLs and domains for in-bailiwick experiment [41]. Italics indicate glue records.**

| | in-bailiwick | out-of-bailiwick |
|---|---|---|
| Frequency | 600 s | 600 s |
| Duration | 4h | 4h |
| Query | AAAA probeid.sub.cachetest.net | |
| Date | 20190315 | 20190314 |
| Probes | 9131 | 9150 |
| Probes (val.) | 8864 | 9053 |
| Probes (disc.) | 267 | 97 |
| VPs | 15618 | 16103 |
| Queries | 367060 | 387037 |
| Queries (timeout) | 39471 | 10436 |
| Responses | 341707 | 368478 |
| Responses (val.) | 340522 | 366853 |
| Responses (disc.) | 1185 | 1625 |

**Table 3: Bailiwick experiments [41].**

glue records for the addresses of our authoritative servers (italic in Figure 5). By default, records in .net have a TTL of 172800 s, or 2 days.

Our cachetest.net one is run on authoritative servers running in EC2 VMs. (We run our own DNS servers in our own VMs and do not use Amazon's Route53 [4] hosted DNS.) We set the TTLs for the NS and A records in our zone to 3600 s.

As can be seen in Figure 5, recursives resolvers ill find two different TTLs for the same record (at both parent and child). Even thought most resolvers are expected to be child-centric (§3), for sake of precision, we decided to rule out this influence by creating the subcachetest.net zone. We configure this subzone in two different experiments using a third dedicated EC2 VM also in Frankfurt.

## 4.2 Effective TTLs for Servers *Inside* the Served Zone

We first look at how resolvers handle authoritative servers with names in the served zone—those that are *in-bailiwick*. We show that most recursives require both fresh NS and A records, and they re-fetch even valid A records when the NS record expires.

For this experiment, we configure ns1.subcachetest.net as an authoritative server for our subzone (subcachetest.net). We set the TTL of its NS record to 3600 s and its A record TTL to 7200 s. These TTLs are consistent in both the parent and child zones, so recursives will have the same cache duration regardless of which they prefer.

At $t$ = 9 min, we *renumber* ns3.subcachetest.net, changing its IP address to a different EC2 VM. This new VM also serves this zone, but with some changes to the records, so we can determine if the old or new authoritative server is used and thus determine how caching works.

We test this potential dependency by querying the AAAA record of PROBEID.subcachetest.net from all RIPE Atlas VPs every 600 s, watching for the returned answer to change. Since the authoritative's NS and A records have different TTLs, the time at which the response changes reveals the caching behavior of the recursive resolver. We are looking to see if the NS and A records are independent, or if they are linked, causing the A record to expire earlier than it needs to, when the NS record times out. To ensure the test answer is not cached, PROBEID is inserted as a unique identifier for the quering RIPE Atlas Probe [44] (see Table 3), and the AAAA records have a TTL of 60 s, one tenth our probe interval, which is record type used in our queries.

Table 3 shows the results of about 340k valid responses from 15.6k RIPE Atlas VPs. (We discard responses that included NS records, SERVFAIL, and others that did not include the answer we expected.)

Figure 6 is a timeseries chart of the AAAA answers received by our vantage points. We count how many responses were sent by each authoritative server (original and new), aggregated to 10-minute bins. In this figure, the first arrow down shows the time when we renumber the IP address of the authoritative server (after 9 minutes).

This figure shows that before renumbering (at 9 minutes), all queries are answered by the original server. From 9 to 60 minutes we see that some resolvers (the dark blue bars) continue to use the original server, showing they have cached and trust its A and NS records. Other resolvers (light yellow bars) switch to the new server, suggesting they refetched the new A record. We see that most resolvers trust their cache up to the 1-hour TTL.

After one hour the NS records begin to expire. Over the next hour we can test if the recursive resolver trusts its already-cached, yet-still-valid TTL for the A record, or if it drops it and refreshes it anyway and discovers the new server. We see that with an in-domain server, *very few recursives* continue to trust the cached A record—*in-domain servers have tied NS and A record cache times in practice*. Specifically,

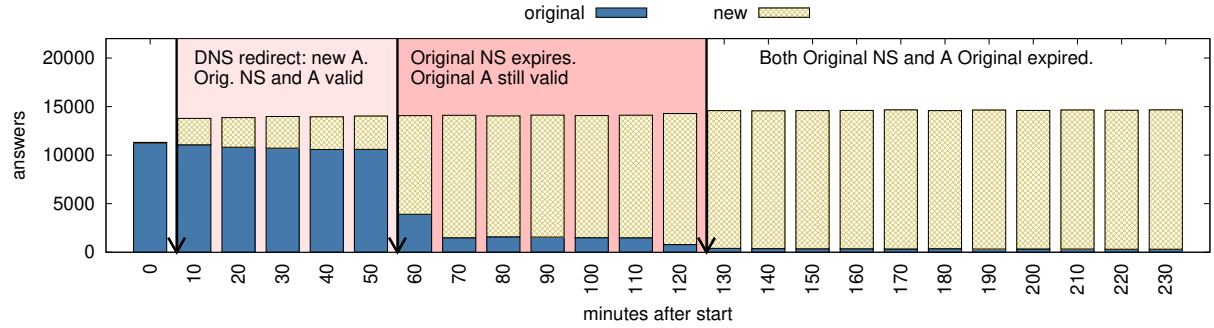



**Figure 6: Timeseries of answers for in-bailiwick experiment**

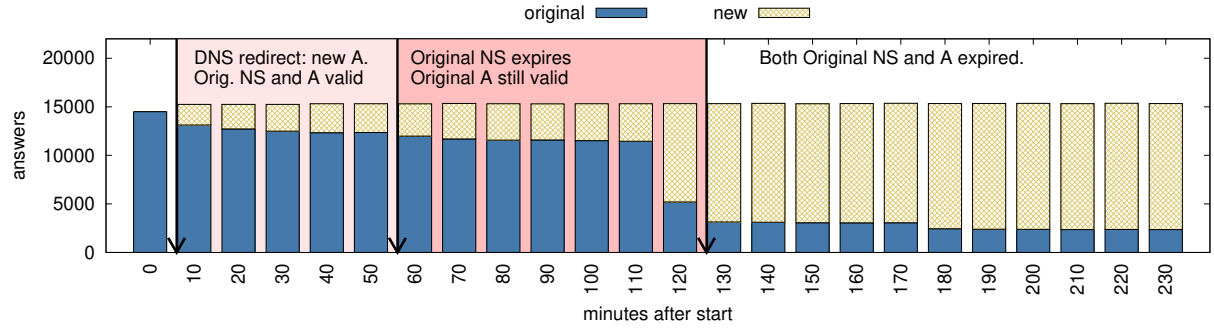



**Figure 7: Timeseries of answers for out-of-bailiwick experiment**

about 90% of the resolvers that queried on the first round (blue on $t = 0$) refresh both the NS and A records at $t = 60$, switching to the new server. We confirm this result from the authoritative side in §4.4.

After two hours, both the NS and A should expire, so we expect *all* recursives to switch to the new server.

We see that 305-390 VPs (about 2.25% of the total) continue with the old resolver, a phenomena known as "sticky resolvers" [34].

## 4.3 Effective TTLs for Servers *Outside* the Served Zone

We now move to what effective TTLs are seen when the authoritative servers are *outside* the served zone (*out-of-bailiwick* servers). In this case, the server's IP addressed is trusted even when the NS record is expired.

For this experiment, we replace both in-bailiwick authoritative servers with `ns1.zurrundeddu.com`. Since it is not with the `cachetest.net` domain, it is an out-of-bailiwick server. As before, the NS records in the glue has a TTL of 3600 s, and the A glue record has a TTL of 7200 s. As before, we renumber the A record of `ns1.zurrundeddu.com` after 9 minutes. (The `.com` zone supports dynamic updates and we verify this change is visible in seconds.) Finally, we query the AAAA record from 16k RIPE Atlas VPs, every 600 s and watch for changes (Table 3).

Figure 7 shows how VPs react to the changed records, and Table 3 provides details about the experiment. Comparing the in- and out-of-bailiwick cases (Figure 6 and Figure 7), we see that VPs trust the A record for the old authoritative server for nearly its full cache lifetime, out to 120 minutes, not just 60 minutes. This result shows that *most recursive resolvers trust cached A records when served from different zones (out-of-bailiwick)*, but not in-bailiwick servers.

## 4.4 Confirmation from the Authoritive Side

We investigate in this section *why* results from in-bailiwick and out-of-bailiwick differ that much. We analyze traffic obtained at the authoritative servers.

First, we compare the responses for the queries issued by RIPE Atlas VPs. The responses send to RIPE Atlas for the in-bailiwick scenario had, besides the AAAA records in the *answers* section and NS record in the *authority section*, an A record (glue) of the NS records found in the *additional*. In comparison, the out-of-bailiwick scenario had no *additional* section, as the zone had no glue.

For out-of-bailiwick servers, the resolver must explicitly fetch the address of the authoritative servers. This difference affects *who* provides that information: for in-bailiwick, it comes from the parent based on the glue records, while

for out-of-bailiwick it is from NS's authoritative server. Resolvers therefore get different TTLs because parent and child provide different TTLs.

## 5 TTLS IN THE WILD

While there is limited guidance for setting TTLs, we can look at how TTLs are set *in the wild* to see if there is consensus. We look both at top-level and popular second-level domains, and report on our discussion of these results with operators.

### 5.1 Crawling TLDs and Popular Domains

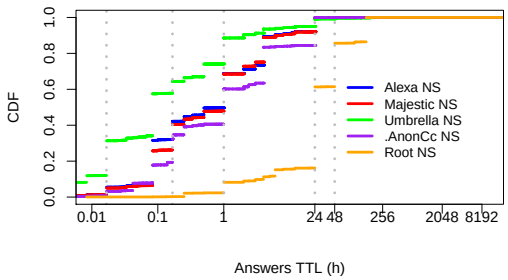
To evaluate how TTL values are used in the wild, we study five data sources; the root zone, the country-level domain of the Netherlands, and "top" lists from Alexa [3], Majestic [28], and Umbrella [50]. We use root data from 2019-02-13, and all other data from 2019-02-06. While it is know that top lists show considerable churn [45], our results provide a snapshot of one instant, and the diversity of sources allow us to evaluate trends.

*Methodology:* For each list, we retrieve TTLs of several DNS records (NS, A, AAAA, MX, and DNSKEY) directly from both the *parent* and *child* authoritative servers, measuring from Amazon EC2 in Frankfurt. Here we report results only for the child zone, since that reflects the operator's intent, and most recursives are child-centric (§3). (A full comparison of parent and child is future work, but we know that the TTL of `.nl` are 1 hour, so we know that about 40% of `.nl` children have shorter TTLs, as can be seen in Figure 8a.)

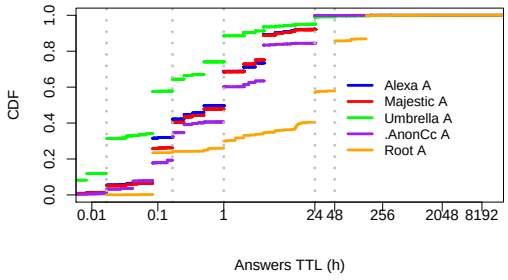
*Results:* Table 4 shows the sizes of each crawl, and how many replies we see each record and list. Most lists have high response rates, with Umbrella as an exception with only 78% responding. The Umbrella list includes many transient names that point to CDNs or cloud instances (for example, wp-0f21050000000000.id.cdn.upcbroadband.com). We report the number of unique records, and higher ratios of unique records show greater levels of shared hosting. (The top lists reflect diverse users and diverse hosting, while `.nl` reflects a large number of static domains with relatively little use that use low-cost shared hosting.)

Figure 8 shows the CDFs of TTLs of authoritative answers for each record type. There are no A or AAAA records for TLDs in root zone, so there we report the A and AAAA records of their respective NS servers. For other lists of SLDs or full names, we do not do this indirection and report only A records for the given name. Our first observation is that *TTLs show a large variation in values*, from 1 minute to 48 hours, for all lists and record types.

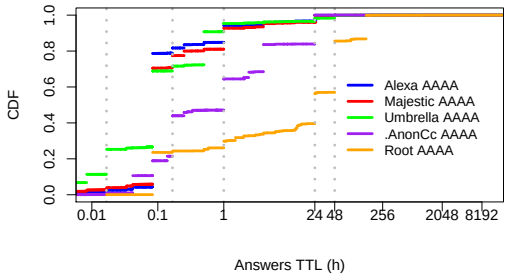
Second, we see some trends in different lists. In general, the times reflect human-chosen values (10 minutes and 1, 24, or 48 hours). In the root, about 80% of records have TTLs of 1 or 2 days. For Umbrella, on the other hand, 25% of its domains with NS records under 1 minute. This difference



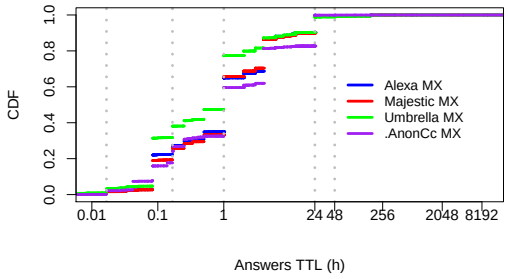

**(a) NS-TTL**



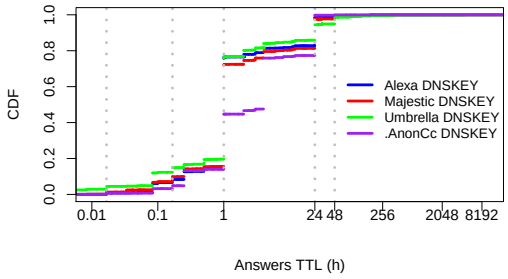

**(b) A-TTL**




**(c) AAAA-TTL**




**(d) MX-TTL**




**(e) DNSKEY-TTL**

**Figure 8: CDF of TTLs per record type, for each list.**

| | Alexa | Majestic | Umbre. | .nl | Root |
|---|---|---|---|---|---|
| format | 2LD | 2LD | FQDN | 2LD | TLD |
| domains | 1000000 | 1000000 | 1000000 | 5582431 | 1562 |
| responsive | 988654 | 928299 | 783343 | 5454833 | 1535 |
| discarded | 11346 | 71701 | 216657 | 345479 | 27 |
| *ratio* | 0.99 | 0.93 | 0.78 | 0.94 | 0.97 |
| Date | 2019-02-06 | | | | 2019-02-13 |
| NS | 2479257 | 2430773 | 855147 | 14184460 | 7289 |
| *unique* | 269896 | 234356 | 106475 | 74619 | 4169 |
| *ratio* | 9.19 | 10.37 | 8.03 | 190.09 | |
| A | 1247139 | 1069314 | 1126842 | 5389560 | 4145 |
| *unique* | 572689 | 539301 | 451220 | 274920 | 3188 |
| *ratio* | 2.18 | 1.98 | 2.50 | 19.60 | |
| AAAA | 282818 | 215935 | 287069 | 2127664 | 3740 |
| *unique* | 106235 | 97545 | 139456 | 134751 | 2951 |
| *ratio* | 2.66 | 2.21 | 2.06 | 15.79 | |
| MX | 1697001 | 1532026 | 522089 | 7494383 | 88 |
| *unique* | 480787 | 435455 | 130751 | 2157676 | 35 |
| *ratio* | 3.53 | 3.52 | 3.99 | 3.47 | |
| DNSKEY | 42950 | 38262 | 11731 | 3800612 | – |
| *unique* | 26274 | 25275 | 6838 | 3597613 | – |
| *ratio* | 1.63 | 1.51 | 1.72 | 1.06 | |
| CNAME | 45228 | 2493 | 344500 | 10666 | – |
| *unique* | 3592 | 1512 | 166230 | 3509 | – |
| *ratio* | 12.59 | 1.65 | 2.07 | 3.04 | |

**Table 4: Datasets and RR counts (child authoritative)**

| | Alexa | Majestic | Umbrella | .nl | Root |
|---|---|---|---|---|---|
| NS | 4524 | 4187 | 1365 | 3414 | 0 |
| A | 896 | 575 | 529 | 673 | 0 |
| AAAA | 244 | 1549 | 116 | 45 | 0 |
| MX | 506 | 374 | 211 | 266 | 0 |
| DNSKEY | 0 | 2 | 12 | 15 | 0 |
| *unique* | 5385 | 6202 | 1955 | 4047 | 0 |

**Table 5: Domains with TTL=0 s, per Record Type**

reflects the list populations—the root is slowly changing, and changes are carefully monitored and managed. Umbrella instead reflects many cloud and CDN names, that are often transient and changing as load comes and goes and cloud instances are created and discarded.

We also see variation across record type. NS and DNSKEY records tend to be the longest-lived (Figure 8a), while IP addressees are the shortest (Figure 8b and Figure 8c). These reflect service dynamics: changing authoritative servers is an administrative action with careful planning, while server addresses are often dynamic with automated creation in clouds and CDNs.

This diversity in TTL choices of major domains suggests some combination of differing needs and lack of consensus in TTL choice.

| | Alexa | Majestic | Umbre. | .nl | Root |
|---|---|---|---|---|---|
| responsive | 988654 | 928299 | 783343 | 5454833 | 1535 |
| CNAME | 50981 | 7017 | 452711 | 9436 | 0 |
| SOA | 12741 | 8352 | 59083 | 12268 | 0 |
| responsive NS | 924932 | 912930 | 271549 | 5433129 | 1535 |
| Out only | 878402 | 873447 | 244656 | 5417599 | 748 |
| *ratio* | 95.0% | 95.7% | 90.1 | 99.7% | 48.7% |
| In only | 37552 | 28577 | 20070 | 12586 | 654 |
| Mixed | 8978 | 10906 | 6823 | 2941 | 133 |

**Table 6: Bailiwick distribution in the wild.**

*5.1.1 TTL 0 s.* TTL values may range from 0 s to ~68 years ($2^{31} - 1$ s) [32], but in practice most TTLs, in the wild, are under two days (Figure 8).

While not an error per se, a TTL of 0 s effectively *undermines caching* at the resolvers. We show in Table 5 the counts of domains with TTL equal zero. We see that few domains are configured this way – which is not recommended, given that by undermining caching, it automatically increases latency to users and provides little resilience in case of DDoS attacks [33].

*5.1.2 Bailiwick configuration in the wild.* We have seen in §4 how bailiwick affects the choice of TTLs for a given record. We now investigate how domains are configured in the wild with regards bailiwick.

Table 6 summarizes the results. We start with the *responsive* domains (obtained from Table 4), which are domains that responded to at least one of our queries (in regardless of query type). To evaluate how domains are configured, we consider only NS queries that had NS records in the answers (we disregard domains that either returned a CNAME or SOA records to NS queries). We see that the majority of domain names (responsive NS row) remain, with exception of Umbrella list, given it uses long FQDNs also from clouds and CDNs.

Then, we proceed to analyze how these domains are configured with regardless their bailiwick setup. For the popular lists, we see that the *far majority* (>90%) are configured with out-of-bailiwick NSes *only*. The exception to that is the list of TLDs (Roots), which roughly half are out-of-bailiwick and the other half are either only in-bailiwick or mixed.

## 5.2 Discussions with Operators

Our crawl of the root zone (§5.1) showed 34 TLDs (including 8 country-code) with NS TTLs less than 30 minutes, and 122 TLDs with NS TTLs under 120 minutes. These short TTLs are only partially effective because of parent-centric resolvers (§3), and they prevent caching that can help latency (§6.2), and increase DDoS vulnerability [33].

We reached out to operators of of the eight ccTLDs, asking them why they chose such short TTLs. Five of them

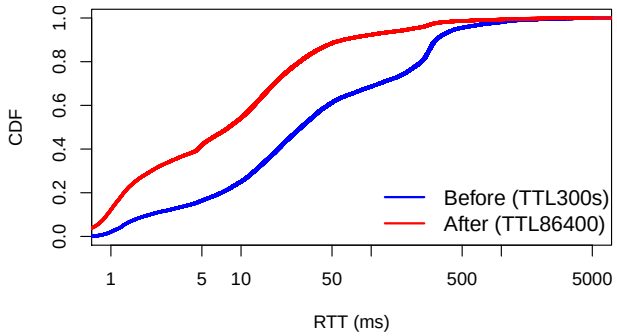



**Figure 9: RTT from RIPE Atlas VPs for NS .uy queries before and after changing TTL NS records.**

responded, with three stating that they had not considered the implications of such short TTLs. Later three *increased their NS records TTL* to 1 day (from 30 s 300 s, and 480 s) . Two said the short TTLs were intentional to account for planned infrastructure changes. The final reply was from a large operator who stated they kept the TTL values in place when they took over service.

One should be cautious in drawing conclusions from such a small sample but while some operators intentionally use small TTLs, many appear to have not carefully considered the implications and are interested in considering longer TTLs.

### 5.3 Early Feedback from Uruguay's .uy

One operator we approached was Uruguay's .uy ccTLD. Our study of Uruguay's ccTLD showed that in early 2019 they had very different TTLs between their parent and child, with authoritative TTLs of only 5 minutes while the root zone defaults to 2 days (300 s vs. 172800 s!), having, during the time of our analysis, 8 NS records (5 in-bailiwick, 3 out). After sharing our early results with them, on 2019-03-04 they changed their authoritative TTLs to one day (86400 s).

Uruguay's .uy change provides a natural experiment to test the effects of different TTLs on DNS latency. Our studies had measurements from RIPE Atlas VPs both before and after this change (see uy-NS and uy-NS-new in Table 2). We measure the response time for a .uy/NS query from around 15k VPs, querying for two hours every 600 s. Since .uy is a country-level TLD, it may be cached, so this study reflects a dynamic snapshot remaining TTL.

*Results:* Figure 9 shows the CDF of query response times for .uy before, with a short TTL (the top, red line), and after, with long TTLs (the bottom, blue line). With short TTLs, .uy often falls out of the cache, and the median response time is 28.7 ms. With long TTLs .uy remains in the cache and so many queries are handled directly by the recursive, providing an 8 ms response time.

Differences in tail latency are even larger: at the 75%ile, longer RTTs have median of 21 ms compared to 183 ms with short RTTs; at the 95%ile, longer RTTs have a median of 200 ms compared to 450 ms, and, at 99%ile, these values raise to 1375 ms and 678 ms, respectively.

This natural experiment shows the large benefit to user latency from increased caching and long TTLs. We do not have access to authoritative traffic at .uy, so we cannot evaluate traffic reduction, but it too is likely substantial (we evaluate traffic reduction due to longer TTLs in §6.2) .

## 6 RECOMMENDATIONS FOR DNS OPERATORS

We next consider recommendations for DNS operators and domain owners, about TTL durations and the other operational issues.

### 6.1 Reasons for Longer or Shorter TTLs

TTLs in use range from as short as 5 minutes, to a few hours, to one or two days (§5.1). This wide range of time values seen in TTL configurations is because there are many trade-offs in "short" vs. "long", and which factors are most important is specific to each organization. Here are factors operators consider:

**Longer caching results in faster responses:** The largest effect of caching is to allow queries to be answered directly from recursive resolvers. With a cache hit, the recursive can respond directly to a user, while a cache miss requires an additional query (or queries, in some cases) to authoritative servers. Although a query to the authoriative is usually fast (less than 100 ms), a direct reply from the recursive resolver is much faster. We see this effect for Uruguay's .uy in §5.3, and demonstrate it with controlled experiments in §6.2.

**Longer caching results in lower DNS traffic:** caching can significantly reduce DNS traffic. However, DNS queries and replies are quite small, and DNS servers are relatively lightweight. Therefore, costs of DNS traffic are likely smaller than costs of web hosting or e-mail. We evaluate this effect in §6.2.

**Longer caching results in lower cost if DNS is metered:** Some DNS-As-A-Service providers charges are metered, with a per query cost (often added to a fixed monthly cost). Even if incremental costs are small relative to fixed charges, caching can reduce this cost.

**Longer caching is more robust to DDoS attacks on DNS:** DDoS attacks on a DNS service provider [19] harmed several prominent websites [39]. Recent work has shown that DNS caching can greatly reduce the effects of DDoS on DNS, provided caches last longer than the attack [33].

**Shorter caching supports operational changes:** An easy way to transition from an old server to a new one is to change the DNS records. Since there is no method to

remove cached DNS records, the TTL duration represents a necessary transition delay to fully shift to a new server, so low TTLs allow more rapid transition. However, when deployments are planned in advance (that is, longer than the TTL), then TTLs can be lowered "just-before" a major operational change, and raised again once accomplished.

**Shorter caching can help with a DNS-based response to DDoS attacks:** Some DDoS-scrubbing services use DNS to redirect traffic during an attack [35]. Since DDoS attacks arrive unannounced, DNS-based traffic redirection requires the TTL be kept quite low at all times to be ready to respond to a *potential* attack.

**Shorter caching can with DNS-based load balancing:** Many large services use DNS-based load balancing (for example, the Akamai CDN [10] and Bing search engine [9]). Each arriving DNS request provides an opportunity to adjust load, so short TTLs may be desired to react more quickly to traffic dynamics. (Although many recursive resolvers have minimum caching times of tens of seconds, placing a limit on agility.)

Organizations must weigh these trade-offs to find a good balance; we propose two recommendations next.

## 6.2 Caching Will Reduces Query Volume and Latency

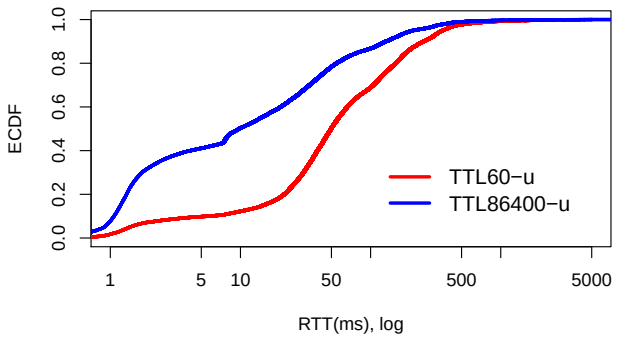
We know that caching will reduce query volume at the authoritative server and improve latency to users. Exactly how much depends on the workload: who queries, when, and from where. We saw a significant reduction in latency for Uruguay in in §5.3. We next study those questions with a controlled experiment.

*Methodology:* We carry out five experiments listed in Table 7. We use DNS servers at Amazon EC2 in Frankfurt, with short (60 s) and long (84,400 s) TTLs, and we use anycast (Route53, with 45 global sites at experiment time) with 60 s TTLs.

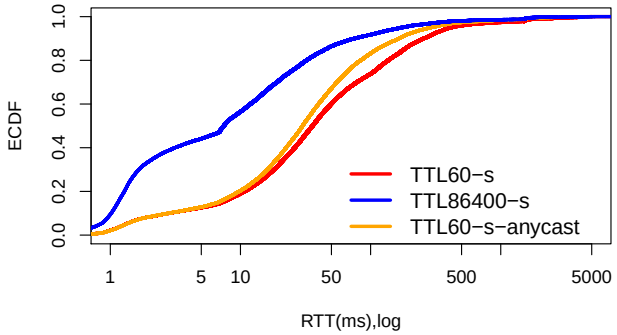
We place queries to a test domain (unique to this experiment) from 15k Atlas VPs to different types of DNS configurations. We use either unique names (the left two columns) or a common name (the right three).

**Longer TTL reduces authoritatives load**: We see that the traffic to authoritative servers is reduced by about 77% with the long TTL (from 127k to 43k with unique names, and from 92k to 20k with shared names). While real traffic loads may be different (higher or lower), this controlled experiment shows the economic savings when DNS is provided as a metered service [5].

**Longer TTL improves response time**: Figure 10 shows latency distributions, comparing short TTLs with long TTLs. We can see that for unique queries (Figure 10a), using a TTL of 60 s leads to a median RTT of 49.28 ms, while a TTL of 84600 s reduces the median to 9.68 ms.




**(a) Unique QNAMES**




**(b) Shared QNAMES**

**Figure 10: Distribution of client latency from Atlas VPs to controlled DNS with different TTLs.**

For shared query names (Figure 10b), the median RTT for a TTL60 s is 35.59 ms, and 7.38 ms for TTL86400, which can be explained that some VPs benefit from caches being warmed by others VPs. This controlled experiment confirms improved latency seen for Uruguay (Figure 9), since TTL86400 (the leftmost, green line) has much lower median latency than TTL60 (the rightmost cyan line).

**Longer TTL reduces latency, even more than anycast**: In addition, this controlled experiment lets us compare to an anycast service (Figure 10b). We see that *caching is far better than anycast at reducing latency*, comparing TTL86400 (the leftmost blue line) against anycast (the center orange line, median RTT =29.95 ms). While anycast helps a great deal in the tail of the distribution, caching greatly helps the median. (At 75%ile, 60 s TTLs have 106 ms latency, with anycast that drops to 67 ms, but 86,400 s TTLs reduce it to 24 ms.) This result is consistent with prior work that showed diminishing returns from very large anycast networks [13]. The cache in a recursive close to the client is often *far* faster even than an anycast site 100 km away.

## 6.3 Recommendations

From our analysis, we make the following recommendations:

| | *unique QNAME* | | *shared QNAME* | | |
|---|---|---|---|---|---|
| | **TTL60-u** | **TTL86400-u** | **TTL60-s** | **TTL86400-s** | **TTL60-s-anycast** |
| Frequency | 600s | 600s | 600s | 600s | 600s |
| Duration | 60min | 60min | 65min | 65min | 60min |
| Query | `PID.mapache-de-madrid.co` | | `1.mapache-de-madrid.co` | `2.mapache-de-madrid.co` | `4.mapache-de-madrid.co` |
| Query Type | AAAA | AAAA | AAAA | AAAA | AAAA |
| Date | 20190227 | 20190227 | 20190228 | 20190228 | 20190228 |
| | | | *Client Side* | | |
| Probes | 9095 | 9109 | 9105 | 9117 | 8869 |
| Probes (val.) | 8991 | 9009 | 8950 | 8981 | 8572 |
| Probes (disc.) | 104 | 100 | 155 | 136 | 117 |
| VPs | 15996 | 16025 | 15834 | 15910 | 15274 |
| Queries | 96438 | 96585 | 103666 | 107912 | 90553 |
| Responses | 96492 | 96645 | 103666 | 107912 | 90553 |
| Responses (val.) | 96469 | 96603 | 103640 | 107861 | 90553 |
| Responses (disc.) | 23 | 42 | 26 | 51 | 0 |
| | | | *Authoritative Server* | | |
| Querying IPs | 12967 | 10334 | 11166 | 7882 | 13773 |
| Queries | 126763 | 43220 | 92547 | 20325 | 60813(only AAAA) |
| Responses | 126763 | 43220 | 92547 | 20325 | 60813(only AAAA) |

**Table 7: TTL experiments: clients and authoritative view [41].**

**TTL Duration:** Choice of TTL depends in part on external factors (§6.1) so no *single* recommendation is appropriate for all.

For *general users, we recommend longer TTLs*: at least one hour, and ideally 4, 8, 12, or 24 hours. Assuming planned maintenance can be scheduled at least a day in advance, long TTLs have little cost.

For *TLD operators*: TLD operators that allow public registration of domains (such as most ccTLDs and `.com`, `.net`, `.org`) host, in their zone files, NS records (and glues if in-bailiwick) of their respective domains. We have seen in §3.3 that most resolvers will use TTL values provided by the child delegation, but *some* will use the parent's TTL. As such, similarly to general users, we recommend longer TTLs for NS records of their delegations (at least one hour, preferably more).

*Users of DNS-based load balancing or DDoS-prevention may require short TTLs*: TTLs may be as short as 5 minutes, although 15 minutes may provide sufficient agility for many operators. Shorter TTLs here help agility; they are are an exception to our first recommendation for longer TTLs.

**Use A/AAAA and NS records:** TTLs of A/AAAA records should be shorter or equal to the TTL for NS records for *in-bailiwick* DNS servers (§4.2). Our reasoning is they will be treated that way by many resolvers, so the configuration should reflect what will happen in practice.

For *out-of-bailiwick* servers, A and NS records are usually cached independently, so different TTLs, if desired, will be effective.

In either case, short A and AAAA records may be desired if DDoS-mitigation services are an option.

**Location of Authoritative Nameservers:** It seems good practice to operate at least one out-of-bailiwick authoritative server, since it can help provide service if the zone itself becomes unavailable (perhaps due to DDoS attack).

**Who is in control:** Given that most resolvers are child-centric, one can directly control used TTLs within the zone itself (§3).

However, one should recognize that a minority of resolvers will use TTLs from glue records stored served by a zone's parents, so operators should either configure both in-zone and glue TTLs identically, or recognize some users will use one or the other.

## 7 RELATED WORK

*DNS performance and caching efficiency:* Jung *et al.* [23], in 2002, carried out simulations based on university traces to estimate the DNS cache hit rate given TTL. They showed that longer TTLs improves caching, but TTLS shorter then 1000 s were sufficient to reap most of the benefits. In their subsequent study [22], they modeled DNS caches as a function of TTL to explain their earlier results.

Several groups evaluated DNS performance at the root. Danzig *et al.* showed that there was a significant number of misbehaving resolvers [12]. Fomenkov *et al.* examined Root DNS latency before anycast was widspread [15], and then later Liu *et al.* reexamined performance with anycast [27]. Thomas and Wessels showed how complicated caching is as seen from the Roots DNS servers [49].

More recently, Moura *et al.* [33] evaluated caching hit rates with datasets from production networks and experiments with RIPE Atlas, finding cache hit rates of around 70% for

TTLs ranging from 1800–86400 s. While this prior work measured caching and its effects, our work instead focuses on how TTLs set in different places interact to *create* an effective TTL.

*TTL and DNS resilience*: Pappas *et al.*proposed changes two strategies to improve DNS resilience to DDoS with NS-record caching [38]. They proposed refreshing TTLs in some circumstances, and renewing (pre-fetching before expiration) NS records for popular domains. This kind of pre-fetching is deployed in Unbound today [36].

Moura *et al.* investigated the relationship between TTLs in DNS and resilience in face of DDoS attacks [33]. They simulated a series of scenarios with various degrees and packet loss and showed that, together with retries, caching is a key component of DNS resilience. They showed that, to be most effective, TTLs must be longer than the DDoS attack. They recommend long TTLs where possible, but refrain from suggesting specific values.

Unlike these these two works, we focus we focus on DNS under normal operations. We examine how different records create ambiguity in the effective TTL, and make recommendations for TTL values and where they must be set.

## 8 CONCLUSION

This paper examined DNS TTLs, showing that the effective DNS TTL is often different from what is configured because TTLs appear in multiple locations and resolvers make different choices in which TTL they prefer. We use controlled experiments to demonstrate how these factors interact, and that one must control TTL in both parent and child zones. We showed that longer TTLs have important performance benefits, since caching greatly reduces latency, even more than anycast, as well as reducing traffic. Our scans of deployed DNS show that operators today have little consensus on typical TTLs, Initial discussions with selected operators suggest interest in longer TTLs, and changes at Uruguay's .uy, after our discussions, result in much lower median latency to users. We list the issues operators should consider when selecting TTLs, and suggest while those using DNS-based load-balancing or DDoS-mitigation may require short TTLs (5 or 15 minutes), others may benefit from longer TTLs (of a few hours).

## ACKNOWLEDGMENTS

We thank Paul Ebersman and Warren Kumari for their comments on drafts of this paper.

This work uses measurements from RIPE Atlas (https://atlas.ripe.net/), an open measurements platform operated by RIPE NCC.

Giovane Moura's work on this project is part of the SAND project (http://www.sand-project.nl).

John Heidemann's work is based in part on research sponsored by the U.S. Department of Homeland Security Science and Technology Directorate, Cyber Security Division (DHS S&T/CSD) via contract number HSHQDC-17-R-B0004-TTA.02-0006-I, by the Air Force Research Laboratory under agreement number FA8750-18-2-0280, by the DHS S&T/CSD via contract number 70RSAT18CB0000014. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

Wes Hardaker's work in this paper is partially supported by USC as part of B-Root research activity.

## REFERENCES

[1] J. Abley and K. Lindqvist. 2006. *Operation of Anycast Services.* RFC 4786. Internet Request For Comments. https://doi.org/10.17487/RFC4786 (also Internet BCP-126).

[2] J. Abley and K. Lindqvist. 2006. *Operation of Anycast Services.* RFC 4786. IETF. http://tools.ietf.org/rfc/rfc4786.txt

[3] Alexa. 2019. Alexa: Keyword Research, Competitive Analysis & Website Ranking. https://www.alexa.com/

[4] Amazon AWS. 2019. Amazon Route 53 | Product Details. https://aws.amazon.com/route53/.

[5] Amazon AWS. 2019. Route 53 pricing. https://aws.amazon.com/route53/pricing/.

[6] IETF DNSOP Mailing List Archive. 2016. [DNSOP] draft-fujiwara-dnsop-resolver-update-00. https://mailarchive.ietf.org/arch/msg/dnsop/1ESN4Uo8BRGrkadS-YXFR3z_cx0

[7] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. 2005. *Protocol Modifications for the DNS Security Extensions.* RFC 4035. IETF. http://tools.ietf.org/rfc/rfc4035.txt

[8] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. 2005. *Resource Records for the DNS Security Extensions.* RFC 4034. IETF. http://tools.ietf.org/rfc/rfc4034.txt

[9] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitendra Padhye. 2015. Analyzing the Performance of an Anycast CDN. In *Proceedings of the ACM Internet Measurement Conference.* ACM, Tokyo, Japan. https://doi.org/10.1145/2815675.2815717

[10] Fangfei Chen, Rmaesh K. Sitaraman, and Marcelo Torres. 2015. End-User Mapping: Next Generation Request Routing for Content Delivery. In *Proceedings of the ACM SIGCOMM Conference.* ACM, London, UK, 167–181. https://doi.org/10.1145/2785956.2787500

[11] CZ-NIC. 2016. cache prefers parent-side TTL to authoritative. https://github.com/CZ-NIC/knot-resolver/issues/34

[12] Peter B. Danzig, Katia Obraczka, and Anant Kumar. 1992. An Analysis of Wide-Area Name Server Traffic: A study of the Domain Name System. In *Proceedings of the ACM SIGCOMM Conference.* ACM, 281–292. https://doi.org/10.1145/144191.144301

[13] Ricardo de Oliveira Schmidt, John Heidemann, and Jan Harm Kuipers. 2017. Anycast Latency: How Many Sites Are Enough?. In *Passive and Active Measurements (PAM).* 188–200.

[14] R. Elz and R. Bush. 1997. *Clarifications to the DNS Specification.* RFC 2181. IETF. http://tools.ietf.org/rfc/rfc2181.txt

[15] Marina Fomenkov, k. c. claffy, Bradley Huffaker, and David Moore. 2001. Macroscopic Internet Topology and Performance Measurements From the DNS Root Name Servers. In *Proceedings of the USENIX Large Installation Systems Administration Conference.* USENIX, San Diego, CA, USA, 221–230. http://www.caida.org/publications/papers/2001/Rssac2001a/rssac_lisa.pdf

[16] K. Fujiwara. 2017. Updating Resolver Algorithm. Internet Draft. https://tools.ietf.org/html/draft-fujiwara-dnsop-resolver-update-00

[17] Shuai Hao and Haining Wang. 2017. Exploring Domain Name Based Features on the Effectiveness of DNS Caching. *SIGCOMM Comput. Commun. Rev.* 47, 1 (Jan. 2017), 36–42. https://doi.org/10.1145/3041027.3041032

[18] Wes Hardaker. 2018. Analyzing and Mitigating Privacy with the DNS Root Service. http://www.isi.edu/%7ehardaker/papers/2018-02-ndss-analyzing-root-privacy.pdf

[19] Scott Hilton. 2016. Dyn Analysis Summary Of Friday October 21 Attack. Dyn blog https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/.

[20] P. Hoffman, A. Sullivan, and K. Fujiwara. 2018. *DNS Terminology.* RFC 8499. IETF. http://tools.ietf.org/rfc/rfc8499.txt

[21] ISC BIND. 2018. Chapter 6. BIND 9 Configuration Reference. https://ftp.isc.org/www/bind/arm95/Bv9ARM.ch06.html.

[22] Jaeyeon Jung, Arthur W. Berger, and Hari Balakrishnan. 2003. Modeling TTL-based Internet Caches. In *Proceedings of the IEEE Infocom.* IEEE, San Francisco, CA, USA. http://www.ieee-infocom.org/2003/papers/11_01.PDF

[23] Jaeyeon Jung, E. Sit, H. Balakrishnan, and R. Morris. 2002. DNS performance and the effectiveness of caching. *IEEE/ACM Transactions on Networking* 10, 5 (Oct 2002), 589–603. https://doi.org/10.1109/TNET.2002.803905

[24] Brian Krebs. 2019. A Deep Dive on the Recent Widespread DNS Hijacking Attacks. Krebs-on-Security blog at https://krebsonsecurity.com/2019/02/a-deep-dive-on-the-recent-widespread-dns-hijacking-attacks/. https://krebsonsecurity.com/2019/02/a-deep-dive-on-the-recent-widespread-dns-hijacking-attacks/

[25] W. Kumari and P. Hoffman. 2015. *Decreasing Access Time to Root Servers by Running One on Loopback.* RFC 7706. IETF. http://tools.ietf.org/rfc/rfc7706.txt

[26] D. Lawrence and W. Kumari. 2018. Serving Stale Data to Improve DNS Resiliency. Internet Draft. https://tools.ietf.org/html/draft-ietf-dnsop-serve-stale-02

[27] Ziqian Liu, Bradley Huffaker, Marina Fomenkov, Nevil Brownlee, and kc claffy. 2007. Two Days in the Life of the DNS Anycast Root Servers. In *Proceedings of the Passive and Active Measurement Workshop.* Springer-Verlag, Louvain-la-neuve, Belgium, 125–134. https://www.caida.org/publications/papers/2007/dns_anycast/dns_anycast.pdf

[28] Majestic. 2019. Majestic Million. https://majestic.com/reports/majestic-million

[29] P.V. Mockapetris. 1987. *Domain names - concepts and facilities.* RFC 1034. IETF. http://tools.ietf.org/rfc/rfc1034.txt

[30] P. Mockapetris. 1987. *Domain names—concepts and facilities.* RFC 1034. Internet Request For Comments. ftp://ftp.rfc-editor.org/in-notes/rfc1034.txt

[31] P. Mockapetris. 1987. *Domain names—implementation and specficiation.* RFC 1035. Internet Request For Comments. ftp://ftp.rfc-editor.org/in-notes/rfc1035.txt

[32] P.V. Mockapetris. 1987. *Domain names - implementation and specification.* RFC 1035. IETF. http://tools.ietf.org/rfc/rfc1035.txt

[33] Giovane C. M. Moura, John Heidemann, Moritz Müller, Ricardo de O. Schmidt, and Marco Davids. 2018. When the Dike Breaks: Dissecting DNS Defenses During DDoS. In *Proceedings of the ACM Internet Measurement Conference.* https://doi.org/10.1145/3278532.3278534

[34] Moritz Müller, Giovane C. M. Moura, Ricardo de O. Schmidt, and John Heidemann. 2017. Recursives in the Wild: Engineering Authoritative DNS Servers. In *Proceedings of the ACM Internet Measurement Conference.* London, UK, 489–495. https://doi.org/10.1145/3131365.3131366

[35] Neustar. 2019. DDoS Prevention & Protection FAQs. https://www.home.neustar/resources/faqs/ddos-faqs.

[36] NLNetLabs. 2019. Unbound. https://unbound.net/.

[37] OpenDNS. 2019. Setup Guide: OpenDNS. https://www.opendns.com/setupguide/. https://www.opendns.com/setupguide

[38] V. Pappas, D. Massey, and L. Zhang. 2007. Enhancing DNS Resilience against Denial of Service Attacks. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07).* 450–459. https://doi.org/10.1109/DSN.2007.42

[39] Nicole Perlroth. 2016. Hackers Used New Weapons to Disrupt Major Websites Across U.S. *New York Times* (Oct. 22 2016), A1. http://www.nytimes.com/2016/10/22/business/internet-problems-attack.html

[40] Jing Qiao. 2017. Resolver centricity experiment. https://blog.nzrs.net.nz/resolvers-centricity-detection/.

[41] RIPE NCC. 2019. RIPE Atlas Measurement IDs. https://atlas.ripe.net/measurements/ID. ID is the experiment ID: uy-NS: 19544918, a.nic.uy-A: 19581585, google.co-NS: 19927577, mapache-de-madrid.co-NS: 19584842, in-bailiwick: 20199814, out-of-bailiwick: 20181892, TTL60-u:19862830, TTL86400-u:19863763, TTL60-s:19871393, TTL86400-s:19871498, TTL60-s-anycast:19875360, uy-NS2: 19925152.

[42] RIPE NCC Staff. 2015. RIPE Atlas: A Global Internet Measurement Network. *Internet Protocol Journal (IPJ)* 18, 3 (Sep 2015), 2–26.

[43] RIPE Network Coordination Centre. 2015. RIPE Atlas. https://atlas.ripe.net.

[44] RIPE Network Coordination Centre. 2018. RIPE Atlas - Raw data structure documentations,https://atlas.ripe.net/docs/data_struct/.

[45] Quirin Scheitle, Oliver Hohlfeld, Julien Gamba, Jonas Jelten, Torsten Zimmermann, Stephen D. Strowes, and Narseo Vallina-Rodriguez. 2018. A Long Way to the Top: Significance, Structure, and Stability of Internet Top Lists. In *Proceedings of the Internet Measurement Conference 2018 (IMC '18).* ACM, New York, NY, USA, 478–493. https://doi.org/10.1145/3278532.3278574

[46] Ricardo de O. Schmidt, John Heidemann, and Jan Harm Kuipers. 2017. Anycast Latency: How Many Sites Are Enough?. In *Proceedings of the Passive and Active Measurement Workshop.* Springer, Sydney, Australia, 188–200. http://www.isi.edu/%7ejohnh/PAPERS/Schmidt17a.html

[47] Kyle Schomp, Tom Callahan, Michael Rabinovich, and Mark Allman. 2013. On measuring the client-side DNS infrastructure. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference.* ACM, 77–90.

[48] Steve Souders. 2008. High-Performance Web Sites. *Commun. ACM* 51, 12 (Dec. 2008), 36–41. https://doi.org/10.1145/1409360.1409374

[49] Matthew Thomas and Duane Wessels. 2015. A study of caching behavior with respect to root server TTLs. DNS-OARC. https://indico.dns-oarc.net/event/24/contributions/374/

[50] Umbrella. 2019. Umbrella Popularity List. https://s3-us-west-1.amazonaws.com/umbrella-static/index.html

[51] Lan Wei and John Heidemann. 2017. Does Anycast Hang up on You?. In *IEEE International Workshop on Traffic Monitoring and Analysis (Dublin, Ireland.*