TTL Violation of DNS Resolvers in the Wild

Protick Bhowmick[1], Md. Ishtiaq Ashiq[1], Casey Deccio[2], and Taejoong Chung[1]

[1] Virginia Tech
{protick, iashiq5, tijay}@vt.edu
[2] Brigham Young University
casey@byu.edu

Abstract. The Domain Name System (DNS) provides a scalable name resolution service. It uses extensive caching to improve its resiliency and performance; every DNS record contains a time-to-live (TTL) value, which specifies how long a DNS record can be cached before being discarded. Since the TTL can play an important role in both DNS security (e.g., determining a DNSSEC-signed response's caching period) and performance (e.g., responsiveness of CDN-controlled domains), it is crucial to measure and understand how resolvers *violate* TTL.
Unfortunately, measuring how DNS resolvers manage TTL around the world remains difficult since it usually requires having the cooperation of many nodes spread across the globe. In this paper, we present a methodology that measures TTL-violating resolvers using an HTTP/S proxy service, which allows us to cover more than 27 K resolvers in 9.5 K ASes. Out of the 8,524 resolvers that we could measure through at least five different vantage points, we find that 8.74% of them extend the TTL arbitrarily, which potentially can degrade the performance of at least 38% of the popular websites that use CDNs. We also report that 44.1% of DNSSEC-validating resolvers incorrectly serve DNSSEC-signed responses from the cache even after their `RRSIG`s are expired.

1 Introduction

The Domain Name System (DNS) provides a scalable name resolution service. It uses extensive caching to improve its resiliency and performance with a time-to-live (TTL) value that specifies how long a DNS record can be cached before being discarded [22]; the TTL value is assigned by the DNS authoritative servers. DNS consumers (e.g., DNS resolvers) can cache the DNS responses during the TTL so that the future requests can be fulfilled locally without sending extra DNS queries to the DNS authoritative server.

Due to its resiliency and efficiency, DNS has evolved from simply providing a mapping between human-readable names and network-level IP addresses, to providing security features for other protocols (e.g., `MTA-STS` [19], `TLSA` [13], and `BIMI` [4] for email protocols) or better performance by delegating its control to another entity (e.g., CDN). For example, an email server can publish its

certificate information as a DNS record (i.e., `TLSA`) so that a sender can cross-check the certificate. Thus, the service operators have to manage the DNS records and their security information in a synchronous way. When they update (i.e., rollover) their credential information such as public key, they usually publish the updated DNS records in advance [13], [19] and wait at least for the TTL (or twice of TTL), *expecting that DNS resolvers clear the old cache after then.* However, it is unclear how DNS clients follow such practice; for example, a DNS resolver may cache DNS responses longer than its TTL to reduce DNS requests towards authoritative servers. This may bring a negative impact on both security and performance; for example, DNS resolvers that *extends the TTL value* may impair the performance of CDNs, which typically uses a lower TTL value to improve their resilliency and responsiveness [9].

However, it is challenging to understand how such TTL violations exist in the wild without access to devices or users in affected networks; for example, it is not straightforward to understand if a local DNS resolver in an ISP extends TTL without deploying a vantage point in the ISP. To address this challenge, there have been several successful prior approaches to measuring DNS TTL violations by using datasets collected from DNS authoritative servers [15], residential networks [7], or using active probes such as RIPE Atlas [20]. While these approaches have identified a number of resolvers that violates the TTL value in DNS records, but it is typically difficult for others to replicate and often to scale [20], [16], and mostly focus on public DNS resolvers [16].

In this paper, we explore an alternative approach to detecting DNS TTL violations of resolvers using a residential proxy service called, BrightData, which allows us to achieve measurements from over 274,570 end hosts and their 27,131 resolvers across 9,514 ASes in 220 countries. We discovered the TTL violation is prevalent; for example, we find 745 (8.74%) resolvers that extends TTLs. Furthermore, we find that another form of TTL violation that can happen to *DNSSEC-signed* records; we find that 285 DNSSEC-validating resolvers that return expired DNSSEC-signed responses when the TTL does not expire yet.

We make our analysis code and data public to the research community at

https://ttl-violation-study.github.io

2 Background and Related work

2.1 Related Work

There have been a long thread of work focusing on TTL violations in DNS resolvers, using different datasets and methodologies. Early in 2004, Pang et al. [15] used DNS logs collected from a large CDN, Akamai, to measure TTL violations and reported that 47% of clients used the expired DNS record, which indicates the prevalent violation of the TTL. Similarly, Callahan et al. [7] found 13.7% of user connections measured from residential network across 90 homes used expired DNS records in 2013.

Schomp et al. [16] took an active measurement-based approach by sending DNS queries to open resolvers from 100 PlanetLab nodes and found that 81% of open resolvers did not consistently return the correct TTL values.

Some studies [3,12] found that TTL violations are more likely to happen with small TTLs (e.g., 20 seconds); for example, Flavelet al. reported that 2% of users used stale DNS responses with 20 seconds TTL even after 15 minutes and Almeida et al. [18] found similar patterns in the traffic measured from a European mobile network operator.

Recently, RIPE labs used 9,119 unique RIPE Atlas probes reported that 4.1% of the measured resolvers increased the TTL value and 1.97% of the measured resolvers decreased the TTL value [20].

While these studies have identified several different TTL violations, it is still challenging to provide the overall TTL violation on the Internet as each of them used a different approach (e.g., passive vs. active) to focus on a different type of DNS resolvers (e.g., local vs. public resolver). Our goal is to develop an approach that achieves the same goal, but without having privileged access (e.g., CDN logs), and without having to spend significant effort to deploy software or hardware for users to install.

2.2 BrightData

In this work, we use `BrightData`, a residential proxy service, to characterize the behavior of resolvers. BrightData, formerly known as Luminati, is the paid HTTP/S proxy service that routes traffic via residential nodes (called exit nodes), who installed *Hola Unblocker* [14]. In order to route traffic, the client needs to send a HTTP request to a BrightData server, called the *super proxy*; the super proxy then forwards the request to an exit node. The exit node can perform the HTTP request and return the response back to the client via the super proxy.

BrightData offers options that can be passed with HTTP request to control exit nodes. Figure 1 shows the overview of how the BrightData platform works.

Exitnode preference: BrightData allows clients a measure of control over which exit node is chosen to forward the traffic. The client can select the country or autonomous system (AS) that the exit node is located in by adding a `-country-XX` (where `XX` is the ISO country code) or a `-asn-YY` (where `YY` is AS number) parameter to the HTTP request. The client is also allowed to choose the same exit node for subsequent requests by adding a `-session-XX` (where `XX` is a random number) to the HTTP request. Within 60 seconds, the client can choose the same exit node by using the same session number.

Exitnode persistence: The client can find the hash of the exit node's IP address in the HTTP response header, `x-luminati-ip`. By adding the `-ip-XX` option to the HTTP request (where `XX` is the hash of the IP address), the client can use the same exit node if available. This option is extremely useful to measure the TTL violation behavior of the resolvers; we can still measure the same resolvers

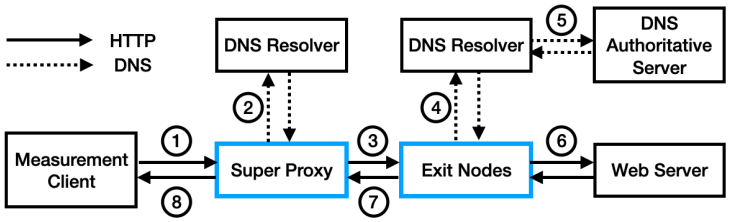

Fig. 1: Timeline of a request in BrightData: the client sends an HTTP request to the super proxy ①; the super proxy makes a DNS request for the sanity check and forward the request to an exit node ②∼③; the exit node uses its DNS resolver and fetch the HTTP response ④∼⑥ and forward it back to the super proxy ⑦, which return it to the client ⑧. Brightdata controls the Super Proxy and exit nodes (shown with blue boxes).

(used by the same exit node) by finding the same exit node after a TTL with a longer period of time (e.g., 60 minutes) expires.

DNS request location: By default, DNS resolution is done and cached at the super proxy's end; however, the client can specify the `dns-remote` option to the HTTP request to make DNS resolution done by the the exit node (using the exit node's DNS server). In out experiment, we do so as we want the resolution to be done at the client's end.

With these options, we use BrightData to let exit nodes send HTTP requests to our domains; the exit nodes will also send DNS requests to our DNS authoritative server through their resolvers, which gives an opportunity to understand their behavior. In the following section, we introduce our experiment methodology and its challenges.

3 TTL Extension in the wild

In this section, we describe how we use BrightData to understand how DNS resolvers *extend* TTL in the DNS responses.

3.1 Methodology

At first glance, measuring and identifying resolvers that extend the TTL seems straightforward: we pick one exit node and request it fetch the domain that resolves IP_1. After its TTL expires, we update its `A` record to IP_2 and let the same exit node fetch the same domain to see if they connect to IP_1. However, in practice it more difficult, because the client may use multiple DNS resolvers that have many upstream resolvers, thus it may receive multiple DNS responses; this behavior is common mainly to improve the performance; modern public DNS servers usually have multiple caches with complex caching hierarchy [28], [1].

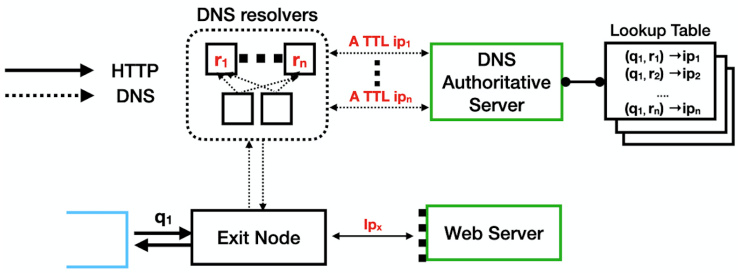

Fig. 2: Our methodology that extends the BrightData platform in Figure 1. We control the DNS authoritative and web server (shown with green boxes); for the same `qname`, our DNS authoritative server now returns a different `A` record to each different resolver so that we can infer which DNS response the exit node used by monitoring the incoming IP address of HTTP request.

However, we are not allowed to see which DNS response the exit node actually used, making it hard for us to identify who has extended the TTL. To address these issues, we return *a different `A` record to each different resolver* so that we can identify which DNS resolver's response the exit node has used by monitoring the incoming IP address of the HTTP request for a certain domain. More specifically, we proceed our experiments as follows as illustrated in Figure 2.

(a) As the first phase (P_1), we first let an exit node fetch a unique subdomain, `http://<<UID>>.m.com`. We extract the `x-luminati-ip` value from the HTTP response header so that we can choose the same exit node after the TTL expires.

(b) At our authoritative nameserver, for each resolver that looks up the same `qname`, we pick an IP address that has never been used for the `qname` and dynamically generate an `A` to serve the request. Then, we create an entry that maps a tuple of `qname` and the resolver's IP address to the served IP address and insert it to the mapping table. If we observe more DNS resolvers for the same `qname` than N, we discard the exit node from further analysis.

(c) From the webserver, we examine the destination of the IP address of the HTTP request to find the matched DNS resolver's IP address in the mapping table. This allows us to find the DNS resolver that the exit node used.

(d) Then, we immediately *retract* all DNS entries from the DNS authoritative name server to ignore all subsequent DNS requests, and we wait for TTL to let the cached DNS responses expire.

(e) Once TTL expires, we set our authoritative name server to serve `A` that points to the IP address (IP_{new}) that has never been assigned to any DNS resolver. We then use `-ip-XX` option in the HTTP request to choose the same node and let it fetch `http://<<UID>>.m.com` again. We call this step the second phase.

In our experiment, we use 8 (N) different IP addresses based on the observation where 99.9% of HTTP requests incur less than 9 DNS requests.

Ethical Consideration: First of all, we adhere to the Terms of Service of Bright-Data; our experiment followed their terms and condition and only used the commercial services provided by Brightdata network. The peers are always explicitly asked with a clear consent screen to opt-in the proxy network. Additionally, we do not collect any PII of the exit nodes' users and the exit nodes agreed to allow Brightdata to route traffic through themselves in exchange for their free VPN services. Our experiments only involve generating HTTP and DNS queries to the DNS authoritative servers and HTTP servers that we control. Moreover, we do not send any other queries to other domains that we do not control. Thus, we believe that the experiments do not introduce any harm to the proxy service or the exit nodes.

3.2 Results

During our measurement period, we are able to send 2,068,686 unique HTTP queries served by 274,570 unique exit nodes and their 27,131 resolvers [3] in 9,514 ASes across 220 countries.

We also run our experiment with five different TTLs (1, 5, 15, 30, and 60 minutes) to investigate how TTL values impact on a DNS resolver's caching behavior.

Identifying potential TTL-extending resolvers is straightforward; when we observe an exit node that still connects to the webserver with the old IP address, we can find it by looking up the mapping table and label it as a potential TTL-extending resolver. However, when we observe an exit node that uses the new IP address (IP_{new}), we cannot simply mark all resolvers in the second phase as TTL-honoring resolvers because some resolvers only show up in the first phase. Thus, we mark resolvers as the potential TTL honoring resolvers only when they appear in both the first and second phase.

Since we use exit nodes as a proxy to understand DNS resolvers behavior, we cannot blindly use the results to characterize the DNS resolvers; for example, a stub resolvers on the exit node may extend the TTL making their resolvers look like TTL-extending ones. Thus, we focus on the resolvers where we have at least 5 exit nodes, this sample size allows us to draw strong inferences to characterize their behaviors; this leaves us 9,031 resolvers, and their 234,605 exit nodes across the different TTL setups. Then, for each resolver, we calculate the fraction of the exit nodes that connect to the IP_{old}; Figure 3 shows the results and we make a number of observations. First, we find that there is a clear separation between TTL-extending resolvers and TTL-honoring resolvers. For example, when we set our TTL values to 60 minutes, 4,147 (92.5%) resolvers perfectly honor the TTL while 14 (0.31%) resolvers extend TTL; the others (7.1%) show mixed

[3]Since we are only permitted to observe the only egress resolver IPs querying our authoritative servers, we label each querying IP as a resolver.

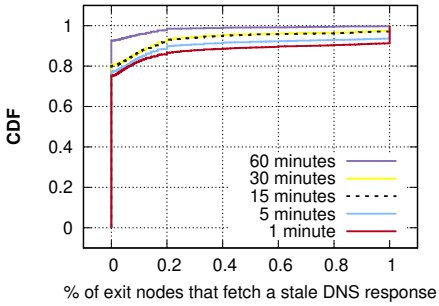

Fig. 3: CDF of the fraction of the exit nodes that use the the stale response for each resolver.

		Ours	
		Honor.	**Ext.**
Direct Scan	**Honor.**	197	0
	Ext.	0	16
Proxy Rack	**Honor.**	381	1
	Ext.	0	62

Table 1: Validation results with direct probing and ProxyRack

behaviors, which could be due to the stub or other frontend resolvers that we could not measure. Second, we find that *t*he number of TTL extending resolvers constantly grows as we decrease the TTL value; for example, the percentage of TTL extending resolvers increases from 14 (0.31%) to 129 (2.53%), 161 (2.87%), 414 (6.53%), and 745 (8.74%) as we decrease the TTL value from 60, 30, 15, 5 and 1 minute. Surprisingly, we also find that the set of TTL extending resolvers that we measured with TTL_x is *always* a subset of what we measure with TTL_y if TTL_x is less than TTL_y. This strongly suggests that some resolvers use a default minimum TTL value; this could be due to reduce their resolution load; for example, popular DNS software such as PowerDNS [26], KnotDNS [17] and Unbound [29] has an option for this.

3.3 Cross-validation

We now attempt to cross validate our methodology by focusing on the resolvers that *always* honor (or extend) the TTL with 1 minute, which leaves us 7,160 resolvers. For each resolver, we first attempt to directly send DNS queries to test whether they respond, and if so, if they extend the TTL by looking up our domain twice with a time gap of TTL. Since this is likely to allow us to measure public resolvers, we also leverage another residential proxy service, ProxyRack [27] to cover local resolvers as well; the coverage is limited (less than 2+ million residential proxies), but it permits to send an arbitrary UDP traffic so that we can send a DNS request to its local resolver. For each of the rest of the resolvers, we attempt to find exit nodes that share the same AS with the resolver and send DNS requests. Table 1 shows the result; surprisingly, we find that *all 212 resolvers that we could measure show the consistent behaviors our observation.* From the ProxyRack experiment, all 443 resolvers except one show the consistent behavior; we found one resolver that our methodology and a ProxyRack experiment disagree, but could not find the cause.

Interestingly, when we consider the number of TTL extending resolvers, we see more TTL-violating resolvers in ProxyRack experiments than the direct scan-

Rank	Country	Exit nodes w/ TTL-extended	Total	Ratio
1	Togo	91	106	85.84%
2	China	1,514	2,425	62.43%
3	Réunion	112	189	59.26%
4	Jamaica	175	481	36.38%
5	Sint Maarten	137	455	30.12%
6	France	81	329	24.62%
7	Ivory Coast	68	288	23.61%
8	Cayman Islands	105	461	22.77%
9	Ireland	347	1,726	20.1%
10	Switzerland	141	704	20.02%
11	Spain	489	2,603	18.79%
12	Myanmar	136	762	17.85%
13	Germany	36	226	15.93%
14	Finland	300	1,912	15.69%
15	Russia	8,808	57,283	15.38%

Table 2: Top fifteen countries sorted by the fraction of exit nodes that use TTL-extending resolvers

ning (i.e., 14.0% vs. 7.5%). Since direct scanning only allows us to measure the public resolvers, it may indicate that the local resolvers are more likely to extend TTLs; we will explore this in the following section. In summary, we confirm that our methodology can accurately find the TTL-extension policy of DNS resolvers.

3.4 Macroscopic Analysis

To obtain a macroscopic view of TTL extension phenomena, we first map ASes to ISPs (as one ISP may operate many ASes) and countries using CAIDA's AS-organizations dataset [8]. Next, we group exit nodes according to country and AS, and focus on the groups where we have at five exit nodes. We first notice that the exit nodes that fetch expired DNS responses are widely spread across the globe; Table 2 shows the top 15 countries sorted by the fraction of exit nodes that use TTL-extending resolvers. For example, we found that in Togo, more than 85% of exit nodes we measured experienced TTL extension.

Now, we focus on individual DNS resolvers; as we have observed from the validation results, local resolvers tend to have more TTL-extending resolvers than that of the public ones. Now, we try to find local resolvers by grouping exit nodes by the DNS resolver.

Again, to minimize potential client side impact, we focus on those where we observe at least 5 exit nodes using the DNS resolver.

We then identify ISP-provided DNS servers as ones where all exit nodes and the DNS server belong to the same ISP. With this method, we have identified 6,871 ISP-provided DNS resolvers in our measurement. Table 3 shows the top 15 local resolvers, *all of which exit nodes always receive the old, TTL-expired, responses*. The majority of these ISPs and DNS resolvers are in Russia and China;

Country	ISP	DNS Servers	Exit Nodes
Russia	PSJC Vimpelcom	16	366
	PSJC Rostelecom	12	124
	Net By Net	8	58
	TIS Dialog	6	108
	MTS PSJC	4	69
	MSK-IX	4	36
China	China Telecom	13	125
	China Mobile	7	39
	Tianjin Provincial	5	50
	China Unicom	4	27
South Africa	MTN SA	6	49
	Neology	5	97
Cayman Islands	Cable & Wireless	7	88
Hong Kong	HGC Global Communications	4	38
Trinidad and Tobago	Columbus Comm.	6	115
Turkey	Netonline Billisim.	5	84

Table 3: Table showing the top 15 local resolvers that extend TTLs.

for example, we measure 13 local resolvers in China Telecom, *all of which extend TTL*; this strongly suggests that the TTL extension is imposed by the ISP.

3.5 Impact of TTL extension: Case Study of CDNs

It is known that CDN typically uses short TTLs for performance (e.g., load-balancing) or security reasons [23,11]; for example, if a PoP (Point of Presence) experiences outages, a short TTL can help them rapidly direct traffic to a different one. Thus, TTL-extending resolvers may hurt their responsiveness; for example, Moura et al. [21] found that A records have relatively shorter TTLs than other record types due to dynamic changes of server addresses in clouds and CDNs. We now focus on Tranco 1M domains [25] and try to identify domains that use CDNs. We note that most CDNs use DNS-based redirection scheme such as Akamai [23] to redirect users to CDN infrastructures by using Canonical Name (`CNAME`) records; for example, when a user requests a domain, `www.reddit.com`, it will redirect to another domain controlled by `Fastly`, so that they can handle the request as shown below.

```
$ dig www.reddit.com
...
;; ANSWER SECTION:
www.reddit.com.          3600       IN  CNAME   reddit.map.fastly.net.
reddit.map.fastly.net    60         IN  A       151.101.1.140
```

We use a OpenINTEL dataset [24] that collects `A` records of Tranco Top 1M domains including full `CNAME` expansion. For each domain, we focus on whether a `CNAME` record exists in its lookup and whether the `CNAME` record was used to direct traffic to popular CDNs; in order to do so, we manually compiled the list of `CNAME` patterns for popular CDNs (e.g., `e[1-9]*.a.akamaiedge.net` for

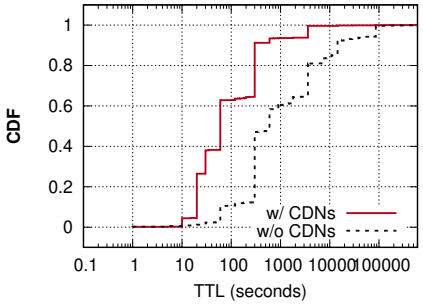

Fig. 4: CDF of TTLs in the Tranco top 1 million domains

CDN	TTL	Domains
Akamai	20	12,247 (99.9 %)
CloudFlare	300	10,736 (98.7 %)
Cloudfront	60	9,642 (99.8 %)
Fastly	30	6,237 (98.6 %)
Google	300	2,759 (98.8 %)
Azure	10	2,536 (47.0 %)
Netlify	20	1,531 (98.2 %)
XCDN	20	99 (47.8 %)
Alibaba	120	91 (58.7 %)
CDN77	15	68 (91.8 %)

Table 4: Top 10 CDNs and their TTL of expanded `A` records used by most of their domains

`Akamai`), which contains 38 CDNs in total.[4] Figure 4 shows the CDF of the TTL of the `A` records after their `CNAME`s are expanded. We immediately notice that the TTL of `A` records from CDN is much shorter than the rest of domains; for example, 38% of TTLs from CDNs is less than 60 seconds. Considering that we have found that 8.74% of resolvers extend the TTL when it is less than or equal to 60 seconds (§3.2), this indicates that these resolvers will extend the TTLs for more than 38% of CDN-managed websites, which potentially hurt their responsiveness. For example, we find that Akamai sets the TTL to 20 seconds for 99.9% of their domains; Table 4 shows the TTL values for the top 10 CDNs in terms of the number of domains they serve and we can find that most of them use very short TTLs (e.g., 10 seconds for Azure).

4 TTL Violation in DNSSEC

When it comes to DNSSEC, the TTL in a DNS response is not the only attribute that determines the caching period; a DNSSEC-signed response can come with its signature, which is called `RRSIG` records. `RRSIG` records also carry `inception` and `expiration dates` that limits its validity, thus DNSSEC-supporting resolvers *must evict DNS responses of which* `RRSIG`*s are expired from the cache, even if their TTL is not expired yet* [2]. Now, we also examine whether DNSSEC-supporting resolvers in the wild correctly honor TTL values for signed DNS records by expanding our methodology.

Experiment Settings: We follow the similar methodology as presented in §3.1. Additionally, we make our domain name fully signed (e.g., uploading a `DS` record to the parent zone) and provide DNS responses of which signature expires

[4]Our methodology can miss domains that delegate its name server to CDNs by replacing their `NS` records with CDN's ones. We could potentially identify them by checking whether both of their web server and DNS server are managed by the same CDN. However, some companies (e.g., Alibaba and Google) also provide VPS hosting service, which will cause false-positive (e.g., the domain owner manages both servers within the same VPS), thus we only focus on the `CNAME` expansion information.

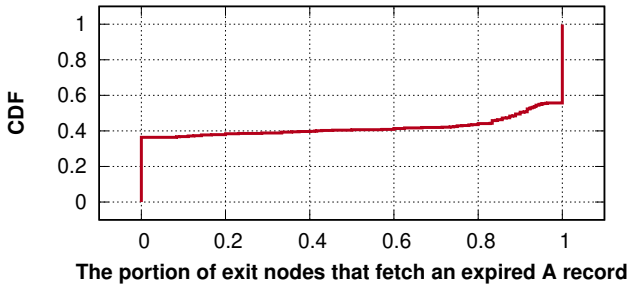

Fig. 5: 44.1% of resolvers serve expired (thus invalid) DNS responses.

earlier than its TTL. More specifically, we set our TTL value to 60 minutes for `A` records and other DNSSEC-related records, and their corresponding `RRSIG`s to be invalidated in 30 minutes. After sending the first request, we make the second request after the `expiration date`, *but within TTL* to see whether the resolver fetches a fresh `A` from the authoritative server.

Results: We run our measurement from October 27th, 2022 to Nov 1st, 2022 and obtain 91,634 exit nodes and 13,679 DNS resolvers. For the rest of this section, we now focus on DNS resolvers where we observed at least 5 exit nodes to minimize the potential client-side impacts, which leaves us 5,274 (38.5%) resolvers with 75,684 (82.6%) exit nodes.

DNSSEC-validating resolvers: DNSSEC-validating resolvers must specify `DO` ("DNSSEC OK") bit in the EDNS prseudorecord so that the DNS authoritative servers can provide the `RRSIG`s and other DNSSEC-related records. In our measurement, we find 4,917 (93.2%) resolvers covering 94% (71,242) of exit nodes enabled `DO`, which indicates that the majority of DNS resolvers *seem* to support DNSSEC. However, not all DNS resolvers with `DO` correctly support DNSSEC. For example, a study [6] found that 82% do not validate the response even though they have requested and received `RRSIG` records. To consider the only DNSSEC-validating resolvers, we make exit nodes to send another HTTP requests, which is incorrectly signed (i.e., the `RRSIG` of `A` record is cryptographically invalid); thus, if an exit node can only fetch the correct record, it indicates that its DNS resolver actually performs DNSSEC validation. With the additional step, we find 646 (13.1)% resolvers covering 6,001 (8.4%) exit nodes perform validation.

DNSSEC-validating resolvers with TTL violation: We now calculate the percentage of exit nodes that fetch an expired DNS record for each resolver. As shown in Figure 5, we find that 520 (80.4%) resolvers show consistent behavior among the exit nodes; 235 (36.3%) resolvers (with 1,505 exit nodes) have fetched the DNS response again from our authoritative server, which indicates that the resolvers evicted the DNS responses with expired `RRSIG`s. However, 285 (44.1% of considered) resolvers (with 2,645 exit nodes) have served the second client request from its cache without making the second request to the authoritative server, which is a direct violation of the DNSSEC standard [2].

5 Concluding Discussion

In this paper, we have leveraged a residential proxy network, BrightData, to measure TTL violations in resolvers. BrightData manages millions of exit nodes, which potentially opens an opportunity for researchers to understand DNS resolvers in the wild. However, since we are not permitted to directly send DNS requests to DNS resolvers, we developed a methodology to pinpoint which DNS response an exit node uses and which DNS resolver disregard TTL; we identified 745 resolvers that extend TTL values and 285 DNSSEC-validating resolvers that do not consider validity period in cache. Before concluding the paper, we wish to discuss our limitation and measuring resolvers that shorten the TTL.

Limitation: If an exit node uses a DNS resolver that leverages a multi-layer distributed caching infrastructure like Cloudflare [28], our methodology can only measure the backend caching DNS resolvers because we can only monitor the incoming DNS requests to the authoritative server. This makes it hard for us to determine where the TTL violation exactly happens; it could be due to the frontend caches, stub resolvers, or middleboxes. Thus, we have only focused on the resolvers that we are able to measure at least from five exit nodes that show consistent behavior, which provides more confidence on our inference, but costs us to lose the number of resolvers that we could analyze.

TTL shortening in the wild: A DNS resolver may cache the DNS response shorter than the TTL; unlike TTL extension, however, caching DNS records shorter than the TTL is not any violation of the DNS standard since RFC 2181 [10]. We can use a similar methodology to detect resolvers that cache DNS records shorter than the TTL set by the authoritative server; for example, some resolvers may have a parameter that determines the maximum TTL mainly not to trust very large TTL values for security purpose [29] [5]. However, resolvers can also decide to evict the cached DNS response depending on its cache size and eviction policy, which makes it a bit hard to consistently capture DNS resolvers that always cache shorter than the TTL. By making the second request earlier than the TTL, we are able to measure 49 (0.99%, out of 4,965) resolvers that *always* shorten the TTL and 4653 (93.7%) resolvers that always preserve the original TTL, but we also find 263 (5.3%) resolvers showing mixed behaviors, which suggests that their eviction policy might have impacted on, and eventually, makes it hard for us to further investigate.

Acknowledgments

We thank the anonymous reviewers and our shepherd, Paul Schmitt, for their helpful comments. We also thank BrightData for their credits to use the service. This research was supported in part by NSF grants CNS-2053363 and CNS-2051166, and 4-VA, a collaborative partnership for advancing the Commonwealth of Virginia.

References

1. K. Amit, S. Haya, and W. Michael. Counting in the Dark: DNS Caches Discovery and Enumeration in the Internet. *DSN*, IEEE Computer Society, 2017.
2. R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. RFC 4033, IETF, 2005. `http://www.ietf.org/rfc/rfc4033.txt`.
3. H. A. Alzoubi, M. I. Rabinovich, and O. S. The anatomy of LDNS clusters: findings and implications for web content delivery. *WWW*, 2013.
4. S. Blank, P. Goldsten, T. Loder, T. Zinkn, and M. Bradshaw. Brand Indicators for Message Identification (BIMI). IETF, 2021.
5. BIND max-cache-ttl. `https://bind9.readthedocs.io/en/v9_18_7/reference.html?highlight=max-cache-ttl`.
6. T. Chung, R. van Rijswijk-Deij, B. Chandrasekaran, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson. A Longitudinal, End-to-End View of the DNSSEC Ecosystem. *USENIX Security*, 2017.
7. T. Callahan, M. Allman, and M. R. On Modern DNS Behavior and Properties. *CCR*, 43(4), 2013.
8. CAIDA ASOrganizations Dataset. `http://www.caida.org/data/as-organizations/`.
9. DNS based load-balancing. `https://www.cloudflare.com/learning/performance/what-is-dns-load-balancing/`.
10. R. Elz and R. Bush. Clarifications to the DNS Specification. RFC 2181, IETF, 1997.
11. Edge and Browser Cache TTL. `https://developers.cloudflare.com/cache/about/edge-browser-cache-ttl/`.
12. A. Flavel, P. Mani, and D. A. Maltz. Re-evaluating the responsiveness of DNS-based network control. *LANMAN*, 2014.
13. P. Hoffman and J. Schlyter. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. RFC 6698, IETF, 2012.
14. Hola VPN. `http://hola.org/`.
15. P. Jeffrey, A. Aditya, S. Anees, K. Balachander, and S. Srinivasan. On the Responsiveness of DNS-Based Network Control. *IMC*, 2004.
16. S. Kyle, C. Tom, R. Michael, and A. Mark. On Measuring the Client-Side DNS Infrastructure. *IMC*, 2013.
17. cache-min-ttl in KnotDNS. `https://knot-resolver.readthedocs.io/en/stable/daemon-bindings-cache.html`.
18. A. Mario, F. Alessandro, P. Diego, V.-R. Narseo, and V. Matteo. Dissecting DNS Stakeholders in Mobile Networks. *CoNEXT*, 2017.
19. D. Margolis, M. Risher, B. Ramakrishnan, A. Brotman, and a. J. Jones. SMTP MTA Strict Transport Security (MTA-STS). RFC 8461, IETF, 2018.
20. G. Moura. DNS TTL Violations in the Wild - Measured with RIPE Atlas. `https://labs.ripe.net/author/giovane_moura/dns-ttl-violations-in-the-wild-measured-with-ripe-atlas`.
21. G. Moura, J. H. a. R. de O. Schmidt, and W. Hardaker. Cache Me If You Can: Effects of DNS Time-to-Live. *IMC*, 2019.
22. P. Mockapetris. Domain Names - Concepts and Facilities. RFC 1034, IETF, 1987.
23. E. Nygren, R. K. Sitaraman, and J. Sun. The Akamai network: a platform for high-performance internet applications. *OSR*, 44(3), 2010.
24. OpenINTEL. `https://www.openintel.nl/`.

25. V. L. Pochat, T. V. Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen. TRANCO: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. *NDSS*, 2019.
26. minimum-ttl-override option in PowerDNS. `https://doc.powerdns.com/recursor/settings.html#minimum-ttl-override`.
27. ProxyRack. `https://www.proxyrack.com`.
28. A. Randall, E. Liu, G. Akiwate, R. Padmanabhan, G. M. Voelker, S. Savage, and A. Schulman. Trufflehunter: Cache Snooping Rare Domains at Large Public DNS Resolvers. *IMC*, 2020.
29. Cache-min-ttl, Cache-max-ttl option in Unbound. `https://nlnetlabs.nl/documentation/unbound/unbound.conf/`.