Measuring and Understanding TTL Violations in DNS Resolvers

Protick Bhowmick

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science and Application

December 13, 2023

Blacksburg, Virginia

Keywords: DNS, Network Measurement, Web Security Protocol

Measuring and Understanding TTL Violations in DNS Resolvers

Protick Bhowmick

(ABSTRACT)

The Domain Name System (DNS) is a scalable-distributed caching architecture where each DNS records are cached around several DNS servers distributed globally. DNS records include a time-to-live (TTL) value that dictates how long the record can be stored before it's evicted from the cache. TTL holds significant importance in aspects of DNS security, such as determining the caching period for DNSSEC-signed responses, as well as performance, like the responsiveness of CDN-managed domains. On a high level, TTL is crucial for ensuring efficient caching, load distribution, and network security in Domain Name System. Setting appropriate TTL values is a key aspect of DNS administration to ensure the reliable and efficient functioning of the Domain Name System. Therefore, it is crucial to measure how TTL violations occur in resolvers. But, assessing how DNS resolvers worldwide handle TTL is not easy and typically requires access to multiple nodes distributed globally. In this work, we introduce a novel methodology for measuring TTL violations in DNS resolvers leveraging a residential proxy service called Brightdata, enabling us to evaluate more than 27,000 resolvers across 9,500 Autonomous Systems (ASes). We found that 8.74% arbitrarily extends TTL among 8,524 resolvers that had atleast five distinct exit nodes. Additionally, we also find that the DNSSEC standard is being disregarded by 44.1% of DNSSEC-validating resolvers, as they continue to provide DNSSEC-signed responses even after the RRSIGs have expired.

Measuring and Understanding TTL Violations in DNS Resolvers

Protick Bhowmick

(GENERAL AUDIENCE ABSTRACT)

The Domain Name System (DNS) works as a global phonebook for the internet, helping your computer find websites by translating human-readable names into numerical IP addresses. This system uses a smart caching system spread across various servers worldwide to store DNS records. Each record comes with a time-to-live (TTL) value, essentially a timer that decides how long the information should stay in the cache before being replaced. TTL is crucial for both security and performance in the DNS world. It plays a role in securing responses and determines the responsiveness of load balancing schemes employed at Content Delivery Networks (CDNs). In simple terms, TTL ensures efficient caching, even network load, and overall security in the Domain Name System. For DNS to work smoothly, it's important to set the right TTL values and the resolvers to strictly honor the TTL. However, figuring out how well DNS servers follow these rules globally is challenging. In this study, we introduce a new way to measure TTL violations in DNS servers using a proxy service called Brightdata. This allows us to check over 27,000 servers across 9,500 networks. Our findings reveal that 8.74% of these servers extend TTL arbitrarily. Additionally, we discovered that 44.1% of servers that should be following a security standard (DNSSEC) are not doing so properly, providing signed responses even after they are supposed to expire. This research sheds light on how DNS servers around the world extend TTL and the potential performance and security risks involved.

This thesis is based on our previously published work in the *International Conference on Passive and Active Network Measurement, 2023* [6]. I have permission from my co-authors and publisher to use the work for my thesis.

Acknowledgments

I am incredibly indebted to my advisor Dr. Tijay Chung for his unwavering support and guidance throughout this research. His insightful feedback, expertise and constant encouragement were instrumental in shaping this thesis. I am very grateful to my committee Members Dr. Bimal Viswanath and Dr. Thang Hoang for their guidance and support. I would like to thank my colleagues at NetSecLab Mohammad Ishtiaq Ashiq Khan and Weitong Li for their discussions and feedbacks that were beneficial to my research. I also thank BrightData for letting me use their platform to carry out the reserach. Finally, I am thankful to my parents for their unwavering encouragement in my academic journey.

Contents

List of Figures

List of Tables

Glossary of Research terminology.

DNS (Domain Name System) is a distributed system that translates human-readable domain names into IP addresses, facilitating internet communication.

DNSSEC (Domain Name System Security Extensions) is a security extension to DNS that provides authentication and data integrity for DNS records.

TTL (Time to live) is numerical value in seconds that controls how long DNS records are kept in cache.

CDN (Content Delivery Network) is a network of servers around the world that deliver web content to users with high availability and performance.

UDP (User Datagram Protocol) is a connectionless protocol that sends datagrams without verifying delivery.

TCP (Transmission Control Protocol) is a connection-oriented protocol guaranteeing dependable delivery of data..

HTTP (Hypertext Transfer Protocol) is a protocol used for transmitting and retrieving web content over the internet.

Chapter 1

Introduction

The Domain Name System (DNS) is a vital component of the Internet that provides a scalable and efficient name resolution service. DNS has an inherent caching architecture to improve its performance and resiliency. It has a time-to-live (TTL) value that specifies how long a DNS record can be cached before being evicted. DNS consumers, such as DNS resolvers, store the DNS responses in their cache during this TTL period. This saves both time and bandwidth without the need to fetch the records from authoritative servers every time a client queries the resolver.

Initially, the purpose of DNS was to hold two fundamental infrastructure components, names and their corresponding addresses. Now, it now also has multiple other functionalities, for example, it provides security features for other protocols [36, 25]. This requires service operators to manage their DNS records in a synchronized manner. Now, DNS resolvers should honor TTL for maintaining the integrity and performance of the Domain Name System, benefiting both end-users and the overall DNS infrastructure. But, a DNS resolver may cache DNS responses longer than their TTL to reduce the number of DNS requests it sends to authoritative servers. Such practice can adversely affect both security and performance, compromising the robustness of the DNS architecture. For example, CDNs often use lower TTLs for better responsiveness [27]. DNS resolvers that extend TTL values will impair the performance of CDNs hurting their resiliency and responsiveness. Now, measuring the prevalence of TTL violations can be quite challenging. To accurately measure the extent and impact of TTL violations, obtaining access to devices or end-users within affected networks may be necessary. Several prior approaches have successfully measured DNS TTL violations [37, 40, 12]. However, these approaches are usually hard to conduct; moreover, these approaches can mainly measure public resolvers. In our work, we rather take an unconventional method to identify DNS TTL violations in resolvers, utilizing a residential proxy service named BrightData. This approach enables us to carry out measurement from a broad spectrum, encompassing more than 274,570 end hosts and their 27,131 associated resolvers spanning 9,514 Autonomous Systems (ASes) across 220 countries.

We make the following contributions. *First*, we propose a novel approach to detect DNS TTL violations using a residential proxy service. *Second*, we discover that TTL violation is quite prevalent, with 745 (8.74%) resolvers extending TTLs. *Third*, we also find another form of TTL violation that can occur to DNSSEC-signed records: where resolvers cache records beyond the DNSSEC expiry as long as the TTL had not expired, violating DNSSEC standards.

Chapter 2

Background

This section provides an overview of the concepts and protocols relevant to this work.

2.1 DNS

The Domain Name System (DNS) is a scalable system that translates domain names to IP addresses, which are used to send traffic between devices on the Internet. DNS resolvers cache DNS records for a period of time specified by the time-to-live (TTL) value in the record. This allows resolvers to respond to future DNS queries locally without having to query authoritative DNS servers. Also, DNS plays an important role in both the security and performance of the Internet. In the following subsections, we go through several key components and concepts of the Domain Name System (DNS).

2.1.1 Domain Name

A domain name is a human-readable label that represents a specific website or network resource. It provides a user-friendly way to identify and access resources on the Internet. For example, "google.com" is a domain name that represents the website of the Google search engine. Domain names are organized in a hierarchical structure, with the top-level domains (TLDs) representing the highest level of the hierarchy (e.g., .com, .org, .net). Beneath the TLDs are the second-level domains (SLDs). For example, in the domain "example.com" "example" is the second-level domain. Subdomains are further subdivisions of domain names. They are created by adding additional labels to the left of the domain. For example, "sublevel.example.com" is a subdomain of "example.com".

2.1.2 IP Address

An IP address is a numerical label assigned to each device connected to a computer network. It serves as a unique identifier for devices. This allows them to recognize and connect with other devices over the internet. IP addresses are divided into two types: IPv4 (Internet Protocol version 4) and IPv6 (Internet Protocol version 6). The Domain Name System (DNS) acts as a bridge between domain names and IP addresses.

2.1.3 DNS TTL (Time-to-live)

A resolver may store the DNS records in its cache up to a certain period and DNS TTL specifies that. After the TTL expires, the resolver must evict the record and query a name server again to get the updated record. This ensures that DNS records are kept consistent with the authoritative server and that resolvers are not using stale information.

2.1.4 Name Server

Name servers are integral components of the Domain Name System (DNS). They are responsible for storing DNS records. These records contain different information about the domain, for example: the IP address associated with it.

Name servers are organized in a hierarchical structure. The root name servers are at the top, followed by top-level domain (TLD) servers, and finally authoritative name servers for specific domains. When a DNS query is made, the name server hierarchy is traversed from top to bottom to find the authoritative name server that holds the requested information. In DNS, an authoritative name server is a server that holds the definitive records for a specific domain or zone. It is responsible for providing accurate and up-to-date information about the domain, including its IP address, mail servers, and other related information.

2.1.5 DNS Resolver

Resolvers, on the other hand, receive DNS queries from client applications and are responsible for initiating the resolution process. They fetch DNS records from the name servers to retrieve the IP address associated with the requested domain name. Resolvers cache the resolved IP addresses for the TTL period to improve performance and reduce the load on the DNS infrastructure.

DNS resolvers can be broadly classified into two different types: local and public resolvers.

Local DNS resolvers are usually hosted by the ISP. They are located on the ISP's network, and they are typically faster than public DNS resolvers as they are closer to the clients. Local DNS resolvers are only accessible from the same network. On the other hand, Public DNS resolvers are usually provided by large organizations, such as Google, Cloudflare, etc. They are located on the public Internet and hence accessible to all users.

2.1.6 Name Resolution

Domain name resolution is the process by which domain names are translated into corresponding IP addresses. Now we go over the steps that are associated with the name resolution process as shown in Figure 2.1.

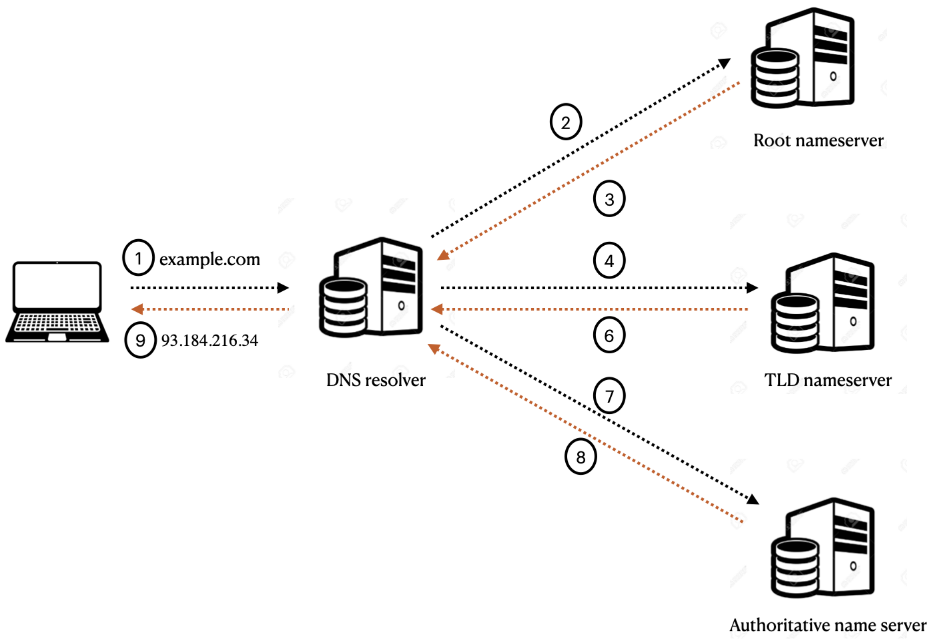

Figure 2.1: DNS resolution process: the figure shows the components and their interaction in a typical DNS name resolution

- The name resolution starts when a Client wants to connect to a domain (example.com).
- The user's device sends a DNS query to the recursive DNS resolver. This resolver can be a local resolver (typically provided by the ISP) or a public resolver (for example: Google, or Cloudflare).
- The resolver first checks if their record is present in its own cache. If the record is present there it immediately returns the response. if not, it starts the name resolution by contacting the root name server.

- The root name server will return the resolver a referral to the TLD name server that will be responsible for the top-level domain of the requested domain name (".com" TLD name server)
- The TLD name server will return an NS record that provides a referral to the authoritative name server for "example.com".
- Now, the resolver queries the authoritative name server. It has the IP address of example.com in its zone file and responds to the request with that record.
- The resolver caches the response in its cache for the TTL period and returns the repose to the client.
- The client now can establish a connection to the web server associated with example.com. It is notable that the client can also cache the records in its local cache so that it can speed up subsequent connections to the same domain.

We can observe that the Domain Name System (DNS) is a distributed caching system where multiple DNS servers work together to resolve domain names to IP addresses. Caching at multiple levels helps to reduce the traffic on DNS servers, resolvers and improve the efficiency of the DNS resolution process.

2.2 DNS-based load balancing and CDNs

DNS-based load balancing is a method for distributing incoming network traffic across different servers to optimize resource utilization and improve performance [27]. This approach leverages the DNS system to direct clients to different server IP addresses based on various load-balancing algorithms.

DNS-based load balancing can be implemented using round-robin DNS, redirection mechanisms, or other dispatching strategies [44]. It offers benefits such as improved scalability, fault tolerance, and efficient resource utilization. Overall, DNS-based load balancing is a valuable technique for optimizing the distribution of network traffic and enhancing the performance of web services and applications.

CDNs (Content Delivery Networks) often utilize DNS-based load balancing to efficiently distribute network traffic and improve the performance of content delivery [28]. CDNs strategically place distributed servers or point-of-presence (PoPs) in various geographic locations.

When a client requests content, the DNS resolver directs the client to the nearest CDN server based on different load balancing schemes. This approach aims to minimize latency and optimize resource utilization. For example, if a CDN server becomes unavailable, the DNS resolver can reroute users to an alternative healthy server, ensuring uninterrupted service. For such scheme to work, CDNs usually employ a small TTL value [1]. Small TTL values allow CDNs to quickly shift traffic to different server locations. So, it is imperative for resolvers to honor DNS TTL, if they extend the TTL, such schemes will fail and hurt the responsiveness of CDNs.

2.3 DNSSEC

The Domain Name System Security Extensions (DNSSEC) [4] is a set of extensions to the Domain Name System (DNS) protocol to provide authentication and data integrity. DNSSEC strengthens the security of DNS by protecting against common attacks like spoofing, cache poisoning, and man-in-the-middle attacks.

The main goal of DNSSEC is to protect the DNS infrastructure and prevent attackers from manipulating DNS responses or impersonating legitimate DNS servers. By using cryptographic signatures, DNSSEC allows DNS resolvers to verify the authenticity of DNS data received from authoritative DNS servers. This ensures that users are directed to the correct IP addresses associated with domain names and prevents attackers from redirecting users to malicious websites.

The deployment of DNSSEC involves several components, including DNSSEC-aware name servers and DNS resolvers. In the context of DNSSEC, name servers generate digital signatures for DNS records and store corresponding public keys in DNSKEY records. Resolvers, on the other hand, are responsible for querying name servers and validating the authenticity of DNS responses using the stored public keys.

Fundamentally, DNSSEC works by adding digital signatures in order to authenticate DNS records. When a resolver queries an authoritative DNS server, the authoritative DNS server will sign the record with a digital signature and include that signature in the response along with the records. When the resolver gets the response, it can use the digital signature to verify the authenticity of the record. If it can validate the signature, we can be sure that the source of the record is the authoritative DNS server and has not been tampered with.

DNSSEC introduces several new record types to the DNS:

- **DNSKEY:** This resource record holds the public key used for DNSSEC. It is used to verify the authenticity of other DNS records.
- **RRSIG:** This record contains the digital signature for a specific DNS record set. It is used to verify the integrity and authenticity of the associated DNS record.
- **DS:** The DS record is used to establish a chain of trust between parent and child zones

in DNSSEC. It contains a hash of the child zone's DNSKEY record, which is used to verify the child zone's authenticity.

- **NSEC/NSEC3:** The NSEC record is used to provide authenticated denial of existence. It proves that a specific DNS record does not exist in a zone. Similar to NSEC, the NSEC3 record provides authenticated denial of existence. However, it adds additional security by hashing the names of the existing DNS records.

2.4 Brightdata

We used BrightData [10] residential proxy service in this study to evaluate DNS resolvers around the world. This is a paid proxy service that can route HTTP/S traffic requests through residential exit nodes.

To use this service, the measurement client has to route HTTP requests through proxy servers controlled by BrightData. These proxy servers forward the requests to residential proxies around the world that we will term as exit nodes throughout this work. These exit nodes can execute the HTTP request by contacting the webserver and returning the response to the measurement client through the proxy servers. Figure 2.2 depicts a high-level overview of the BrightData platform's operation.

BrightData platform also has several options that the users can embed in the request URL to control exit node preference and behavior.

Exit node preference: BrightData gives clients a useful option to choose the location of the exit node. The client can specify the exit node's country or autonomous system (AS) by adding a -country-XX or -asn-YY parameter to the HTTP request, where XX is the Alpha-2 country code and YY is the AS number. The client can also add a session to the HTTP

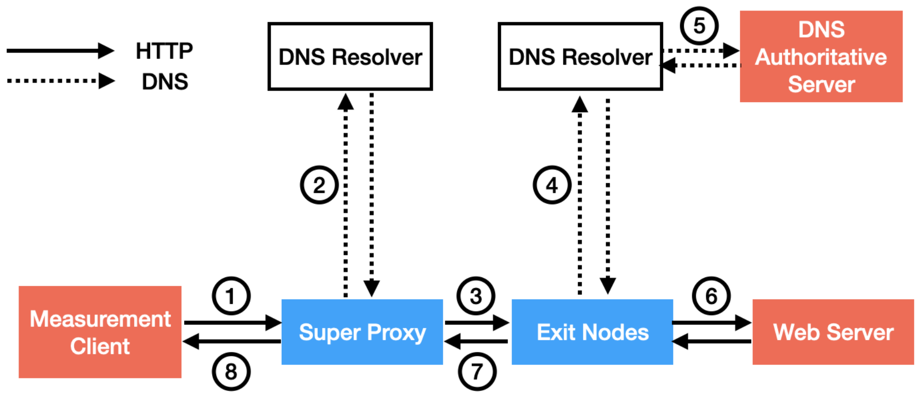

Figure 2.2: Timeline of an HTTP request in BrightData proxy: first the client sends the request to the super proxy. The super proxy checks if the domain is valid by performing a DNS resolution. After that, it forwards the request to an exit node. The exit node performs a DNS resolution using its designated DNS resolver and fetches the HTTP response from the webserver. Then the response is propagated back to the client through the superproxy.

request by using a -session-XX parameter where XX is a random string. Conveniently, the client can choose the same exit node for a small period of time by using the same session identifier.

Exit node persistence: Brightdata platform provides the hash of the exit node's IP address in the HTTP response in the x-luminati-ip header. Now, the client can target the same exit node by adding the -ip-XX parameter, where XX is hash value obtained from the x-luminati-ip header. This option is better than using the session parameter as the session is short-lived. For our experiment, we need to send subsequent HTTP requests through the same exit node after a certain time (eg: the TTL period), so this option is quite useful.

DNS request location: By default, the superproxy performs the DNS resolution and propagates the A record to the exit node so that it does not require to do a DNS resolution.

But, in this experiment, we want to evaluate the DNS resolvers around the world. Fortunately, there is a dns-remote option that we can add to the HTTP request so that the exit node also performs DNS resolution using its designated DNS resolver.

Chapter 3

Related Works

There have been a lot of research efforts dedicated to the TTL violations in the Domain Name System. These studies have employed various datasets and research methodologies to focus on the performance, caching, and security implications of DNS TTL.

Firstly, TTL value is important for DNS performance; smaller TTLs can lead to increased name resolution latency, higher nameserver workload, and increased DNS traffic [13]. Also, nameservers might choose to ignore the TTL entirely and cache DNS entries for longer periods for performance reasons, even though this violates the DNS RFC [26]. Jung et al. [32] measured how DNS cache hit rate changes by changing TTLs leveraging using trace-driven simulations.

TTL violation can have serious implications for DNS caching effectiveness. Overriding the advocated TTL with an arbitrarily chosen value, as done by modern DNS software, can affect the balance of DNS query rates and cache behavior [35, 22]. For example, manipulating TTLs by older BIND nameservers has been reported by Shaikh et al. [46]. In an early study (2004) conducted by Pang et al. [40], DNS logs provided by Akamai were utilized to measure the pervasiveness of TTL violations. They found out that 47% of users used stale A records, stressing that TTL violation is quite prevalent.

Similarly, back in 2013, Callahan et al. [12] reported that 13.7% of user connections in residential networks, spanning 90 homes, employed expired DNS records. Schomp et al. [45]

carried out an active measurement by querying public resolvers via 100 PlanetLab nodes. They discovered that 81% of those resolvers failed to respond with accurate TTL values. Fujiwara et al. [23] proposed a model for TTL-based Internet caches and emphasized the caching importance of shared resolvers. They suggested that understanding the caching behavior and optimizing TTL values can improve DNS performance.

Furthermore, some studies [19, 3] have found out that TTL violations tend to be more frequent with shorter TTLs. For instance, Flavel et al. [21] observed that 2% of users continued to use expired DNS responses with a 20-second TTL even after a 15-minute duration. Almeida et al. [2] identified similar trends in data collected from a network operator. RIPE Labs also conducted a study employing 9,119 distinct probes, revealing that 4.1% of the resolvers violated the TTL, while 1.97% decreased the TTL value [37].

Additionally, there have been research studies that discussed the security implications of DNS caching. Shulman et al. [47] stated that public DNS resolvers can reduce DNS traffic by caching DNS records, which can also provide security benefits. Bassil et al. [5] proposed a new protocol called "S-DNS" that helps to prevent DNS spoofing and cache poisoning. DNS pinning violates RFC 2616, which states that HTTP clients must observe the TTL information reported by Johns et al. [31]. Furthermore, DNS cache poisoning attacks, demonstrated by Kaminsky, highlight the importance of adhering to TTL values to ensure the integrity of DNS records [30].

Although these studies have uncovered various aspects TTL violations, it remains a challenging task to provide a comprehensive assessment of TTL violations on the internet. This challenge arises from the fact that each study employed different research approaches and focused on different types of resolvers, mostly focusing on public resolvers. In this work, our objective is to devise a method that can achieve the same goal without requiring privileged access to devices or networks, and without needing substantial efforts for the deployment of software or hardware for user installations.

Chapter 4

TTL extension

TTL extension is a practice where a DNS resolver extends the TTL of a DNS record beyond the value specified by the authoritative DNS server [6]. This can be done for a variety of reasons, Firstly, it can reduce the load on DNS servers as the resolvers can cache records for a longer time without needing to query the authoritative servers frequently. Secondly, it can help to improve latency as clients can resolve hostnames from the resolver cache. However, TTL extension can also have a number of negative consequences, including:

Reduced reliability: TTL extension can reduce the reliability of DNS resolution, as resolvers may continue to serve DNS records that are no longer valid.

Reduced security: TTL extension can make it easier for attackers to poison DNS caches, as they have more time to propagate their malicious DNS records.

Slower propagation: When a DNS record is changed in an authoritative server, it will take longer to propagate to the DNS resolvers if they cache the records further than the authoritative TTL. For example, if a website is moving to a new IP address, extending the TTL can negatively impact it.

This chapter explains the methodology, results, and evaluation process of measuring TTL extending resolvers out in the wild.

4.1 Methodology

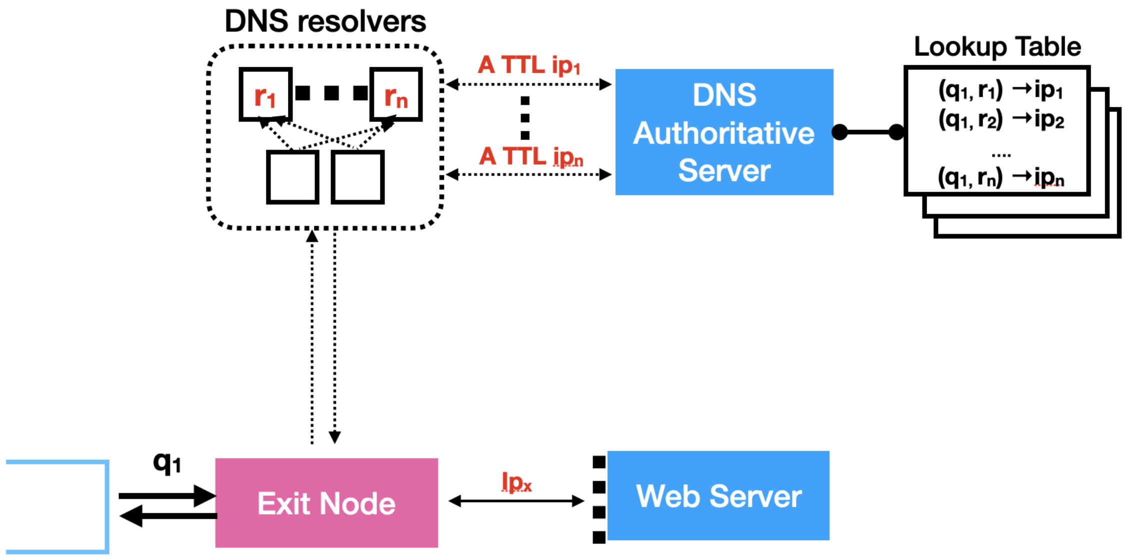

Figure 4.1: TTL extension experiment setup: we control the the DNS authoritative server and the web server (which are marked blue). In the DNS server we keep a mapping between the resolvers and allocated A records so that we can identify which DNS response was used by the exitnode by monitoring the visited webserver.

Our experiment setup hinges on a very straightforward idea. We set up an authoritative server for a domain under our control. Then, we host two webservers in two different machines with IP addresses IP_1 and IP_2. After that, we target a specific exit node and retrieve the domain through an HTTP request by using Brightdata platform. We set our DNS server to return IP_1 as the A record in this turn. From the logs of our DNS authoritative server, we can also identify the resolver used by the client by observing the source IP addresses of incoming DNS requests. Then we wait out the TTL period assuming that the DNS resolver used by the client should have evicted the record.

Now, we modify the A record to IP_2 at our authoritative server. After doing so, we again choose the same exit node by using the previously stored IP hash and connect to the same domain. Normally, it should connect to IP_2 if the resolver correctly evicted the old record and fetched the new record. In this case, the resolver honors TTL and can be labeled as a TTL-honoring resolver.

However, there is an inherent limitation to this methodology. DNS resolver architecture can be complex and multi-layered [9]. The client can contact multiple resolvers which by turn can connect to many upstream resolvers [38, 32]. Consequently, the client may receive multiple DNS responses. This is a practice commonly known as DNS forwarding. DNS forwarding is a mechanism in the Domain Name System (DNS) that allows one DNS server or resolver to send DNS queries to another DNS server on behalf of a client or local network. It is a way to offload DNS resolution to a different DNS server. DNS forwarding serves several purposes like performance improvement, improved caching and load balancing.

So, identifying the client resolver can get tricky as we might see multiple querying resolvers in our authoritative server. We need to adapt our methodology to identify which resolver's response was ultimately used by the client so that we can correctly evaluate it.

As illustrated in Figure 4.1, we bring some changes to our initial setup. We design our authoritative server to generate different A records for distinct resolvers. We also keep a mapping of the resolver to corresponding served A records in our dynamic DNS server. Let's go into detail about how this experiment setup helps us to overcome the previously discussed hurdle.

- In the initial phase that we will call Phase 1, we fetch a unique subdomain, ex: http://«unique-identifier».ttlexp.com through an exit node.

- We find the IP hash from the x-luminati-ip header. We save this header for later usage so that we can select the same exit node again.

- Now, for each resolver that queries our authoritative name server for that qname, we allocate an unused IP address from our available pool and return the A record containing that IP address. We keep track of the mapping between the resolver IP tuple to the served IP address in our mapping table.

- Now, after observing which webserver the exit node connects to we can deduce which A record was used by the exit node. With this information, we can consult our mapping table to find out which resolver was actually used by the exit node.

- Subsequently, we stop our DNS authoritative server. We wait for the expiration of the TTL so that the DNS records are expired.

- After the TTL expiry, we configure our authoritative nameserver to serve an A record that directs to an IP address (IP_{new}). We ensure that this IP_{new} is an separate IP that was not used in Phase 1.

- Now, we utilize x-luminati-ip header that we stored in phase 1 to retarget the same nodes and let it connect to the same subdomain again. We call this step the Phase 2.

So, with this methodology, we can correctly identify the actual resolver that was used by the client. So, depending on whether the client is connected to the IP_{new} (indicating that the resolver was honoring the TTL) or it still connected to the old IP address (indicating that the resolver was extending the TTL), we can correctly classify the resolvers now.

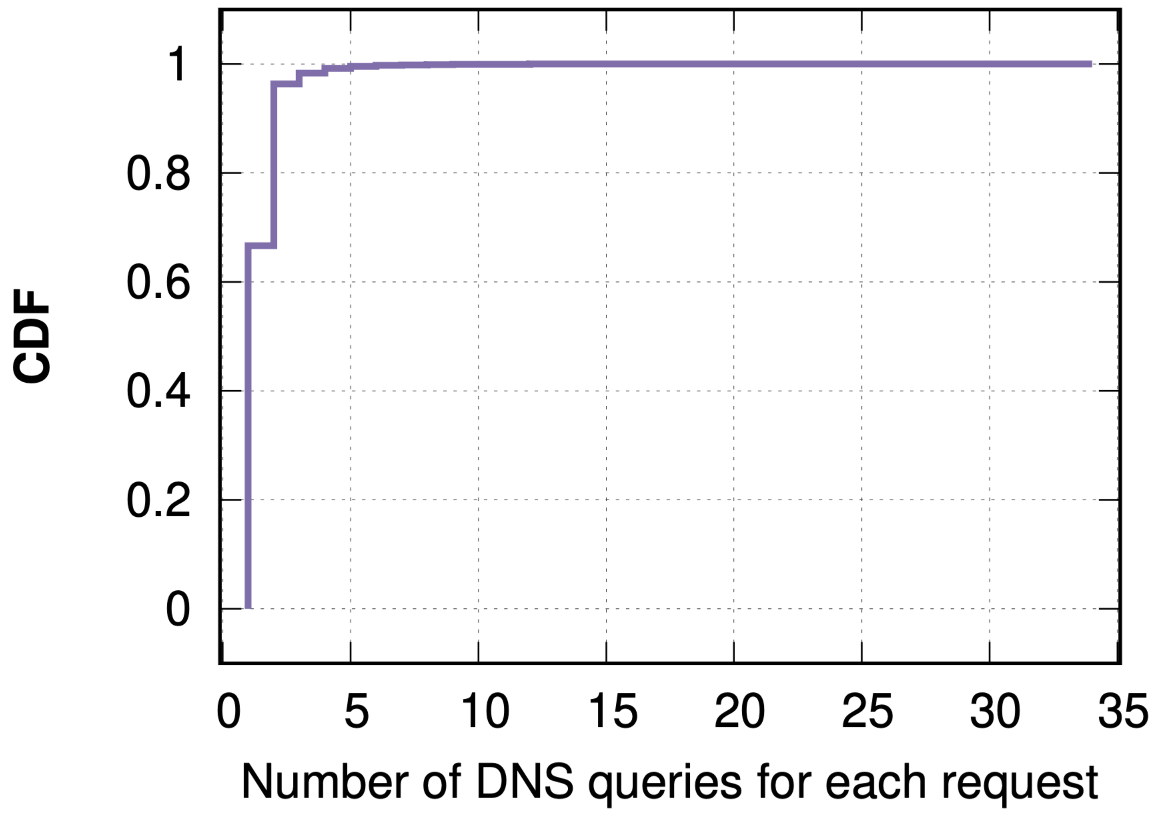

Figure 4.2: CDF of number of DNS queries we observe for each HTTP request through Brightdata platform

4.1.1 IP pool

We have used an IP pool to dynamically generate the A records for our experiment. For each IP we host an Apache webserver and log all incoming HTTP requests. As we see in Figure 4.2, for 99% of HTTP requests we sent through Bright data platform, we observed at most 8 DNS resolvers queried our authoritative server. With this observation, we have used 8 different IP addresses in our pool.

4.2 Results

We ran our experiment for a total of two weeks and in this period we sent a total of 2,068,686 unique HTTP requests through Brightdata platform. In this period, we observed a total of 274,570 exit nodes and 27,131 associated DNS resolvers. These exit nodes were distributed across 220 countries and 9,514 autonomous systems.

Also, in our experiment, we used five different TTLs (1, 5, 15, 30, and 60 minutes), the goal was to see how TTL violation varies with the TTL values. In our experiment setup, when we see that an exit node fetches from an web server associated with the old IP address, we can recognize that the resolver it is using is a potential TTL-extending resolver. For finding out which resolver the exit node is using, we can consult the mapping table.

Now, even if an exit node connects to the new webserver, we can not directly label its resolver as a TTL-honoring resolver. This is due to the fact that some resolvers only appear in the initial phase. So, we employ a criteria: we only categorize a resolver as TTL-violating only if they appear in both Phase 1 and Phase 2 of our experiment.

Also, we are using exit nodes as proxies for understanding the behavior of DNS resolvers, and we can not conduct a direct measurement. So, we can not directly use the results for characterizing the DNS resolver. For example, the exit nodes might have a TTL extending stub resolver [24] that is the actual culprit whereas the egress resolver querying our authoritative server honors TTL. Nevertheless, the client would still connect to the old IP as the stub resolver would be storing stale records. These measurements will falsely label the egress resolver as TTL extending.

To mitigate this issue, we only focus on resolvers for which we have measurement data from at least five exit nodes, so we can make more confident inferences on the behavior of the resolver. After this criteria, we are left with 9,031 resolvers and their associated 234,605 exit

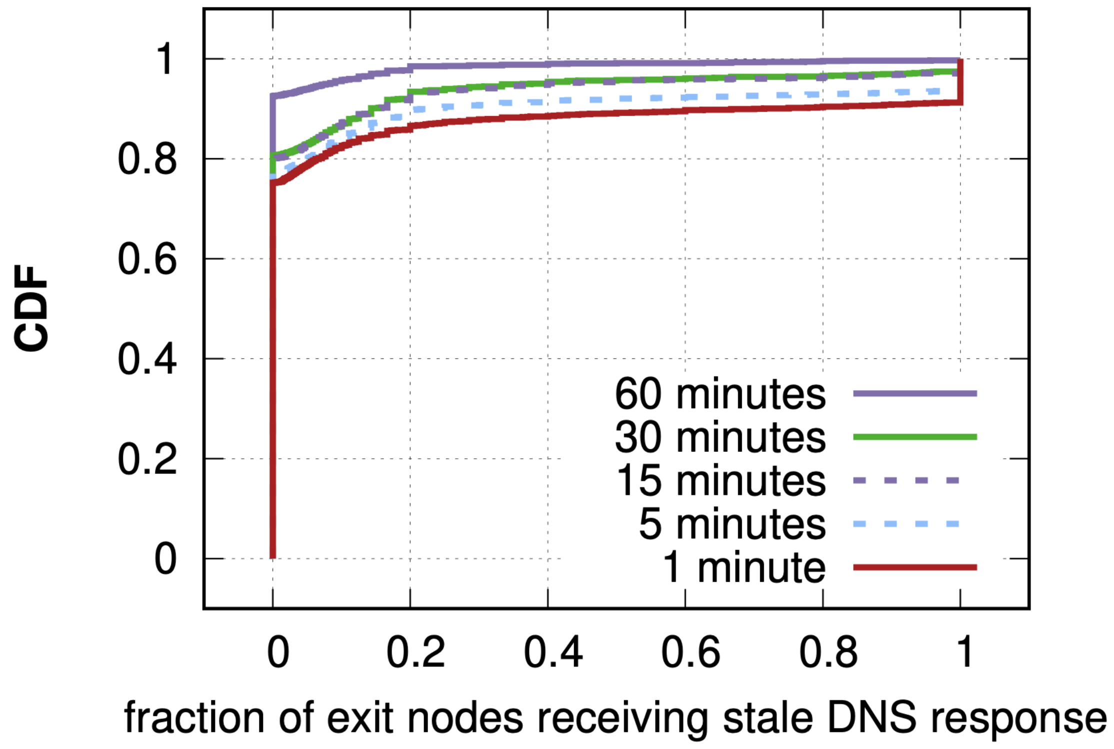

Figure 4.3: CDF of the fraction of exit nodes using stale DNS records

TTL	**Total resolvers**	**ASNs**	**Exitnodes**
1 minutes	8524	4450	133582
5 minutes	6518	3837	84299
15 minutes	5598	3550	67203
30 minutes	5094	3316	58906
60 minutes	4482	3017	49606

Table 4.1: Overall measurement statistics in TTL extension experiment

nodes across various TTL values. Table 4.1 outlines the individual statistics of each TTL setting we have used after applying these criteria.

For evaluating the resolvers, we find out the percentage of exit nodes for each resolver that connects to the old IP address. Ideally for a TTL honoring resolver, this fraction should be 0 as all the exit nodes should correctly connect to the new IP address. On the other hand, for a TTL dishonoring resolver, this fraction should be 1 as all the exit nodes should connect again to the old IP address. After calculating this fraction, we draw the CDF plotting this fraction for each resolver in Figure 4.3. We can make several interesting observations from the figure.

Firstly, the CDF outlines the TTL-extending resolvers and TTL-honoring resolvers and there is a clear separation between them. The vertical line at the left indicates the TTL honoring resolvers with none of its exit nodes connecting to the old webserver. The vertical line at the right comprises the TTL dishonoring resolvers where all of its exit nodes connected to the old webserver. For instance, when we configure TTL values to 60 minutes, the majority of resolvers, 4,147 out of 4,461 (equivalent to 92.5%), honor the TTL, while 0.31% (14 resolvers) extend the TTL, and the remaining 7.1% exhibit mixed behaviors.

Now, this mixed behavior can be attributed to different underlying reasons. For example, there can be presence of stub or other frontend resolvers in the DNS hierarchy. Also, there can be middleboxes that tamper with the DNS. As identifying this is beyond the scope of our experiment, we only consider resolvers where there was no mixed behavior observed.

Secondly, as the TTL values are lowered, the prevalence of TTL-extending resolvers increases. For instance, the proportion of TTL-extending resolvers rises from 0.31% to 2.53%, 2.87%, 6.53%, and finally, 8.74% as the TTL value is set from 60, 30, 15, 5, to 1 minute. We show the trends for all TTL settings in Table 4.2 This strongly suggests that resolvers usually

TTL	**Dishonoring resolvers**
1 minutes	745 (8.74%)
5 minutes	414 (6.35%)
15 minutes	161 (2.87%)
30 minutes	129 (2.53%)
60 minutes	14 (0.31%)

Table 4.2: TTL extending resolvers in different TTL setting

employ a minimum TTL value, possibly to reduce extra lookups and improve performance. We have found out that this min-ttl setting is present in DNS software implementations such as Bind, PowerDNS, KnotDNS, and Unbound [8, 41, 34, 8].

4.3 Cross Validation

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

Table 4.3: Cross-validation with Proxyrack experiment

Now, in our experiment, we could not directly measure the DNS servers. So, we undertake a cross-validation experiment to validate the findings of our study. We focus on only the resolvers from our 1 minute TTL dataset which always honor or extend the TTL (excluding the resolvers showing mixed behavior).

This leaves us with a sample size of 7,160 such resolvers. Firstly, we try to find the open resolvers from this set that we can directly query. For such resolvers, we take a similar strategy like our brightdata experiment. We send a DNS query for a subdomain, and after the TTL expiry, change the A record and resend the same query. If the resolver responds with the stale record, we can conclude that the resolver is extending the TTL.

Now, for evaluating the local resolvers that do not accept DNS queries outside the local network, we leverage a residential proxy service known as ProxyRack [42]. The ProxyRack service covers a limited number of exit nodes compared to the Brighdata platform. But it enables us to send arbitrary UDP traffic through the exit nodes. So, in theory, we can query local resolvers directly if proxyrack provides an exit node in the same network.

Specifically, we try to find if proxyrack has exit nodes in the same Autonomous System (AS) as the local resolvers we are targeting and try to send DNS requests through them. We could query directly 213 resolvers, and we could reach 443 other resolvers through Proxyrack. The outcomes of our experiment are summarized in Table 4.3. We can observe that the results are mostly consistent with our previous findings with the Brightdata platform, except only one resolver showing different behavior. We hypothesize that the resolver operator might have changed its configuration by the time we conducted the Proxyrack experiment. In summary, our findings from the Proxyrack experiment validate the accuracy of our Bright data experiment.

4.4 Geographic Distribution of TTL extending resolvers

Rank	**Country**	**Affected**		**Ratio**
		Exit nodes	**Total**	
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%

Table 4.4: Top 10 countries with the highest fraction of exit nodes using TTL- extending resolvers

In this section, we want to find out the prevalence of TTL extension across different countries. In order to do so, our initial step involves mapping Autonomous Systems (ASes) to Internet

Service Providers (ISPs) and countries. We rely on CAIDA's [11] dataset for this purpose. CAIDA, the Cooperative Association for Internet Data Analysis, is an organization that has focused on network topology measurement and analysis for several years. Then, we categorize exit nodes based on their country of origin and AS. Again, we only consider resolvers with a minimum of five exit nodes.

Our experiment result shows that exit nodes receiving stale DNS records are geographically distributed around the world. Table 4.4 presents a ranking of the top ten countries that has the highest proportion of exit nodes affected by TTL extension. We can see that TTL extension is quite prevalent, for example, the table shows that in China, over 60% of the exit nodes were using TTL-extending resolvers. This emphasizes the widespread occurrence of TTL violations, which can impact users globally.

4.5 TTL Extension in Local Resolvers

As we have seen from the cross-validation with Proxyrack, TTL extension is apparently more prevalent in local resolvers. So, now we particularly focus on local resolvers. When we notice several exit nodes within the same Autonomous System (AS) connecting to a resolver that is also part of the same AS, we classify the resolver as a local resolver. To ensure statistical significance, we concentrate on only those resolvers where we see at least five exit nodes utilizing the same DNS resolver.

Following this methodology, we identify a total of 6,871 local resolvers in our measurement dataset. We map these resolvers to their corresponding countries and ISPs. Table 4.5 presents the top 15 local DNS resolvers that consistently extend the TTL in our experiment. Interestingly, we observe that a major portion of these local DNS resolvers belong to Russia and China. For instance, we have measured 7 DNS resolvers within China Mobile, and they

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 4.5: Top 15 Local Resolvers that dishonors TTL

all extend TTL. This indicates that TTL extension is an ISP-level policy. As we have seen, there are built-in configurations in major DNS servers to enforce a minimum TTL, so it is quite intuitive that all resolvers within an ISP should be configured in the same way.

4.6 Impact of TTL extension on CDNs

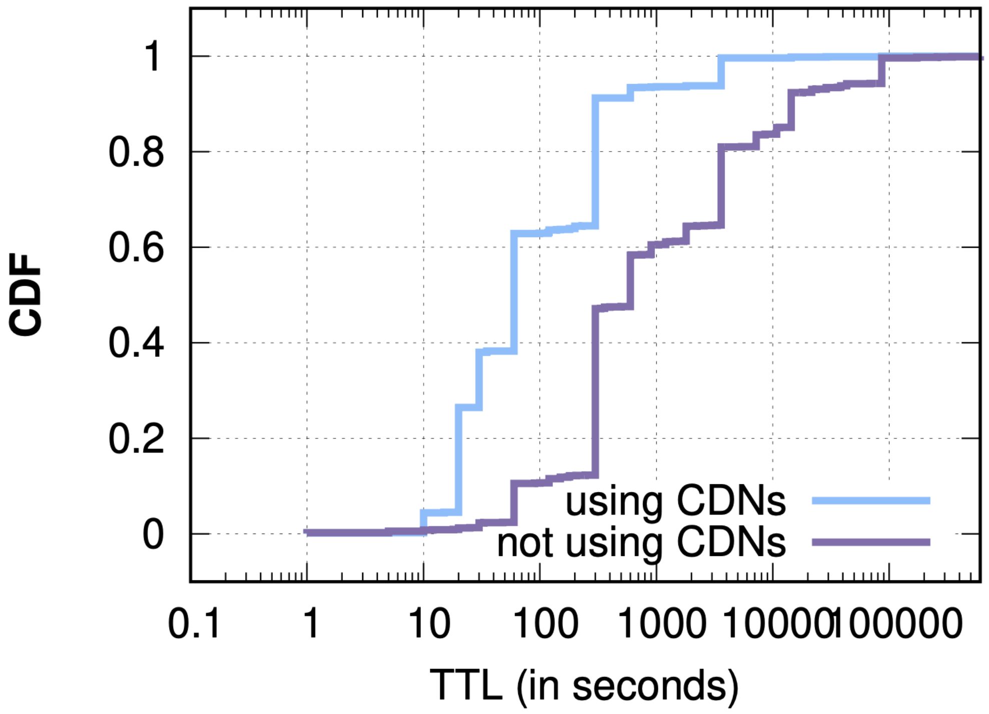

Figure 4.4: CDF of TTL used by Tranco 1M domains

Content Delivery Networks (CDNs) often employ short Time-to-Live (TTL) values for various purposes. The motivation behind this is mainly to enhance performance through load balancing. For example, in case a certain Point of Presence (PoP) of a CDN experiences disruptions, a short TTL can facilitate the redirection of traffic to an alternative PoP quickly. Consequently, if the DNS resolver extends TTL, it can severely compromise the responsiveness of such schemes.

Now we focus on the Tranco 1M dataset, with the aim of identifying domains that rely on CDNs. It is worth noting that CDNs usually employ domain name based load balancing schemes. For example, Fastly [20], which utilizes Canonical Name (CNAME) records to redirect users to CDN infrastructure.

We employ the OpenINTEL [39] dataset, which contains DNS records for the top one million Tranco domains along with CNAMEs. We look for CNAME expansions in these domains' DNS records to verify if they are employed by CDNs. To accomplish this, we made a list of canonical name patterns corresponding to CDNs. Figure 4.4 presents the Cumulative Distribution Function (CDF) of the TTL values for A records. Interestingly, we observe that TTLs for domains hosted in CDNs are notably shorter than other domains. To illustrate, 38% of websites using with CDNs employ a TTL of less than 60 seconds whereas the percentage is 2.43% for websites that do not use CDNs.

We also measure the common TTL values set by popular CDNs. Table 4.6 shows the top 10 CDNs and the most common TTL value they use. The criteria for finding top CDNs is based on the percentage of websites served by them. For example, Netlify uses 20 seconds as TTL. The table clearly shows that a majority of them use very small TTLs. Our experiment found out that many resolvers around the world extended TTLs, especially for short TTL values. For example, 8.74% of resolvers extended TTLs under a 1 minute TTL setting. Such practice can potentially hurt the responsiveness of CDNs.

CDN	**TTL (seconds)**	**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 4.6: Top 10 CDNs and their commonly used TTLs

Chapter 5

TTL violation - how it impacts DNSSEC

DNSSEC introduces multiple new resource records (RR), including Resource Record Signature (RRSIG), DNS Public Key (DNSKEY), NSEC/NSEC3, and DS records. RRSIG records are used to sign and authenticate DNS resource record sets (RRsets) using public key cryptography [18]. These signatures and the associated RRs are published in the same zone.

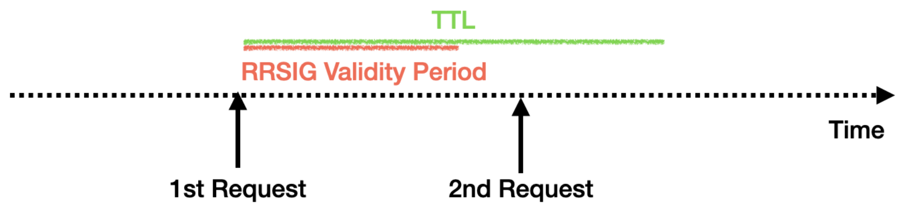

Figure 5.1: Timing of HTTP requests in DNSSEC experiement: the second request is sent when the RRSIG is expired but the TTL has not diminished

The inception and expiration dates in the RRSIG record indicate the time frame when the signature is valid and can be used to verify the corresponding RRset. Also, the DNS records also have a TTL value associated that determines the caching period. In the context of DNSSEC, the RRSIG records play a crucial role in ensuring the integrity and authenticity of

DNS data and it is of utmost importance that the Resolvers only use valid RRSIGs to validate the DNSSEC records. RFC 4033 [4] clearly states that DNSSEC-supporting resolvers should remove DNS responses from the cache when their RRSIGs have expired, even if the TTL still remains. In this section, we try to find out how DNSSEC-supporting resolvers around the world adhere to this specification and correctly evict the RRSIGs after signature expiry, even if TTL remains.

5.1 Methodology

We follow a similar methodology to our TTL extension experiment. But we make some enhancements; firstly, we fully sign our domain names and upload the DS records to the parent zone. Now, as we want to check the resolver's caching behavior in case RRSIG expires but TTL does not, we set out signature expiry time to be less than our TTL value. To elaborate, we configure the TTL value to be 60 minutes for DNS records and set their associated Resource Record Signatures to become invalid after exactly 30 minutes.

As depicted in Figure 5.1, we send the first HTTP request through an exit node, and then we make the second request for the same subdomain through that exit node after the RRSIG has an expired but the TTL has not yet expired. If the DNS resolver retrieves the new A record from the Auth server, it is adhering to the RFC standard. On the other hand, if we see the exit node is still connecting to the old webserver, we can tell that the resolver is using a RRSIG that has expired signature.

5.2 Results

We have conducted our experiment from October 2022 to November 2022. In this period, we observed a total of 91,634 exit nodes and 13,679 associated DNS resolvers. Again, similarly like our previous experiment, we only consider DNS resolvers that had a minimum of 5 associated exit nodes. After we employ this filtering process results in 5,274 resolvers, which represents 38.5% of the initial sample. These resolvers were associated with 75,684 exit nodes, comprising 82.6% of the total exit nodes in our dataset.

DNSSEC-validating resolvers enable the DO (”DNSSEC OK”) bit in the Extended DNS (EDNS) pseudorecord. This is to indicate that they support DNSSEC and signal the authoritative server to provide additional DNSSEC RRs. Any DNSSEC query coming with this bit on signifies the capability and intention to perform DNSSEC validation, thereby enhancing the security and integrity of the DNS resolution process [49].

In our experiment, we observed that 4,917 resolvers, comprising 93.2% of the total had enabled the DO bit. These resolvers covered 94% of exit nodes (71,242 exit nodes). This implies that the majority of DNS resolvers are DNSSEC-validating.

But, nevertheless, it is important to note that not all resolvers supporting DO bit actually validate DNSSEC. A previous study revealed that approximately 82% of resolvers that specified DO bit failed to validate DNSSEC responses properly [15]. So, we implemented an additional step to find out only the DNSSEC-validating resolvers from this set.

We set up an additional test bed where we intentionally host a domain that is incorrectly signed (we place cryptographically incorrect RRSIG records). Now, we send an additional HTTP request through these same exit nodes to that domain. Now, if an exit node only connected to the domain that is correctly signed, it signifies that its resolver was indeed validating DNSSEC. We also consult the DNS server log to check that the resolver queried our authoritative server in both cases. On the other hand, if the exit node was also fetching the incorrectly signed record, we could infer that the DNS resolver was not performing DNSSEC validation. Through this extra step, we discern the actual DNSSEC validation resolvers from the previous set. In total, we identified 646 resolvers, constituting 13.1% of the total, and covering 8.4% of exit nodes, validated DNSSEC correctly.

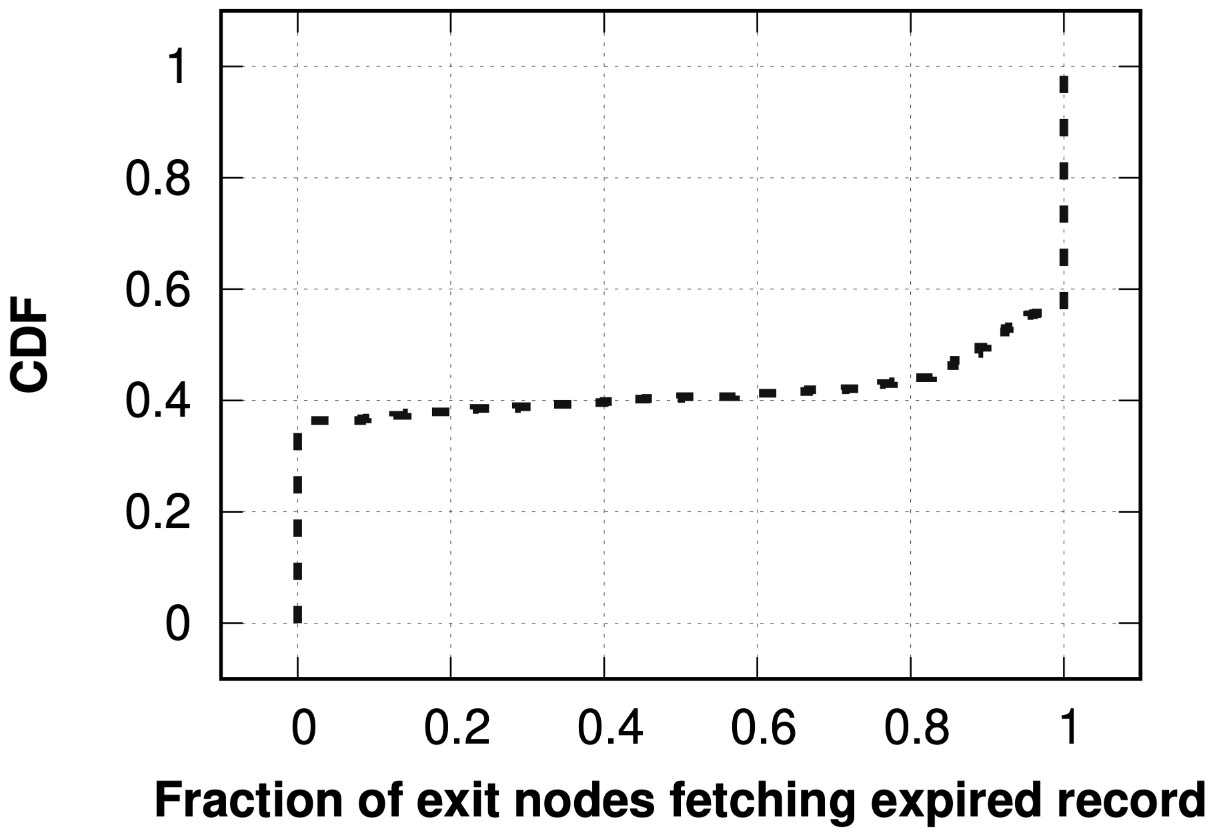

Figure 5.2: CDF of the fraction of exit nodes that fetched expired A records

Now, we proceed to find out the resolvers that do not honor the RFC standard. Like our previous TTL extension study, for each resolver, we find the fraction of exit nodes that fetched stale DNS records. In Figure 5.2, we draw the CDF that plots the fraction for each resolver. As we see, among the DNSSEC validating 646 resolvers, 520 resolvers (80.4% of the total) exhibited consistent behavior. To elaborate, these resolvers had either 0% or 100% of the exit nodes fetching expired A records.

Among them, 235 resolvers, corresponding to 36.3% of the total, requested fresh DNS records from our authoritative server in the second request. This signified that they honor DNSSEC standards by evicting responses with expired RRSIGs. On the other hand, 285 resolvers (44.1%) responded with records with expired RRSIGs and did not fetch fresh records from our authoritative server. So, these resolvers violated DNSSEC standard and did not evict the records even after RRSIG expiry.

Chapter 6

Discussion

6.1 Mixed behavior

DNS resolver architecture can be multi-layered with a complex hierarchy. There can be multiple ingress resolvers that in turn pass on the query to egress resolvers. For example, Cloudflare employs such multi-layered architecture.

But in our methodology, we can only monitor the egress resolvers that query our authoritative server. So, in cases like this, we do not have much visibility of where exactly the TTL violation happens. Our methodology can not identify between the resolvers and servers that sits behind the egress resolvers [29] and we assume the mixed behavior we see in our experiment can be attributed to them.

6.2 Ethical considerations

We use the Brightdata platform to send HTTP requests through actual end users that we have termed as exit nodes throughout this work. Bright-data ethically sources the residential proxies and multiple research articles used this platform as a proxy [14].

These users or peers explicitly need to provide consent through a clear opt-in screen before joining the proxy network. It also clearly lays out the terms to the users. Also, we have strictly followed the terms and conditions set by Brighdata.

Firstly, we only used the commercial services provided by this platform. Secondly, we did not collect any sensitive information like: PII (Personal Identifiable Information) of the users. Lastly, We served a small HTML page from our webservers that had no script associated with it. The content length was below 19 for all webservers. So, it did not affect the regular network usage of the users.

6.3 TTL shortening in the wild

A resolver has flexibility to cache a DNS response for a duration shorter than the Time to Live (TTL) specified in the authoritative server's response. It is important to note that evicting DNS records earlier than TTL does not constitute a breach of the standard, as outlined in RFC 2181. For instance, certain resolvers may incorporate a parameter to determine a maximum TTL, primarily to override excessively large TTL values.

To detect such resolvers that exhibit this behavior, a similar methodology like our TTL extension experiment can be employed. However, it is essential to recognize that resolvers can also make caching decisions based on factors such as cache size and eviction policies, rendering it somewhat challenging to identify resolvers that consistently cache records for a shorter duration than the TTL. By initiating the second request prior to the TTL expiry, we were able to identify 49 resolvers, representing 0.99% of the total (out of 4,965), that consistently truncated the TTL. Additionally, 4,653 resolvers, constituting 93.7% of the total, consistently maintained the original TTL. Nonetheless, we also encountered 263 resolvers, accounting for 5.3% of the total, exhibiting mixed behaviors. This observation suggests that the eviction policy employed by these resolvers may have influenced their caching behavior.

6.4 Impact on DNS-over-HTTPS

DNS-over-HTTPs [17] is gaining rapid popularity with its privacy and security benefits over regular DNS. Mozilla Firefox has turned on DoH as the default option, the other popular browsers have also started supporting this feature. Now, if DNS-over-HTTPs and DNS-over-UDP share the same cache, TTL extension behavior will also affect the customers opting for DoH. We have tested two popular DoH vendors (Google, Cloudflare) and found that DNS-over-UDP and DNS-over-HTTPS share the same cache using the methodology presented by Randall et al. [43]. So, the implications of TTL violation also extend to the DNS-over-HTTPs which is continually growing popular.

6.5 Brightdata exit nodes distribution

To the best of our knowledge, Brightdata exit nodes are not biased. According to Brightdata's official documentation, they have approximately 72 million residential IPs worldwide. In our research, we collected data from more than 274,570 exit nodes and their 27,131 resolvers in 9,514 Autonomous Systems (ASes) spanning 220 countries. We consider these exit nodes to be a representative sample of actual internet clients.

Furthermore, we did not observe a significant fraction of Brightdata clients in countries with internet censorship measures. For example, when we focused on exit nodes originating from countries known for internet censorship [16] (such as North Korea, Iran, Myanmar, UAE, Turkmenistan, Belarus, Oman, Pakistan, Qatar, Syria, Thailand, Turkey, and Uzbekistan), we found that only 13,584 (4.9%) of the exit nodes in our study belonged to these nations. In contrast, 84,889 (29.4%) exit nodes were based in Europe and North America, regions where users typically experience more online freedom and fewer restrictions.

6.6 Popular minimum TTL

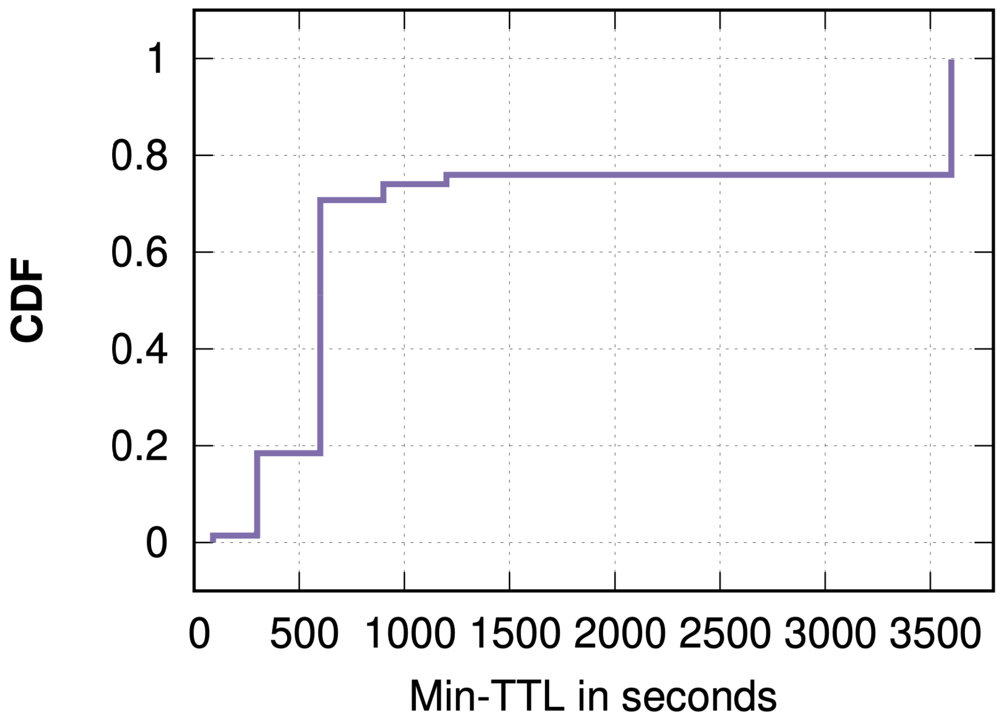

Figure 6.1: CDF of min-ttl values used by resolvers

Common DNS software utilizes the min-ttl parameter to establish a minimum time for retaining cached DNS records before removal. This min-ttl setting overrides the authoritative TTL. In our Proxyrack experiment, we could send DNS queries directly to resolvers that did not honor TTL values. If the initial request returned a DNS record with a TTL differing from the authoritative TTL, we could determine that the resolver was employing a min-ttl setting. As illustrated in Figure 6.1, the most frequently employed min-ttl durations are 10 minutes, followed by 1 hour and 5 minutes.

Chapter 7

Future work

7.1 Cache Eviction Anomalies

In our Proxyrack experiment, it became evident that numerous resolvers utilized a minimum TTL setting, which overrides the authoritative TTL. However, we also saw resolvers who extended the TTL without using this option.

This indicates potential issues with resolver cache eviction mechanisms, or they could not decrement the TTL properly. In our future research, we aim to conduct a more comprehensive analysis and the underlying reasons behind such cases.

7.2 Extending DNSSEC experiment

In the DNSSEC experiment, we used 60 minute TTL value and 30 minute expiry of RRSIGs. Similarly like our TTL extension setup, we will conduct a more thorough experiment in the future. We intend to systematically vary TTL values and RRSIG expiration times, with a goal to find out if the resolver cache expired RRSIGs arbitrarily or upto a certain period.

7.3 Advantages of using a minimum TTL setting

A resolver might use a minimum TTL setting for several reasons. For example, setting a minimum TTL can help improve resolver performance by reducing the need to query authoritative name servers frequently. Also, a minimum TTL can lower the overall volume of DNS traffic on the network.

In the future, we aim to quantify the performance advantages, such as a reduction in outgoing network traffic, that can be achieved through the implementation of this minimum TTL configuration. We plan to accomplish this through active experiments and collaboration with real-world network operators. These findings can assist DNS operators in assessing the advantages and disadvantages of TTL extension, enabling them to make informed decisions.

7.4 DNS record prefetching

Modern DNS servers also employ record prefetching techniques. They proactively monitor the TTL expiry and before a certain time threshold of the expiry, they can send automatic DNS queries to refresh their cache. This ensures that clients are not hung up querying a domain that has been just evicted out of the cache.

For example, Knot resolver software has a predicting module to facilitate proactive caching [33]. There are two supported modes: 1) any time the resolver answers a query for a domain that is about to expire, it proactively fetches the up-to-date record; 2) it also has a prediction mechanism to learn from query patterns and repetitive queries and prefetch certain records that are more likely to be queried in a specific time-frame. BIND supports a prefetch mechanism with a configurable trigger option [7]. Any record that queries within trigger seconds of its TTL expiry will be proactively fetched again. Unbound's prefetch mechanism proactively fetches new records if a query comes within the last 10% of the TTL period [48]. This threshold is not configurable. Usually, the cache is refreshed with a user query results in a cache miss. This feature has potential for improved user experience as well as reduced latency. In our future work, we also want to measure how many DNS resolvers out there employ such prefetching mechanism.

Chapter 8

Conclusion

In this work, we have introduced a novel methodology for assessing TTL violations in DNS resolvers leveraging the BrightData residential proxy network. Our key findings highlight the prevalence of TTL violations, with approximately 8.74% of the measured resolvers arbitrarily extending TTL values, potentially impacting the performance of content delivery networks (CDNs). Notably, these violations are more prevalent among local Internet Service Provider (ISP) resolvers, especially in regions like Russia and China. Additionally, the research reveals a significant concern with DNSSEC-validating resolvers where they incorrectly serve expired RRSIGs from the cache, violating DNSSEC standards. These insights underscore the importance of establishing standard practices for DNS caching and TTL enforcement and also provide a foundation for further study of DNS resolver behavior on a global scale.

Bibliography

[1] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig. "Comparing dns resolvers in the wild". In: (2010). DOI: 10.1145/1879141.1879144.

[2] M. Almeida, A. Finamore, D. Perino, N. Vallina-Rodriguez, and M. Varvello. "Dissecting DNS Stakeholders in Mobile Networks". In: 2017.

[3] H. A. Alzoubi, M. I. Rabinovich, and O. Spatscheck. "The anatomy of LDNS clusters: findings and implications for web content delivery". In: 2013.

[4] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. *RFC 4033: DNS security introduction and requirements*. 2005.

[5] R. Bassil, R. Hobeica, W. Itani, C. Ghali, A. Kayssi, and A. Chehab. "Security analysis and solution for thwarting cache poisoning attacks in the domain name system". In: (2012), pp. 1–6.

[6] P. Bhowmick, M. I. Ashiq, C. Deccio, and T. Chung. "TTL Violation of DNS Resolvers in the Wild". In: *International Conference on Passive and Active Network Measurement*. Springer. 2023, pp. 550–563.

[7] *Prefetching records in Bind resolver*. https://kb.isc.org/docs/aa-01122.

[8] *BIND9*. https://www.isc.org/bind/.

[9] J.-Y. Bisiaux. "DNS threats and mitigation strategies". In: *Network Security* 2014.7 (2014), pp. 5–9.

[10] *BrightData*. https://www.brightdata.com.

[11] *Caida*. https://www.caida.org/.

[12] T. Callahan, M. Allman, and M. Rabinovich. "On Modern DNS Behavior and Properties". In: *SIGCOMM Comput. Commun. Rev.* 43.3 (July 2013), pp. 7–15. ISSN: 0146-4833.

[13] X. Chen, H. Wang, S. Ren, and X. Zhang. "Maintaining strong cache consistency for the domain name system". In: *IEEE Transactions on Knowledge and Data Engineering* 19.8 (2007), pp. 1057–1071.

[14] R. Chhabra, P. Murley, D. Kumar, M. Bailey, and G. Wang. "Measuring DNS-over-HTTPS performance around the world". In: *Proceedings of the 21st ACM Internet Measurement Conference.* 2021, pp. 351–365.

[15] 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". In: *26th USENIX Security Symposium (USENIX Security 17).* 2017, pp. 1307–1322.

[16] J. D. Clark, R. M. Faris, R. J. Morrison-Westphal, H. Noman, C. B. Tilton, and J. L. Zittrain. "The shifting landscape of global internet censorship". In: (2017).

[17] *How DNSSEC Works.* https://www.cloudflare.com/dns/dnssec/how-dnssec-works/.

[18] *How DNSSEC Works.* https://www.cloudflare.com/dns/dnssec/how-dnssec-works/.

[19] *Edge and Browser Cache TTL.* https://developers.cloudflare.com/cache/about/edge-browser-cache-ttl/.

[20] *Fastly.* https://www.fastly.com//.

[21] A. Flavel, P. Mani, and D. A. Maltz. "Re-evaluating the responsiveness of DNS-based network control". In: 2014.

[22] N. C. Fofack and S. Alouf. “Modeling modern DNS caches”. In: (2013), pp. 184–193.

[23] K. Fujiwara, A. Sato, and K. Yoshida. “Cache effect of shared DNS resolver”. In: *IEICE Transactions on Communications* 102.6 (2019), pp. 1170–1179.

[24] X. T. Gorjón and W. Toorop. “Discovery method for a DNSSEC validating stub resolver”. In: *University of Amsterdam, MSC System and Network Engineering* (2015).

[25] P. Hoffman and J. Schlyter. *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*. RFC 6698. Internet Engineering Task Force, Aug. 2012. URL: http://www.ietf.org/rfc/rfc6698.txt.

[26] B. Hong, S. Bae, and Y. Kim. “GUTI Reallocation Demystified: Cellular Location Tracking with Changing Temporary Identifier.” In: (2018).

[27] Y. S. Hong, J. No, and S. Kim. “DNS-based load balancing in distributed Web-server systems”. In: *The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06)*. IEEE. 2006, 4–pp.

[28] C. Huang, A. Wang, J. Li, and K. W. Ross. “Measuring and evaluating large-scale CDNs”. In: *ACM IMC*. Vol. 8. 2008, pp. 15–29.

[29] S. Huang, F. Cuadrado, and S. Uhlig. “Middleboxes in the Internet: a HTTP perspective”. In: *2017 Network Traffic Measurement and Analysis Conference (TMA)*. IEEE. 2017, pp. 1–9.

[30] P. Jeitner and H. Shulman. “Injection Attacks Reloaded: Tunnelling Malicious Payloads over {DNS}”. In: (2021), pp. 3165–3182.

[31] M. Johns. “Code-injection vulnerabilities in web applications—exemplified at cross-site scripting”. In: (2011).

[32] J. Jung, E. Sit, H. Balakrishnan, and R. Morris. “DNS performance and the effectiveness of caching”. In: (2001), pp. 153–167.

[33] *Prefetching records in Knot resolver*. https://knot-resolver.readthedocs.io/en/stable/modules-predict.html.

[34] *KnotDNS*. https://www.knot-dns.cz/.

[35] T. Mackus, T. Simonaitis, and D. Tamulioniene. “Adaptive TTL based approach to balance DNS server load”. In: *Elektronika ir Elektrotechnika* 117.1 (2012), pp. 81–84.

[36] D. Margolis, M. Risher, B. Ramakrishnan, A. Brotman, and J. Jones. *SMTP MTA Strict Transport Security (MTA-STS)*. Internet Engineering Task Force, Sept. 2018. URL: https://tools.ietf.org/rfc/rfc8461.txt.

[37] 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.

[38] M. Nawrocki, M. Koch, T. C. Schmidt, and M. Wählisch. “Transparent forwarders: an unnoticed component of the open DNS infrastructure”. In: *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*. 2021, pp. 454–462.

[39] *OpenINTEL*. https://www.openintel.nl/.

[40] J. Pang, A. Akella, A. Shaikh, B. Krishnamurthy, and S. Seshan. “On the Responsiveness of DNS-Based Network Control”. In: 2004.

[41] *PowerDNS*. https://www.powerdns.com/downloads.html.

[42] *ProxyRack*. https://www.proxyrack.com.

[43] 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”. In: *Proceedings of the ACM Internet Measurement Conference*. 2020, pp. 50–64.

[44] J. Ruohonen. “Measuring Basic Load-Balancing and Fail-Over Setups for Email Delivery via DNS MX Records”. In: (2020), pp. 815–820.

[45] K. Schomp, T. Callahan, M. Rabinovich, and M. Allman. “On Measuring the Client-Side DNS Infrastructure”. In: 2013.

[46] A. Shaikh, R. Tewari, and M. Agrawal. “On the effectiveness of DNS-based server selection”. In: 3 (2001), pp. 1801–1810.

[47] H. Shulman. “Pretty bad privacy: Pitfalls of DNS encryption”. In: (2014), pp. 191–200.

[48] *Prefetching records in Unbound resolver*. https://docs.netgate.com/pfsense/en/latest/services/dns/resolver-advanced.html.

[49] D. Wessels, W. Kumari, and P. Hoffman. “Signaling trust anchor knowledge in dns security extensions (dnssec)”. In: (2017). DOI: 10.17487/rfc8145.