When the Dike Breaks: Dissecting DNS Defenses During DDoS (extended)

USC/ISI Technical Report ISI-TR-725

May 2018

Giovane C. M. Moura
SIDN Labs and TU Delft

John Heidemann
USC/Information Sciences Institute

Moritz Müller
SIDN Labs and University of Twente

Ricardo de O. Schmidt
University of Passo Fundo

Marco Davids
SIDN Labs

ABSTRACT

The Internet's Domain Name System (DNS) is a frequent target of Distributed Denial-of-Service (DDoS) attacks, but such attacks have had very different outcomes—some attacks have disabled major public websites, while the external effects of other attacks have been minimal. While on one hand the DNS protocol is relatively simple, the *system* has many moving parts, with multiple levels of caching and retries and replicated servers. This paper uses controlled experiments to examine how these mechanisms affect DNS resilience and latency, exploring both the client side's DNS *user experience*, and server-side traffic. We find that, for about 30% of clients, caching is not effective. However, when caches are full they allow about half of clients to ride out server outages that last less than cache lifetimes, caching and retries together allow up to half of the clients to tolerate DDoS attacks longer than cache lifetimes, with 90% query loss, and almost all clients to tolerate attacks resulting in 50% packet loss. While clients may get service during an attack, tail-latency increases for clients. For servers, retries during DDoS attacks increase normal traffic up to 8×. Our findings about caching and retries help explain why users see service outages from some real-world DDoS events, but minimal visible effects from others.

KEYWORDS

DNS, recursive DNS servers, caching, DDoS attacks, authoritative servers

ACM Reference Format:
Giovane C. M. Moura, John Heidemann, Moritz Müller, Ricardo de O. Schmidt, and Marco Davids. 2018. When the Dike Breaks:, Dissecting DNS Defenses During DDoS (extended) : USC/ISI Technical Report ISI-TR-725, May 2018 . In . ACM, New York, NY, USA, 20 pages.

1 INTRODUCTION

DDoS attacks have been growing in frequency and intensity for more than a decade. Large attacks have grown from 100 Gb/s in 2012 [4] to over 1 Tb/s in 2017 [34], and 1.7 Tb/s in 2018 [18, 22]. Such attacks are sourced from large botnets (for example, with Mirai peaking at 600k hosts [3]), fueled by the continued deployment of new devices. Gigabit-size attacks are commodities today, selling for a few dollars via DDoS-as-a-Service [44].

The Internet's Domain Name System (DNS) is a popular target of DDoS attacks. DNS is a very visible target, since name resolution is a necessarily step in almost any Internet activity. Root DNS servers have seen multiple attacks over more than a decade [23, 33, 41, 42, 54], as well as threats of attacks [49]. Other authoritative DNS servers have also been attacked, with the huge October 2016 attack against Dyn [14] resulting in disruptions at a number of prominent websites, including Twitter, Netflix and the New York Times [34].

The *outcome* of these attacks on services has varied considerably. The October 2016 Dyn attack is noted for disruption to websites that were using Dyn as their DNS provider, and extortion attempts often include DDoS [35]. However, multiple attacks on the DNS Root have occurred with, as far as has been reported, no visible service outages [41, 42].

An important factor in DNS resilience is heavy use of caching—we believe that differences in use of DNS caching contribute to the very different outcomes when DNS is subject to DDoS attack. Yet understanding DNS caching is difficult, with requests traveling from *stub resolvers* in web browsers and at client computers, to *recursive resolvers* at ISPs, which in turn talk to multiple *authoritative DNS servers*. There are many parts involved to fully resolve a DNS name like www.example.com: while the goal is an IP address (an A or AAAA DNS record), multiple levels of the hierarchy (root, .com, and .example.com) are often on different servers (requiring NS records), and DNSSEC may require additional information (RRSIG, DNSKEY, and DS records). Each of these records may have different cache lifetimes (TTLs), by choice of the operator or because of DNS cache timeouts. We explore caching through controlled experiments (§3) and analysis of real-world use (§4).

Another factor in DNS resilience is recursives that retry queries when they do not receive an answer. Recursives fail to receive answers occasionally due to packet loss, but pervasively during a DDoS attack. We examine how retries interact with caching to mitigate DDoS attacks for loss during DDoS attacks (§5) and their effects on authoritatives (§6).

This paper assesses DNS resilience during DDoS attacks, with the goal of explaining different outcomes from different attacks (§8) through understanding the role of DNS caching, retries, and use of multiple DNS recursive resolvers. It is common knowledge

ISI-TR-724b, May 2018 (updated September 2018),

that these factors "help", but knowing *how* and *how much* each contributes builds confidence in defenses. We consider this question both as an operator of an authoritative server, and as a user, defining the *DNS user experience* latency and reliability users should expect.

Our first contribution is to build an end-to-end understanding of DNS caching. Our key result is that *caching often behaves as expected, but about 30% of the time clients do not benefit from caching.* While prior work has shown DNS resolution infrastructure can be quite complex [48], we establish a baseline DNS user experience by assessing the prevalence of DNS caching in the "wild" through both active measurements (§3) and through analysis of passive data from two DNS zones (.nl and the root zone §4).

Our second contribution is to show that *DNS mechanisms of caching and retries provide significant resilience client user experience during denial-of-service (DDoS) attacks* (§5). For example, about half of the clients continue to receive service during a full outage if caches are filled and do not expire during the attack. Often DDoS attacks cause very high loss, but not a complete outage. When a few queries succeed, caches amplify their benefits, even for attacks that are longer than cache lifetime. With very heavy query loss (90%) on all authoritatives, full caches protect half of the clients, and retries protect 30%. With a DDoS that causes 50% packet loss, nearly all clients succeed, although with greater latency than typical.

Third, we show that there is a large increase in legitimate traffic during DDoS attacks—up to 8× the number of queries (§6). While DNS servers are typically heavily overprovisioned, this result suggests the need to review by how much. It also shows the importance that stub and recursive resolvers follow best practices and exponentially back-off queries after failure so as to not add fuel to the DDoS fire.

Our final contribution is to suggest why users have seen relatively little impact from root servers DDoSes, while customers from some DNS providers quickly felt attacks (§8). When cache lifetimes are longer than the duration of a DDoS attack, many clients will see service for names popular enough to be cached. While many websites use short cache timeouts to support control with DNS-based load balancing, they may wish to consider longer timeouts as part of strategies for DDoS defense. Retries provide additional coverage, preventing failures during large attacks.

This technical report [25] follows the conference paper at ACM IMC 2018 [26], adding additional information in several appendices to fill in details about some experiments and add additional scenarios that supplement the paper. The technical report was first released in May 2018, and was updated in September 2018 to include input from the IMC reviewers and shepherd.

All public datasets from this paper is available [24], with our RIPE Atlas data also available from RIPE [38]. Privacy concerns prevent release of .nl and Root data (§4).

2 BACKGROUND

As background, we briefly review the components of the DNS ecosystem and how they interact with IP anycast.

2.1 DNS Resolvers: Stubs, Recursives, and Authoritatives

Figure 1 shows the relationship between three components of DNS resolvers: stubs and recursive resolvers and authoritative servers.

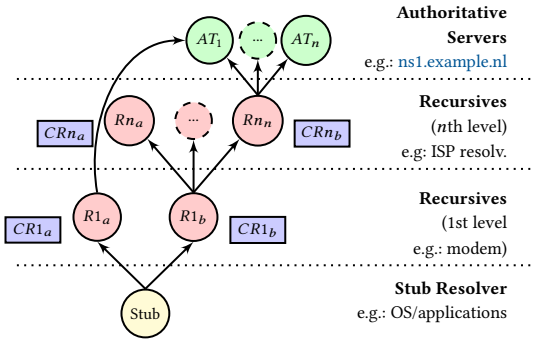


Figure 1: Relationship between stub resolver (yellow), recursive resolvers (red) with their caches (blue), and authoritative servers (green).

Authoritative servers (authoritatives hereafter) are servers that know the contents of a given DNS zone and can answer queries without asking other servers [11].

Resolvers on the other hand, are servers that can ask, on behalf of others, queries to other servers [20]. *Stub* resolvers run directly on clients and query one or a few *recursive* resolvers (shortened to stubs and recursives here). Recursives perform the full resolution of a domain name, querying one or more authoritatives, while caching responses to avoid repeatedly requesting popular domains (*e.g.,* .com or .google.com). Sometimes recursives operate in multiple tiers, with clients talking directly to *R*1 resolvers, that forward queries to other *Rn* resolvers, that ultimately contact authoritatives.

In practice, stubs are part of the client OS or browser, recursives are provided by ISPs, and authoritatives are run by DNS providers or large organizations. Multi-level recursives might have *R*1 at a home router and *Rn* in the ISP, or might occur in large, public DNS providers.

2.2 Authoritative Replication and IP Anycast

Replication of a DNS service is important to support high reliability and capacity and to reduce latency. DNS has two complementary mechanisms to replicate service. First, the protocol itself supports *nameserver replication* of DNS service for a zone (.nl or example.nl), where multiple servers operate on different IP addresses, listed by that zone's NS records. Second, each of these servers can run from multiple physical locations with *IP anycast* by announcing the same IP address from each and allowing Internet routing (BGP) to associate clients with each anycast site. Nameserver replication is recommended for all zones, and IP anycast is used by most large zones such as the DNS Root and most top-level domains [23, 43]. IP anycast is also widely used by *public resolvers*, recursive resolvers that are open for use by anyone on the Internet, such as Google Public DNS [12], OpenDNS [29], Quad9 [37], and 1.1.1.1 [1].

Figure 2 illustrates the relationship of elements in the DNS infrastructure when anycast is in place. The recursive resolver *R* is announced using an anycast prefix from five different *anycast sites* (*R*1 to *R*5). Each anycast site can have multiple servers, with DNS traffic designated to them by load balancing algorithms.

When the stub resolver sends a DNS query to the anycast address of *R*, BGP forwards the query to the nearest site (*R*3 in this

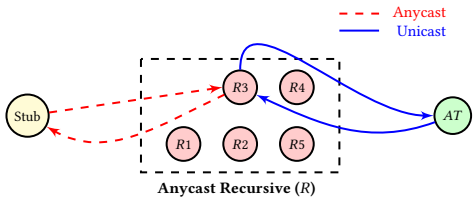


Figure 2: Stub resolver and Anycast Recursive

example) of *R*—note that "nearest" metrics can vary [8]. The BGP mapping between source and anycast destination is known as anycast catchment, which under normal conditions is very stable across the Internet [53]. On receiving the query, *R*3 contacts the pertinent authoritative *AT* to resolve the queried domain name. The communication from *R*3 to *AT* is done using *R*3's unicast address, ensuring the reply from *AT* is sent back to *R*3 instead of any other anycast site from *R*. On receiving the answer from *AT*, *R*3 replies to the stub resolver with the answer to its query; and for this reply *R*3 uses its anycast address as source.

2.3 DNS Caching with Time-to-Live (TTLs)

DNS depends on caching to reduce latency to users and load on servers. Authoritatives provide responses that are then cached in applications, stub resolvers, and recursive resolvers. We next describe its loose consistency model.

An authoritative resolver defines the lifetime of each result by its *Time-to-Live* (TTL); although TTLs is not usually exposed to users, this information is propagated through recursive resolvers.

Once cached by recursive resolvers, cached results cannot removed; they can only be refreshed response by a new query and response after the TTL expires.

Some recursive resolvers discard long-lived cache entries after a configurable timeout. BIND defaults to dropping entries after 1 week [17], and Unbound after 1 day [28].

Operators select TTLs carefully. Content delivery networks (CDNs) often use DNS to steer users to different content servers. They select very short TTLs (60 seconds or less) to force clients to re-query frequently, providing opportunities to redirect clients with DNS in response to changes in load or server availability [30]. Alternatively, DNS data for top-level domains often has TTLs of hours or days. Such long TTLs reduce latency for clients (the reply can be reused immediately if it is in the cache of a recursive resolver) and reduce load on servers for commonly used top-level domains and slowly changing DNSSEC information.

3 DNS CACHING IN CONTROLLED EXPERIMENTS

To understand the role of caching at recursive resolvers in protection during failure of authoritative servers, we first must understand *how often are cache lifetimes (TTLs) honored.*

In the best-case scenario, authoritative DNS operators may expect clients to be able to reach domains under their zones even if their authoritative servers are unreachable, for as long as cached values in the recursives remain "valid" (*i.e.,* TTL not expired). Given the large variety of recursive implementations, we pose the following question: *from a user point-of-view, can we rely on recursives caching when authoritatives fail?*

To understand cache lifetimes in practice, we carry out controlled measurements from thousands of clients. These measurements determine how well caches work in the field, complementing our understanding of how open source implementations work from their source code. This study is important because operational software can vary and large deployments often use heavily customization or closed source implementations [48].

3.1 Potential Impediments to Caching

Although DNS records should logically be cached for the full TTL, a number of factors can shorten cache lifetimes in practice: caches are of limited size, caches may be flushed prematurely, and large resolvers may have fragmented caches. We briefly describe these factors here; understanding how often they occur motivates the measurements we carry out.

Caches are of limited size. Unbound, for example, defaults to a 4 MB limit, but the values are configurable. In practice, DNS results are small enough and caches large enough that cache sizes are usually not a limiting factor. Recursive resolvers may also override record TTLs, imposing either a minimum or maximum value [52].

Caches can be flushed explicitly (at the request of the cache operator), or accidentally on restart of the software or reboot of the machine running the cache.

Finally, some recursive resolvers handle very high request rates—consider a major ISP or public resolver [12, 29, 37]. Large recursive resolvers are often implemented as many separate recursives behind a load balancer or on IP anycast. In such cases the caches may be fragmented with each machine operating an independent cache, or they may share a cache of common names. In practice these may reduce the cache hit rate.

3.2 Measurement Design

To evaluate caching we use controlled experiments where we query from specific names to authoritative servers we run from thousands of RIPE Atlas sites. Our goal is to measure whether the TTL we define for the RRs of our controlled domain is honored across recursives.

Authoritative servers: we deploy two authoritatives that answer for our new domain name (cachetest.nl). We place the authoritatives on virtual machines in the same datacenter (Amazon EC2 in Frankfurt, Germany), each at a distinct unicast, IPv4 addresses. Each authoritative runs BIND 9.10.3. Since both authoritatives are in the same datacenter, they will have similar latencies to recursives, so we expect recursives to evenly distribute queries between both authoritative servers [27].

Vantage Points: We issue queries to our controlled domain from around 9k RIPE Atlas probes [39]. Atlas Probes are distributed across 3.3k ASes, with about one third hosting multiple *vantage points* (VPs). Atlas software causes each probe to issue queries to each of its local recursive resolvers, so our VPs are the tuple of probe and recursive. The result is that we have more than 15k VPs (Table 1).

Queries and Caching: We take several steps to ensure that caching does not interfere with queries. First, each *query* is for a

TTL	60	1800	3600	86400	3600-10min
Probes	9173	9216	8971	9150	9189
Probes (val.)	8725	8788	8549	8750	8772
Probes (disc.)	448	428	422	400	417
VPs	15330	15447	15052	15345	15397
Queries	94856	96095	93723	95780	191931
Answers	90525	91795	89470	91495	183388
Answer (val.)	90079	91461	89150	91172	182731
Answers (disc.)	446	334	323	323	657

Table 1: Caching baseline experiments [38].

name unique to the probe: each probe requests an AAAA record for {probeid}.cachetest.nl, where {probeid} is the probe's the unique identifier. Each *reply* is also customized. In the AAAA reply we encode three fields that are used to determine the effectiveness of caching (§3.4). Each IPv6 address in the answer is the concatenation of four values (in hex):

prefix is a fixed, 64-bit value (`fd0f:3897:faf7:a375`)
serial is a 8-bit value, incremented every 10 minutes (zone file rotation), allowing us to associate replies with specific query rounds
probeid is the unique Atlas probeID [40] encoded in 8 bits, to associate the query with the reply
ttl is a 16-bit value of the TTL value we configure per experiment

We increment the serial number in each AAAA record and reload the zone (with a new zone serial number), every 10 minutes. The serial number in each reply allows us to distinguish cached results from prior rounds from fresh data in this round.

Atlas DNS queries timeout after 5 seconds, reporting "no answer". We will see this occur in our emulated DDoS events.

For example, a VP with probeID 1414 shall send a DNS query for AAAA record for the domain name 1414.cachetest.nl, and should receive AAAA answer in the format `$PREFIX:1:586::3c`, where `$SERIAL=1`and `$TTL=60`.

We focus on DNS over UDP on IPv4, not TCP or IPv6. We use only IPv4 queries from Atlas Probes, and serve only IPv4 authoritatives, but the IPv6 may be used inside multi-level recursives. Our work could extend to cover other protocols, but we did not want to complicate analysis the orthogonal issue of protocol selection. We focus on DNS over UDP because it is by far the dominant transport protocol today (more than 97% of connections for .nl [50] and most Root DNS servers [16]).

Query Load: The query rate of our experiments is designed to explicitly test how queries intersect with TTL experimentation, and not to reproduce real-world traffic rates. Popular domains such as .com will be queried much more frequently than our query rates, so our results represent lower-bounds on caching. In §4 we examine caching rates with real-world names under .nl, testing a range of name popularities.

: **TTL**: TTL values vary significantly in DNS, with top-level domains typically using 1 day TTLs, while CDNs often use short TTLs of 1 or 5 minutes. Given this diversity of configurations, we explicitly design experiments that cover the range from 1 minute to 1 day (60 s and 86400 s TTLs). Thus, rather than trying to capture a single TTL that represents all possible configurations, we study a range of TTLs to explore the full range of caching behavior. §4 examines real-world traffic to provide a view of how well caching works with the distribution of TTLs seen in actual queries.

Representativeness of Atlas Locations and Software: It is well known that the global distribution of RIPE Atlas probes is uneven; Europe has far more than elsewhere [5, 6, 46]. Although quantitative data analysis might be generally affected by this distribution bias, our qualitative analysis, contributions and conclusions do not depend on the geographical location of probes.

Atlas probes use identical stub resolver software, but they are deployed in diverse locations (homes, businesses, universities) and so see a diverse set of recursives vendors and versions . Our study therefore represents Atlas "in the wild", and does not try to study specific software versions or vendors. Although we claim our study captures diverse recursive resolvers, we do not claim they are representative of a "typical" Internet client. It complements prior studies on caching by establishing what Atlas sees, an baseline needed when we study DDoS in §5.

3.3 Datasets

We carried out five experiments, varying the cache lifetime (TTL) and probing frequency from the VPs. Table 1 lists the parameters of experiments. In the first four measurements, the probing interval was fixed to 20 minutes, and TTL for each AAAA was set to 60, 1800, 3600 and 86400 seconds, all frequently used TTL values. For the fifth measurement we fixed the TTL value to 3600 seconds, and reduced the probing interval to 10 minutes to get better resolution of dynamics.

In each experiment, queries were sent from about 9k Atlas probes. We discard 400–448 of these ("probes (disc.)", about 4.4 to 4.9% of probes) that do not return an answer. Successful Atlas probes query multiple recursive resolvers, each a Vantage Point, so each experiment results in about 15k VPs. We also discard 323–657 answers ("answers (disc.)", about 3.5 to 4.9% of answers) because they report error codes (for example, SERVFAIL and REFUSED [21]), or they are referrals instead of the desired AAAA records [15]. (We provide more detail about referrals in Appendix A).

Overall, about 93–96k queries to cachetest.nl from the 9k probes at 20 minute pacing, and about double that with 10 minute pacing. Experiments last two to three hours, with no interference between experiments due to use of unique names, We ensure that experiments are isolated from each other. First, we space experiments about one day apart (details in RIPE [38]). Second, the IP addresses (and their records in cachetest.nl) of both authoritative name servers change in each experiment when we restart their VMs. Finally, we change the replies in the AAAA records, so we can detect any stale results (see §3.2).

3.4 TTL distribution: expected vs. observed

We next investigate how often recursive resolvers honor the full TTL provided by authoritative servers. Our goal is to classify the valid DNS answers from Table 1 into four categories, based on where the answer comes from, and where we *expect* it to come from:

AA answers expected and correctly from the authoritative
CC expected and correct from a recursive cache (cache hits)
AC answers from the authoritative, but expected to be from the recursive's cache (a cache miss)

TTL	60	1800	3600	86400	3600-10m
Answers (valid)	90079	91461	89150	91172	182731
1-answer VPs	38	51	49	35	17
Warm-up (AAi)	15292	15396	15003	15310	15380
Duplicates	25	23	25	22	23
Unique	15267	15373	14978	15288	15357
TTL as zone	14991	15046	14703	10618	15092
TTL altered	276	327	275	4670	265
AA	74435	21574	10230	681	11797
CC	235	29616	39472	51667	107760
CCdec.	4	5	1973	4045	9589
AC	37	24645	24091	23202	47262
TTL as zone	2	24584	23649	13487	43814
TTL altered	35	61	442	9715	3448
CA	42	179	305	277	515
CAdec.	7	3	21	29	65

Table 2: Valid DNS answers (expected/observed)

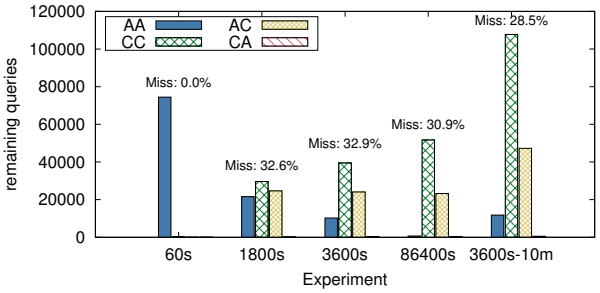


Figure 3: Classification of subsequent answers with warm cache

CA answers from a recursive's cache, but expected from the authoritative (an extended cache)

To determine if a query should be answered by the cache of the recursive, we track the state of prior queries and responses, and the estimated TTL. Tracking state is not hard since we know the initial TTL and all queries to the zone, and we encode the serial number and the TTL in the AAAA reply (§3.2).

Cold Caches and Rewriting TTLs: We first consider queries made against a cold cache (the first query of a unique name) to test how many recursives *override* the TTL. We know that this happens at some sites, such as at Amazon EC2, where their virtual machines (VMs) default recursive resolver caps all TTLs to 60 s [36].

Table 2 shows the results of our five experiments, in which we classify the valid answers from Table 1. Before classifying them, we first disregard VPs that had only one answer (1-answer VPs) since we cannot evaluate their caches status with one answer only (maximum 51 VPs out of 15,000 for the experiments). Then, we classify the remaining queries as Warm-up queries *AAi*, all of which are type AA (expected and answered by the authoritative server).

We see some duplicate responses; for these we use the timestamp of the very first *AAi* received. We then classify each unique *AAi* by comparing the TTL value returned by the recursive with the expected TTL that is encoded in the AAAA answer (fixed per experiment). The *TTL as zone* line counts the answers we expect to get, while *TTL altered* shows that a few hundred recursive resolvers alter the TTL. If these two values differ by more than 10%, we report TTL altered.

We see that the vast majority of recursives honor small TTLs, with only about 2% truncating the TTL (275 to 327 of about 15000, depending on the experiment's TTL). We and others (§7) see TTL truncation from multiple ASes. The exception is for queries with day-long TTLs (86400 s), where 4,670 queries (30%) have shortened TTLs. (Prior work also reported that many public resolvers refreshes at 1 day [51].) We conclude that wholesale TTL shortening does not occur for TTLs of an hour or less.

TTLs with Warm Cache: We next consider a warm cache—subsequent queries where we believe the recursive should have the prior answer cached and classify them according to the proposed categories (AA, CC, AC, and CC).

Figure 3 shows a histogram of this classifications (numbers shown on Table 2). We see that most answers we receive show expected caching behavior. For 60 s TTLs (the left bar), we expect no queries to be cached when we re-query 20 minutes (1200 s) later, and we see few cache hits (235 queries – CC row on Table 2 – which are due to TTL rewriting to values larger than 20min.). We see only a handful of CA-type replies, where we expect the authoritative to reply and the recursive does instead. We conclude that under normal operations (with authoritatives responding), recursive resolvers do not serve stale results (as has been proposed when the authoritative cannot be reached [19]).

For longer TTLs we see cache misses (AC responses) fractions of 28 to 33% ($AC/(\text{Answer\_}(valid) - (1-\text{Answers+ Warm-up}))$. Most of the AC answers did *not* alter the TTL (AC-over), *i.e.*, the cache miss was not due to TTL manipulations (Table 2). We do see 9,715 TTL modifications (about 42% of ACs) when the TTL is 1 day TTLs (86400 s). These TTL truncations are consistent with recursive resolvers that limit cache durations, such as caps of 7 days in BIND [17] and 1 in unbound [28], by default. (We provide more detail about TTL manipulations in Appendix B).

We conclude that DNS caches are fairly effective, with cache hits about 70% of the time. This estimate is likely a lower bound: we are the only users of our domain, and popular domains would see cache hits due to requests from other users. We only see TTL truncation for day-long TTLs. This result will help us understand the role of caching when authoritatives are under stress.

3.5 Public Recursives and Cache Fragmentation

Although we showed that most requests are cached as expected about 30% are not. We know that many DNS requests are served by public recursive resolvers today, several of which exist [1, 12, 29, 37]. We also know that public recursives often use anycast and load balancing [48] and that that can result in caches that are fragmented (not shared) across many servers. We next examine how many cache misses (type AC replies) are due to public recursives.

Although we control queriers and authoritative servers, there may be multiple levels of recursive resolvers in between. From Figure 1, we see the querier's first-hop recursive (*R1*) and the recursive that queries the authoritative (*Rn*). Fortunately, queries and replies are unique, so we can relate queries to the final recursive knowing the time (the query round) and the query source. For each query q, we extract the IP address of R_n and compare against a list of IP

TTL	60	1800	3600	86400	3600-10m
AC Answers	37	24645	24091	23202	47262
Public R_1	0	12000	11359	10869	21955
Google Public R_1	0	9693	9026	8585	17325
other Public R_1	0	2307	2333	2284	4630
Non-Public R_1	37	12645	12732	12333	25307
Google Public R_n	0	1196	1091	248	1708
other R_n	37	11449	11641	12085	23599

Table 3: AC answers public resolver classification.

addresses for 96 public recursives (Appendix C) we obtain from DuckDuckGo search for "public dns" done on 2018-01-15.

Table 3 reexamines the AC replies from Table 2. With the exception of the measurements with TTL of 60 s, nearly half of AC answers (cache misses) are from queries to public R_1 recursives, and about three-quarters of these are from Google's Public DNS. The other half of cache misses start at non-public recursives, but 10% of these eventually emerge from Google's DNS.

Besides identifying public recursives, we also see evidence of cache fragmentation in answers from caches (CC and CA). Sometimes we see serial numbers in consecutive answers decrease. For example, one VP reports serial numbers 1, 3, 3, 7, 3, 3, suggesting that it is querying different recursives, one with serial 3 and another with serial 7 in its cache. We show these occurrences in Table 2 as CCdec. and CAdec. With longer TTLs we see more cache fragmentation, with 4.5% of answers showing fragmentation with day-long TTLs.

From these observations we conclude that cache misses result from several causes: (1) use of load balancers or anycast where servers lack shared caches, (2) first-level recursives that do not cache and have multiple second-level recursives, and (3) caches may reset between the somewhat long probing interval (10 or 20 minutes). Causes (1) and (2) occur in public resolvers (confirmed by Google [12]) and account for about half of the cache misses in our measurements.

4 CACHING PRODUCTION ZONES

In §3 we show that about one-third of queries do not conform with caching expectations, based on controlled experiments to our test domain. (Results may be better for caches that prioritize popular names.) We next examine this question for specific records in .nl, the country code domain (ccTLD) for the Netherlands and the Root (.) DNS zone. With traffic from "the wild" and a measurement target used by millions, this section uses a domain popular enough to stay in-cache at recursives.

4.1 Requests at .nl's Authoritatives

We apply this methodology to data for .nl country-code top-level domain (ccTLD). We look specifically at the A-records for the name-servers of .nl, ns[1-5].dns.nl.

Methodology: We use passive observations of traffic to the .nl authoritative servers.

For each target name in the zone and source (some recursive server, identified by IP address), we build a timeseries of all requests and compute their interarrival time, Δ. Following the classification from §3.4, we label queries as: **AC** if $\Delta < TTL$, showing an unnecessary query to the authoritative; **AA** if $\Delta \geq TTL$, an expected or delayed cache refresh. (We do not see cache hits and so there are no **CC** events.)

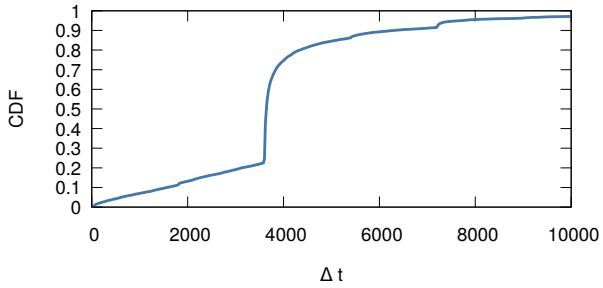


Figure 4: ECDF of the median Δt for recursives with at least 5 queries to ns1-ns5.dns.nl (TTL of 3600 s.)

Dataset: At the time of our analysis (February 2018) there were 8 authoritative servers for the .nl zone. We collect traffic for the 4 unicast and one anycast authoritative servers, and store the data in ENTRADA [55] for analysis.

Since our data for .nl is incomplete, and we know recursives will query all authoritatives over time [27], our analysis represents a conservative estimate of TTL violations—we expect to miss some CA-type queries from resolvers to non-monitored authoritatives.

We collect data for a period of six hours on 2018-02-22 starting at 12:00 UTC. We only evaluate recursives that sent at least five queries for our domains of interest, omitting infrequent recursives (they do not change results noticeably). We discard duplicate queries, for example, a few retransmissions (less than 0.01% of the total queries). In total, we consider more than 485k queries from 7,779 different recursives.

Results: Figure 4 shows the distribution of Δt that we observe in our measurements, reporting the median Δt for any resolver that sends at least 5 queries.

About 28% of queries are frequent, with an inter-arrival less than 10 s, and 32% of these are sent to multiple authoritatives. We believe these are due to recursives submitting queries in parallel to speed up replies (perhaps the "Happy Eyeballs" algorithm [45])

Since these closely-timed queries are not related to recursive caching, we exclude them from analysis. The remaining data is 348k queries from 7,703 different recursives.

The largest peak is at 3600 s, what was expected: the name was queried and cached for the full hour TTL, then the next request causes the name to be re-fetched. These queries are all of type AA.

The smaller peak around 1800 s, as well as queries with other times less than 3600 s, correspond to type AC-queries—queries that could have been supplied from the cache but were not. 22% of resolvers sent most of their queries within an time interval that is less than 3600 s or even more frequent. These AC queries occur because of TTL limiting, cache fragmentation, or other reasons that clear the cache.

4.2 Requests at the DNS Root

In this section we perform a similar analysis as for §4.1, in which we look into DNS queries received at all Root DNS servers (except

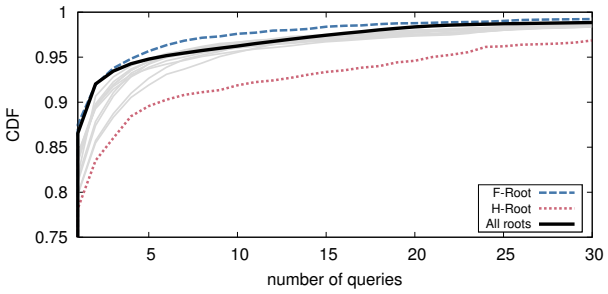


Figure 5: Distribution of the number of queries for the DS record of nl received for each recursive. Dataset: DNS-OARC DITL on 2017-04-12t00:00Z for 24 hours. All Root servers with similar distributions are shown in light-gray lines.

G-Root), and create a distribution of the number of queries received per source IP address (*i.e.,* per recursive).

In this analysis we use data from the DITL (*Day In The Life*) dataset of 2017, available at DNS-OARC [9]. We look at all DNS queries received for the DS record of the domain nl, received at the Root DNS servers along the entire day on April 12, 2017 (UTC). This dataset consists of queries from more than 70.3k unique recursives seen across all Root servers. Note that the DS record for nl has a TTL of 86400 seconds (24 hours). That is, in theory, one could expect to see just one query per recursive arriving at a given root letter, for the DS record of nl within the 24-hour interval.

Each line in Figure 5 shows the distribution of the total number of queries received at the Root servers from individual recursives asking for the DS record of nl. Besides F- and H-Root, the distribution is similar across all Root servers; these are plotted in light-gray lines. F-Root shows the "most friendly" behavior from recursives, where around 5% of them sent 5 or more queries for nl. As opposed to F, H-Root (dotted red line) shows the "worst" behavior from recursives, where more than 10% of them sent 5 or more queries for nl within the 24-hour period.

The solid black line in Figure 5 shows the distribution for all the queries across all Root servers. The majority (around 87%) of recursives does send only one query within the 24-hour interval. However, considering all Root servers, we see around 13% of recursives that have sent multiple queries. Note that the distributions shown in Figure 5 have (very) long tails, and we see up to more than 21.8k queries from a single recursive within the 24-hour period for the nl DS record; *i.e.,* roughly one query every 4 seconds from the same IP address for the same DS record.

Discussion: we conclude that measurements of popular domains within .nl (§4.1) and the Roots (§4.2) show that about 63% and 87% of recursives honor the full TTL, respectively. These results are roughly in-line with our observations with RIPE Atlas (§3).

5 THE CLIENT'S VIEW OF AUTHORITATIVES UNDER DDOS

We next use controlled experiments to evaluate how DDoS attacks at authoritative DNS servers impacts client experience. Our studies of caching in controlled experiments (§3) and passive observations (§4) have shown that caching often works, but not always—about 70% of controlled experiments and 30% of passive observations see full cache lifetimes. Since results of specific experiments vary, we sweep the space of attack intensities to understand the range of response from *complete* failure of authoritative servers, to *partial* failures.

5.1 Emulating DDoS

To emulate DDoS attacks we begin with the same test domain (cachetest.nl) we used for controlled experiments in §3. We run a normal DNS service for some time, querying from RIPE Atlas. After caches are warm, we then simulate a DDoS attack by dropping some fraction or all incoming DNS queries to each authoritative. (We drop incoming traffic randomly with Linux `iptables`. As such, packet drop is not biased towards any recursive.) After we begin dropping traffic, answers come either from caches at recursives or, for partial attacks, from a lucky query that passes through.

This emulation of DDoS captures traffic loss that occurs in DDoS attack as router queues overflow. This emulation is not perfect, since we simulate loss at the last hop-router, but in real DDoS attacks packets are often lost on access links near the target. Our emulation approximates this effect with one aggregate loss rate.

DDoS attacks are also accompanied by queueing delay, since buffers at and near the target are full. We do not model queueing delay, although we do observe latency increasing due to retries. In modern routers, queueing delay due to full router buffers should be less than the retry interval. In addition, observations during real-world DDoS events show that the few queries that are successful see response times that are not much higher than typical [23], suggesting that loss (and not delay) is the dominant effect of DDoS in practice. However, a study that adds queueing latency to the attack model is interesting future work.

5.2 Clients During Complete Authoritatives Failure

We first evaluate the worst-case scenario for a DNS operator: complete unreachability of all authoritative name servers. Our goal is to understand when and for how long caches cover such an outage.

Table 4 shows Experiments A, B, and C which simulate complete failure. In Experiment A, each VP makes only one query before the DDoS begins. In Experiment B we allow several queries to take place, and Experiment C allows several queries with a shorter TTL.

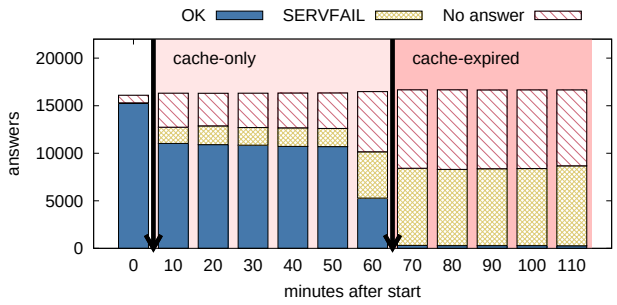
Caches Protect Some: We first consider Experiment A, with one query that warms the cache immediately followed by the attack. Figure 6a shows these responses over time, with the onset of the attack the first downward arrow between 0 and 10 minutes, and with the cache expired after the second downward arrow between 60 and 70 minutes. We see that after the DDoS starts but before the cache has fully expired (between the downward arrows) initially 30% and eventually 65% of queries fail with either no answer or a SERVFAIL error. While not good, this does mean that *35% to 70% of queries during the DDoS are successfully served from the cache.* By contrast, shortly after the cache expires, almost all queries fail (only 25 VPs or 0.2% of the total seem to provide stale answers).

Caches Fill at Different Times: In a more realistic scenario, VPs have filled their caches at different times. In Experiment A, caches are freshly filled and should last for a full hour after the start of attack. Experiment B is designed for the opposite and worst

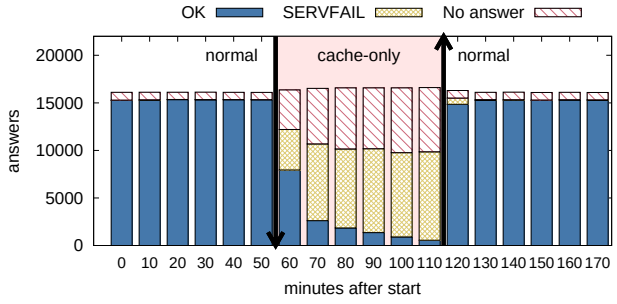
	Experiment Parameters						
	TTL in sec.	**DDoS start**	**DDoS dur.**	**queries before**	**total dur.**	**probe interval**	**failure**
A	3600	10	60	1	120	10	100% (both NSes)
B	3600	60	60	6	240	10	100% (both NSes)
C	1800	60	60	6	180	10	100% (both NSes)
D	1800	60	60	6	180	10	50% (one NS)
E	1800	60	60	6	180	10	50% (both NSes)
F	1800	60	60	6	180	10	75% (both NSes)
G	300	60	60	6	180	10	75% (both NSes)
H	1800	60	60	6	180	10	90% (both NSes)
I	60	60	60	6	180	10	90% (both NSes)

	Results					
	Total probes	**Valid probes**	**VPs**	**Queries**	**Total answers**	**Valid answers**
A	9224	8727	15339	136423	76619	76181
B	9237	8827	15528	357102	293881	292564
C	9261	8847	15578	258695	199185	198197
D	9139	8708	15332	286231	273716	272231
E	9153	8708	15320	285325	270179	268786
F	9141	8727	15325	278741	259009	257740
G	9206	8771	15481	274755	249958	249042
H	9226	8778	15486	269030	242725	241569
I	9224	8735	15388	253228	218831	217979

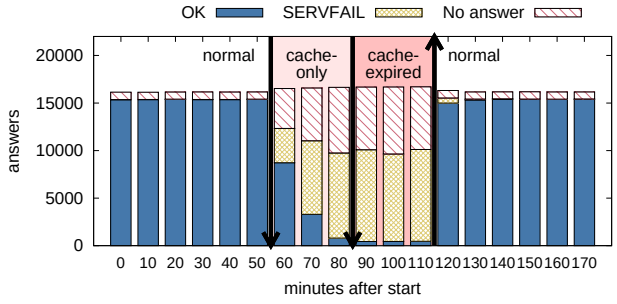
Table 4: DDoS emulation experiments [38]; DDoS start, durations and probe interval are given in minutes.



(a) Experiment A: 3600-10min-1down; arrows indicate DDoS start and cache expiration



(b) Experiment B: 3600-10min-1down-1up; arrows indicate DDoS start and recovery



(c) Experiment C: 1800-10min-1down-1up; arrows indicate DDoS start, cache expiration and recovery

Figure 6: Answers received during DDoS attacks.

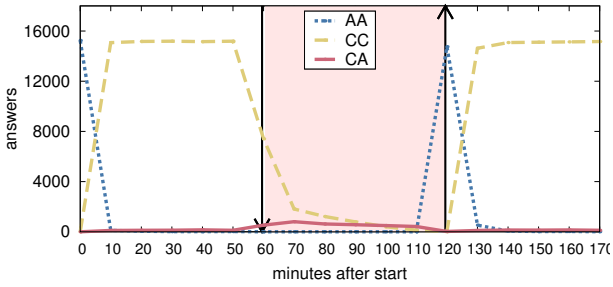


Figure 7: Timeseries of answers for Experiment B

case: we begin warming the cache one hour before the attack and query 6 times from each VP. Other parameters are the same, with the attack lasting for 60 minutes (also the cache duration), but then we restore the authoritatives to service.

Figure 6b shows the results of Experiment B. While about 50% of VPs are served from the cache in the first 10 minute round after the DDoS starts, the fraction served drops quickly and is at only about 3% one hour later. Three factors are in play here: most caches were filled 60 minutes before the attack and are timing out in the first round. While the timeout and query rounds are both 60 minutes apart, Atlas intentionally spreads queries out over 5 minutes, so we expect that some queries happen after 59 minutes and others 61 minutes.

Second, we know some large recursives have fragmented caches (§3.5), so we expect that some of the successes between times 70 and 110 minutes are due to caches that were filled between times 10 and 50 minutes. This can actually be seen in Figure 7, where we show a timeseries of the answers for Experiment B, where we see CC (correct cache responses) between times 60 and 90.

Third, we see an increase in the number of CA queries that are answered by the cache with expired TTLs (Figure 7). This increase is due to servers serving stale content [19].

Caches Eventually All Expire: Finally, we carry out a third emulation but with half the cache lifetime (1800 s or 30 minutes rather than the full hour). Figure 6c shows response over time. These results are similar to Experiment B, with rapid fall-off when the attack starts as caches age. After the attack has been underway for 30 minutes all caches must have expired and we see only a few (about 2.6%) residual successes.

5.3 Discussion of Complete Failures

Overall we see that *caching is partially successful in protecting during a DDoS*. With full, valid caches, half or more VPs get service. However, caches are filled at different times and expire, so an operator cannot count on a full cache duration for any customers, even for popular ("always in the cache") domains. The protection provided by caches depends on their state in the recursive resolver, something outside the operator's control. In addition, our evaluation of caching in §3 showed that caches will end early for some VPs.

Second, we were surprised that a tiny fraction of VPs are successful after all caches should have timed out (after the 80 minutes period in Experiment A, and between 90 and 110 minutes in Experiment C). These successes suggest an early deployment of "serve stale", something currently under review in the IETF [19] is to serve a previously known record beyond its TTL if authoritatives are unreachable, with the goal of improving resilience under DDoS. We investigated the Experiment A, where see that 1048 answers of the 1140 successes in the second half of the outage. These successes are from 471 VPs (and 215 recursives), most of them answered by OpenDNS and Google public DNS servers, suggesting experimentation not yet widespread. Out of these 1048 queries, 1031 return a TTL value equals to 0, as specified in the IETF stale draft [19].

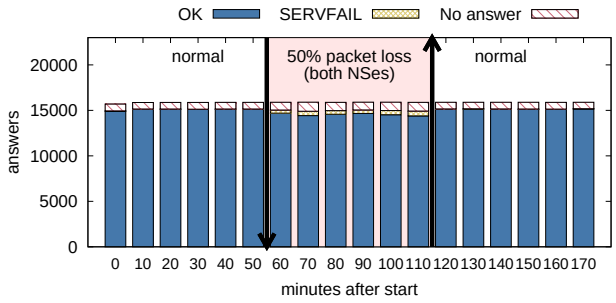
5.4 Client Reliability During Partial Authoritative Failure

The previous section examined DDoS attacks that result in complete failure of all authoritatives, but often DDoS attacks result in *partial failure*, with 50% or 90% packet loss at the authoritatives. (For example, consider the November 2015 DDoS attack on the DNS Root [23].) We next study experiments with partial failures, showing that *caching and retries together nearly fully protect 50% DDoS events, and protect half of VPs even during 90% events.*

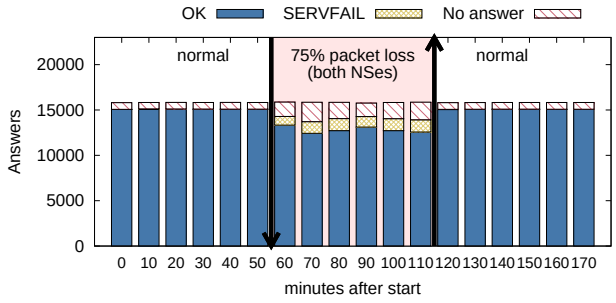
We carry out several Experiments D to I in Table 4. We follow the procedure outlined in §5.1, looking at the DDoS-driven loss rates of 50%, 75%, and 90% with TTLs of 1800 s, 300 s and 60 s. Graphs omitted due to space can be found in Appendix D.

Near-Full Protection from Caches During Moderate Attacks: We first consider Experiment E, a "mild" DDoS with 50% loss, with VP success over time in Figure 8a. In spite of a loss rate that would be crippling to TCP, nearly all VPs are successful in DNS. This success is due to two factors: first, we know that many clients are served from caches, as was shown in Experiment A with full loss (Figure 6a). Second, most recursives retry queries, so they recover from loss of a single packet and are able to provide an answer. Together, these mean that failures during the first 30 minutes of the event is 8.5%, slightly higher than the 4.8% fraction of failures before the DDoS. For this experiment, the TTL is 1800 s (30 minutes), so we might expect failures to increase halfway through the DDoS. We do not see any increase in failures because caching and retries are *synergistic*, a successful retried query will place the answer in a cache for a later query. The importance of this result is that *DNS can survive moderate-size attacks when caching is possible.* While a positive, retries do increase latency, something we study in §5.5.

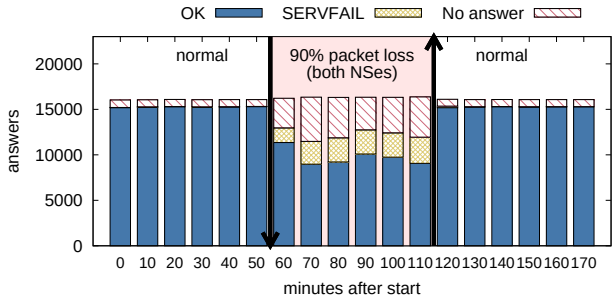
Attack Intensity Matters: While clients do quite well with 50% loss at all authoritatives, failures increase with the intensity of the attack.



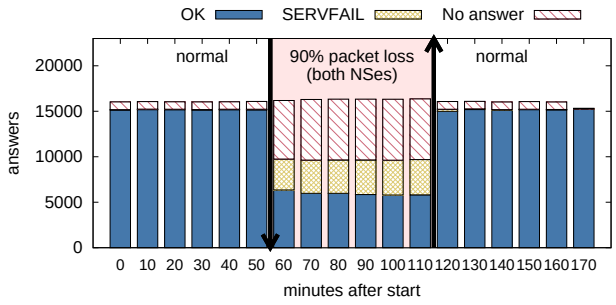
(a) Experiment E (1800-50p-10min): 50% packet loss



(b) Experiment F (1800-75p-10min): 75% packet loss



(c) Experiment H (1800-90p-10min): 90% packet loss



(d) Experiment I (60-90p-10min): 90% packet loss

Figure 8: Answers received during DDoS attacks; first and second vertical lines show start and end of DDoS.

Experiments F and H, shown in Figure 8b and Figure 8c increase the loss rate to 75% and 90%. We see the number of failures increases to about 19.0% with 75% loss and 40.3% with 90% loss. It is important to note that *roughly 60% the clients are still served even with 90% loss.*

We also see that this level of success is consistent over the entire hour-long DDoS event, even though the cache duration is only 30 minutes. This consistency confirms the importance of caching and retries in combination.

To verify the effects of this interaction, Experiment I changes the caching duration to 60 s, less than one round or probing. Comparing Experiment I in Figure 8d to H in Figure 8c, we see that the failure rate increases from 30% to about 63%. However, even with no caching, about 37% of queries still are answered, due to resolvers that serve stale content and recursives retries. We investigate retries in §6.

5.5 Client Latency During Partial Authoritative Failure

We showed that client reliability is higher than expected during failures (§5.4) due to a combination of caching and retries. We next consider client *latency*. Latency will increase during the DDoS because of retries and queueing delay, but we will show that latency increases less than one might expect due to caching.

To examine latency we return to Experiments D through I (Table 4), but look at latency (time to complete a query) rather than success. For these experiments clients timeout after 5 s.

Figures 9a to 9d show latency during each emulated DDoS scenario (experiments with figures omitted here are in Appendix D Latencies are not evenly distributed, since some requests get through immediately while others must be retried one or more times, so in addition to mean, we show 50, 75 and 90% quantiles to characterize the tail of the distribution.

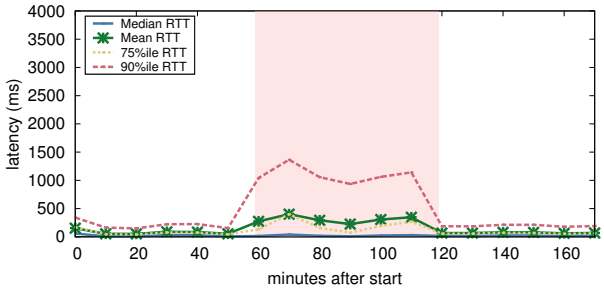
We emulate DDoS by dropping requests (§5.1) and, hence, latencies reflect retries and loss, but not queueing delay, underrepresenting latency in real-world attacks. However, their shape (some low latency and a few long) is consistent with and helps explain what has been seen in the past [23].

Beginning with Experiment E, the moderate attack in Figure 9a, we see *no* change to median latency. This result is consistent with many queries being handled by the cache, and half of those not handled by the cache getting through anyway. We do see higher latency in the 90%ile tail, reflecting successful retries. This tail also increases the mean some.

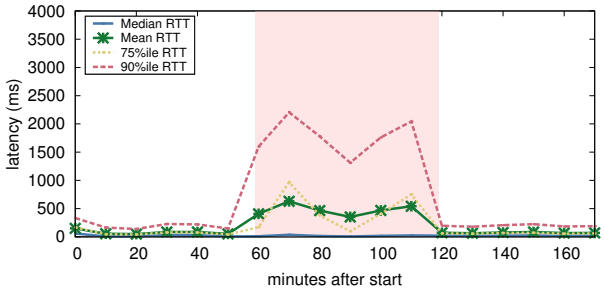
This trend increases in Experiment F in Figure 9b, where 75% of queries are lost. Now we see the 75%ile tail has increased, as has the number of unanswered queries, and the 90%ile is twice as long as in Experiment E.

We see the same latency in Experiment H with DDoS causing 90% loss. We set the timeouts to 5 s, so the larger attack results in more unsuccessful queries, but latency for successful queries is not much worse than with 75% loss. Median latency is still low due to cached replies.

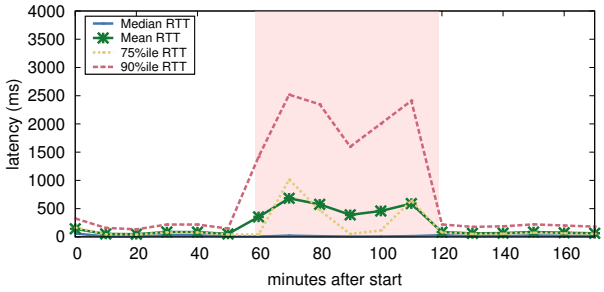
Finally, Experiment I greatly reduces opportunities for caching by reducing cache lifetime to one minute. Figure 9d shows that loss of caching increases median RTT and significantly increases the tail latency. Compared with Figure 9c (same packet loss ratio but



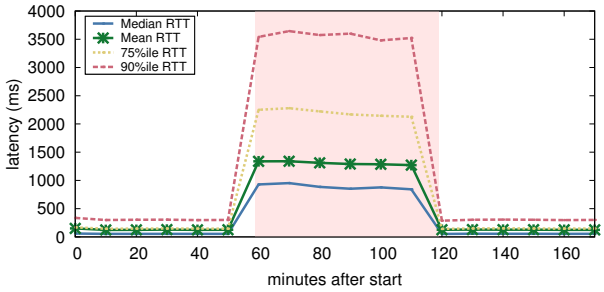
(a) Experiment E: 50% packet loss (1800 s TTL)



(b) Experiment F: 75% packet loss (1800 s TTL)



(c) Experiment H: 90% packet loss(1800 s TTL)



(d) Experiment I: 90% packet loss (60 s TTL)

Figure 9: Latency results; Shaded area indicates the interval of an ongoing DDoS attack.

1800 s TTL), we can clearly see the benefits of caching in terms of latency (in addition to reliability): a half-hour TTL value reduced the latency from 1300 ms to 390 ms. Longer TTLs also help reduce tail latency relative to shorter TTLs (compare, for example, the 90%ile RTT in Experiments I vs. H in Figure 9).

Summary: DDoS effects often increase client latency. For moderate attacks, increased latency is seen only by a few "unlucky" clients whose do not see a full cache and whose queries are lost. Caching has an important role in reducing latency during DDoS, but while it can often mitigate most reliability problems, it cannot avoid latency penalties for all VPs. Even when caching is not available, roughly 40% of clients get an answer, either by serving stale or retries as we investigate next.

6 THE AUTHORITATIVE'S PERSPECTIVE

Results of partial DDoS events (§5.4) show that DNS is surprisingly reliable—even with a DDoS resulting in 90% packet loss and lasting longer than the cache timeout, more than half of VPs get answers with 30 minute caches (Figure 8c), and about 40% of VPs get answers (Figure 8d) even with minimal duration caches. These results are due to a combination of caching and retries. We next examine this from the perspective of the authoritative server.

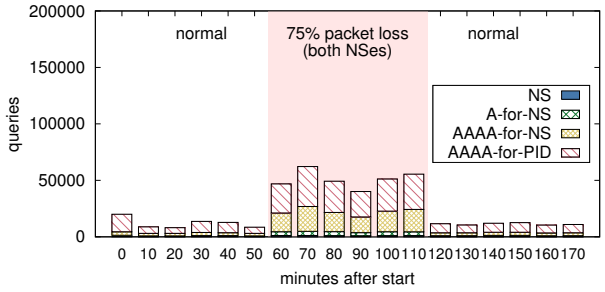
6.1 Recursive-Authoritative Traffic during a DDoS

We first ask: are retries by recursive resolvers responsible for the success rates observed in §5.4? To investigate this question, we return the partial DDoS experiments and look at how many queries are sent to the authoritative servers. We measure queries before they are dropped by our simulated DDoS. Recursives must make multiple queries to resolve a name. We break out each type of query: for the nameserver (NS), the nameserver's IPv4 and v6 addresses (A-for-NS and AAAA-for-NS), and finally the desired query (AAAA-for-PID). Note that the authoritative is IPv4 only, so AAAA-for-NS is non-existent and subject to negative caching, while the other records exist and use regular caching.

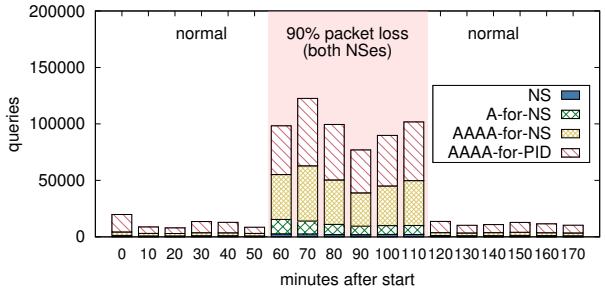
We begin with the DDoS causing 75% loss in Figure 10a. For this experiment, we observe 18,407 unique IP addresses of recursives (R_n) querying for AAAA records directly to our authoritatives. During the DDoS, queries increase by about 3.5×. We expect 4 trials, since the expected number of tries until success with loss rate p is $(1-p)^{-1}$. For this scenario, results are cached for up to 30 minutes, so successful queries are reused in recursive caches. This increase occurs both for the target AAAA record, and also for the non-existent AAAA-for-NS records. Negative caching for our zone is configured to 60 s, making caching of NXDOMAINs for AAAA-for-NS less effective than positive caches.

The offered load on the server increases further with more loss (90%), as shown in Experiment H (Figure 10b). The higher loss rate results in a much higher offered load on the server, average 8.2× normal.

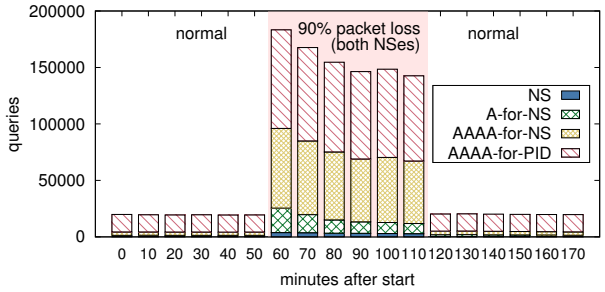
Finally, in Figure 10c we reduce the effects of caching at a 90% DDoS and with a TTL of 60 s. Here we see also about 8.1× more queries at the server before the attack. Comparing this case to Experiment H, caching reduces the offered load on the server by about 40%.



(a) Experiment F: 1800-75p-10min, 75% packet loss



(b) Experiment H: 1800-90p-10min, 90% packet loss



(c) Experiment I: 60-90p-10min, 90% packet loss

Figure 10: Number of received queries by the authoritative servers. Shaded area indicates the interval of an ongoing DDoS attack.

Implications: The implication of this analysis is that legitimate clients "hammer" with retries the already-stressed server during a DDoS. For clients, retries are important to get reliability; and each client independently chooses to retry.

The server is already under stress due to the DDoS, so these retries add to that stress. However, the DDoS traffic is almost certainly much larger than the retried of legitimate traffic. (A server experiencing a volumetric attack causing 90% loss must be receiving 10× its capacity. Regular traffic is a small fraction of normal capacity, so even 4× regular is still much less than the attack traffic.) The multiplier for retried legitimate traffic depends on the implementations stub and recursive resolver, as well as application-level retries and defection (users hitting reload in their browser, and later giving up). Our experiment omits application-level retries and likely gives a

lower bound. We next examine specific recursive implementations to see their behavior.

6.2 Sources of Retries: Software and Multi-level Recursives

Experiments in the prior section showed that recursive resolvers "hammer" authoritatives when queries are dropped. We reexamine DNS software (since 2012 [56]), and additionally show deployments amplify retries.

Recursive Software: Prior work showed that recursive servers retry many times when an authoritative is unresponsive [56], with evaluation of BIND 9.7 and 9.8, DNSCache, Unbound, WindowsDNS and PowerDNS. We studied retries in BIND 9.10.3 and Unbound 1.5.8 to quantify the number of retries. Examining only requests for AAAA records, we see that normal requests with a responsive authoritative ask for the AAAA records for all authoritatives and the target name (3 total requests when there are 2 authoritatives). When all authoritatives are unavailable, we see about 7× more requests before the recursives time out. (Exact numbers vary in different runs, but typically each request is made 6 or 7 times.) Such retries are appropriate, provided they are paced (both use exponential backoff), they explain part of the increase in legitimate traffic during DDoS events. Full data is in Appendix E.

Recursive Deployment: Another source of extra retries is complex recursive deployments. We showed that operators of large recursives often use complex, multi-level resolution infrastructure (§3.5). This infrastructure can amplify the number of retries during reachability problems at authoritatives.

To quantify amplification, we count both the number of *Rn* recursives and AAAA queries for each probe ID reaching our authoritatives. Figure 11 show the results for Experiment I. These values represent the amplification in two ways: during stress, more *Rn* recursives will be used for each probe ID and these *Rn* will generate more queries to the already stressed authoritatives. As the figures show, the median number of *Rn* recursives employed doubles (from 1 to 2) during the DDoS event, as does the 90%ile (from 2 to 4). The maximum rises to 39. The number of queries for each probe ID grows more than 3×, from 2 to 7. Worse, the 90%ile grows more than 6× (3 queries to 18). The maximum grows 53.5×, reaching up to 286 queries for one single probe ID. This value, however, is a lower bound, given there are a large number of A and AAAA queries that ask for NS records and not the probe ID (AAAA and A-for NS in Figure 10).

We can also look at the aggregate effects of retries created by the complex recursive infrastructure. Figure 12 shows the timeseries of unique IP addresses of R_n observed at the authoritatives. Before the DDoS period, for Experiment I with TTL of 60 s, we see a constant number of recursives reaching our authoritatives; *i.e.,* all queries should be answered by authoritatives (no caching at this TTL value). For experiments F and H, both with TTL of 1800 s, the number of recursives reaching our authoritative oscillates before the DDoS; peaks are observed when caches expire as expected.

During the DDoS we observe a similar behavior for all three experiments in Figure 12: as packets are dropped at the authoritative (at rates of 75, 90 and 90% for F, H, and I respectively) we see an increase on the number of *Rn* recursives querying our authoritatives; for experiments F and H we see drops when caching

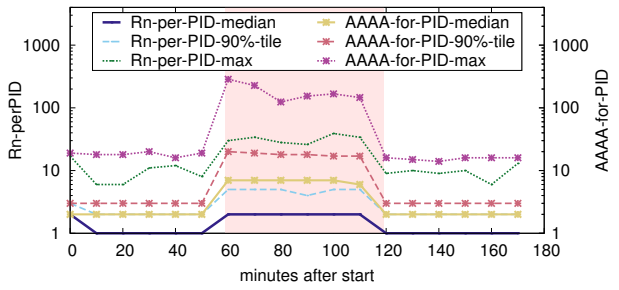


Figure 11: *Rn* recursives and AAAA queries used in Experiment I, normalized by the number of probe IDs.

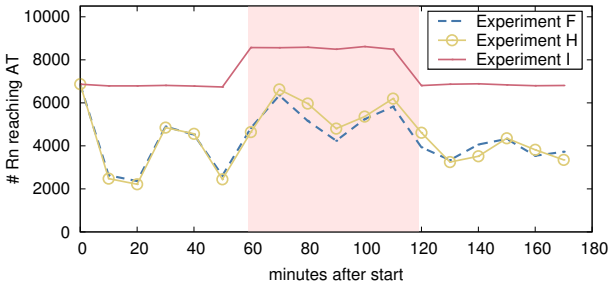


Figure 12: Unique *Rn* recursives addresses observed at authoritatives

is expected, but not for experiment I. The reason for this behavior is that the underlying layer of recursives starts forwarding queries to other recursives, which is amplified in the end. (We show this behavior for an individual probe in Appendix F, where we observe the growth in the number of queries received at the authoritatives and the number of recursives used.)

Most complex resolution infrastructures are proprietary (as far as we know only one study has examined them [48]), so we cannot make recommendations about how large recursive resolvers ought to behave. We suggest that the aggregate traffic of large recursive resolvers should strive to be within a constant factor of single recursives, perhaps a factor of 4. We also encourage additional study of large recursive resolvers, and their operators to share information about their behavior.

7 RELATED WORK

Caching by Recursives: Several groups have shown that DNS caching can be imperfect. Hao and Wang analyzed the impact of nonce domains on DNS recursive's caches [13]. Using two weeks of data from two universities they showed that filtering one-time domains improves cache hit rates. In two studies, Pang *et al.* [31, 32] reported that web clients and local recursives do not always honor TTL values provided by authoritatives. Almeida *et al.* [2] analyzed DNS traces of a mobile operator, and used a mobile application to see TTLS in practice. They find that most domains have short TTLs (less than 60 s), and report and evidence of TTL manipulation by recursives. Schomp *et al.* [48] demonstrate widespread use of

multi-level recursives by large operators, as well as TTL manipulation. Our work builds on this prior work, examining caching and TTL manipulation systematically and considering its effects on resilience.

DNS client behavior: Yu *et al.* investigated how stubs and recursives select authoritative servers, and were the first to demonstrate the large number of retries when all authoritatives are unavailable [56]. We also investigated how recursives select authoritative servers in the wild and found that recursives tend to prefer authoritatives with shorter latency, but query all authoritatives for diversity [27]. We confirm Yu's work and focus on authoritative selection during DDoS from several perspectives.

Authoritatives during DDoS: We investigated how the Root DNS service behaved during the Nov. 2015 DDoS attacks [23]. This report focuses on the interactions of IP anycast and both latency and reachability, as seen from RIPE Atlas. Rather than look at aggregate behavior and anycast, our methodology here examines how clients interact with their recursive resolvers, while this prior work focused on authoritatives only, bypassing recursives. In addition, here we have full access to clients and authoritatives traffic during our experiments, and we evaluate DDoS with controlled loss rates. The prior study has incomplete data and focuses on specific results of two events. These differences stem from their study of natural experiments from real-world events and our controlled experiments.

8 IMPLICATIONS

We evaluated DNS resilience, showing that caches and retries can mitigate much of the harm from a DDoS attack, provided the cache is full and some requests can get to authoritative servers. The key implication of our study is to explain differences in the outcome of recent DDoS attacks.

Recent DDoS attacks on DNS services have seen very different outcomes for users. The Root Server System was a target in Nov. 2015 [41] and June 2016 [42]. The DNS Root has 13 *letters*, each an authoritative "server" implemented with some or many IP anycast instances. Analysis of these DDoS events showed that their effects were uneven across letters: for some, most or all anycast instances showed high loss, while other letters showed little or no loss [23]. However, the Root Operators state "There are no known reports of end-user visible error conditions during, and as a result of, this incident. Because the DNS protocol is designed to cope with partial reachability…" [41].

In Oct. 2016, a much larger attack was directed at Dyn, a provider of DNS service for many second-level domains [14]. Although Dyn has a capable infrastructure and immediately took steps to address service problems, there were reports of user-visible service disruption in the technical and even popular press [34]. Reports describe intermittent failure of prominent websites including "Twitter, Netflix, Spotify, Airbnb, Reddit, Etsy, SoundCloud and The New York Times", each a direct or indirect customer of Dyn at the time.

Our work helps explain these very different outcomes. The Root DNS saw few or no user-visible problems because data in the root zone is cachable for a day or more, and because multiple letters and many anycast instances were continuously available. (All measurements in this paragraph are as of 2018-05-22.) Records in the root zone have TTLs of 1 to 6 days, and www.root-servers.org reports 922 anycast instances operating across the 13 authoritative servers. Dyn also operates a large infrastructure (https://dyn.com/dns/network-map/ reports 20 "facilities"), and faced a larger attack (reports of 1.2 Tb/s [47], compared to estimates of 35 Gb/s for the Nov. 2015 root attack [23]). But a key difference is *all* of the Dyn's customers listed above use DNS-based CDNs (for a description, see [7]) with multiple, Dyn-hosted DNS components with TTLs that range from 120 to 300 s.

In addition to explaining the effects, our experiments help get to the root causes behind these outcomes. Users of the Root benefited from caching and saw performance like Experiment E (Figure 8a), because root contents (TLDs like .com and country codes) are popular and certainly cached in recursives, and because some root letters were always available to refresh caches (either through a successful normal query, or a retry). By contrast, users requiring domains with very short TTLs (like the websites that had problems) receive performance more like Experiment I (Figure 8d) or Experiment C (Figure 6c). Even when some requests succeed an cache a popular name, short TTLs cause caches to clear quickly.

This example shows the importance of DNS's multiple methods of resilience (caching, retries, and at least some availability at one authoritative). It suggests that CDN operators may wish to consider longer timeouts to allow caching to help and give DNS operators deploy defenses. Experiment H suggests 30 minutes, Figure 8c.

Configuring short TTLs serves a role in CDNs that use DNS to direct clients to different application-level servers. Short TTLs allow for re-provisioning during DDoS attacks on web servers, but that leaves DNS servers vulnerable. This tension suggests traffic scrubbing by routing changes with long DNS TTLs may be preferred to short DNS TTLs, so that both layers can be robust. However, the complexity of interactions between DNS at multiple levels and CDNs suggests that more study is needed before recommending specific settings.

Finally, this evaluation helps complete our picture of DNS latency and reliability for DNS services that may consist of multiple authoritatives, some or all using IP anycast with multiple sites. To minimize latency, prior work has shown a single authoritative using IP anycast should maximize geographic dispersion of sites [46]. The latency of an overall DNS service with *multiple* authoritatives can be limited by the one with largest latency [27]. Prior work about resilience to DDoS attack has shown that individual IP anycast sites will suffer under DDoS as a function of the attack traffic that site receives relative to its capacity [23]. We show that the overall reliance of a DNS service composed of multiple authoritatives using IP anycast tends to be as resilient as the *strongest* individual authoritative. The reason for these opposite results is that, in both cases, recursive resolvers will try *all* authoritatives of a given service. For latency, they will sometimes choose a distant authoritative, but for resilience, they will continue until they find the most available authoritative.

9 CONCLUSIONS

This paper represents the first study of how the DNS resolution system behaves when authoritative servers are under DDoS attack. Caching and retries at recursive resolvers are key factors in this behavior. We show that together, caching and retries by recursive resolvers greatly improve the resilience of the DNS as a whole. In

fact, they can largely cover over partial DDoS attacks for many users—even with a DDoS resulting in 90% packet loss and lasting longer than the cache timeout, more than half of VPs get answers with 30 minute caches (Figure 8c), and about 40% of VPs get answers (Figure 8d) even with minimal duration caches.

The primary cost of DDoS for users can be greater latency, but even this penalty is uneven across users, with a few getting much greater latency while some see no or little change. Finally, we show that one result retries is that traffic from legitimate users to authoritatives greatly increases (up to 8×) during service interruption, and that this effect is magnified by complex, multi-layer recursive resolver systems. The key outcome of work is to quantify the importance of caching and retries in recursives to resilience, encouraging use of at least moderate TTLs wherever possible.

Acknowledgments

The authors would like to thank Jelte Jansen, Benno Overeinder, Marc Groeneweg, Wes Hardaker, Duanne Wessels, Warren Kumari, Stéphane Bortzmeyer, Maarten Aertsen, Paul Hoffman, our shepherd Mark Allman, and the anonymous IMC reviewers for their valuable comments on paper drafts.

This research has been partially supported by measurements obtained from RIPE Atlas, an open measurements platform operated by RIPE NCC, as well as by the DITL measurement data made available by DNS-OARC.

Giovane C. M. Moura, Moritz Müller, and Marco Davids developed this work as part of the SAND project (http://www.sand-project.nl).

John Heidemann's research is partially sponsored by the Air Force Research Laboratory and the Department of Homeland Security under agreements number FA8750-17-2-0280 and FA8750-17-2-0096. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

REFERENCES

[1] 1.1.1.1. 2018. The Internet's Fastest, Privacy-First DNS Resolver. https://1.1.1.1/. https://1.1.1.1/

[2] Mario Almeida, Alessandro Finamore, Diego Perino, Narseo Vallina-Rodriguez, and Matteo Varvello. 2017. Dissecting DNS Stakeholders in Mobile Networks. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT '17)*. ACM, New York, NY, USA, 28–34. https://doi.org/10.1145/3143361.3143375

[3] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. 2017. Understanding the Mirai Botnet. In *Proceedings of the 26th USENIX Security Symposium*. USENIX, Vancouver, BC, Canada, 1093–1110. https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-antonakakis.pdf

[4] Arbor Networks. 2012. *Worldwide Infrastructure Security Report*. Technical Report 2012 Volume VIII. Arbor Networks. http://www.arbornetworks.com/resources/infrastructure-security-report

[5] Vaibhav Bajpai, , Steffie Eravuchira, Jürgen Schönwälder, Robert Kisteleki, and Emile Aben. 2017. Vantage Point Selection for IPv6 Measurements: Benefits and Limitations of RIPE Atlas Tags. In *IFIP/IEEE International Symposium on Integrated Network Management (IM 2017)*. Lisbon, Portugal.

[6] Vaibhav Bajpai, Steffie Jacob Eravuchira, and Jürgen Schönwälder. 2015. Lessons Learned from using the RIPE Atlas Platform for Measurement Research. *SIGCOMM Comput. Commun. Rev.* 45, 3 (July 2015), 35–42. http://www.sigcomm.org/sites/default/files/ccr/papers/2015/July/0000000-0000005.pdf

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

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

[9] DNS OARC. 2018. DITL Traces and Analysis. https://www.dns-oarc.net/index.php/oarc/data/ditl/2018.

[10] R. Elz and R. Bush. 1997. Clarifications to the DNS Specification. RFC 2181 (Proposed Standard). , 14 pages. https://doi.org/10.17487/RFC2181 Updated by RFCs 4035, 2535, 4343, 4033, 4034, 5452.

[11] R. Elz, R. Bush, S. Bradner, and M. Patton. 1997. Selection and Operation of Secondary DNS Servers. RFC 2182 (Best Current Practice). , 11 pages. https://doi.org/10.17487/RFC2182

[12] Google. 2018. Public DNS. https://developers.google.com/speed/public-dns/. https://developers.google.com/speed/public-dns/

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

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

[15] Paul Hoffman, Andrew Sullivan, and K. Fujiwara. 2018. DNS Terminology. Internet Draft. https://datatracker.ietf.org/doc/draft-ietf-dnsop-terminology-bis/?include_text=1

[16] ICANN. 2014. RSSAC002: RSSAC Advisory on Measurements of the Root Server System. https://www.icann.org/en/system/files/files/rssac-002-measurements-root-20nov14-en.pdf.

[17] ISC BIND. 2018. Chapter 6. BIND 9 Configuration Reference. https://ftp.isc.org/isc/bind9/cur/9.10/doc/arm/Bv9ARM.ch06.html.

[18] Sam Kottler. 2018. February 28th DDoS Incident Report | Github Engineering. . https://githubengineering.com/ddos-incident-report/.

[19] D. Lawrence and W. Kumari. 2017. Serving Stale Data to Improve DNS Resiliency-02. Internet Draft. https://www.ietf.org/archive/id/draft-tale-dnsop-serve-stale-02.txt

[20] P.V. Mockapetris. 1987. Domain names - concepts and facilities. RFC 1034 (Internet Standard). , 55 pages. https://doi.org/10.17487/RFC1034 Updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592, 5936, 8020.

[21] P.V. Mockapetris. 1987. Domain names - implementation and specification. RFC 1035 (Internet Standard). , 55 pages. https://doi.org/10.17487/RFC1035 Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2673, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604, 7766.

[22] Carlos Morales. 2018. February 28th DDoS Incident Report | Github EngineeringNETSCOUT Arbor Confirms 1.7 Tbps DDoS Attack; The Terabit Attack Era Is Upon Us. https://www.arbornetworks.com/blog/asert/netscout-arbor-confirms-1-7-tbps-ddos-attack-terabit-attack-era-upon-us/.

[23] Giovane C. M. Moura, Ricardo de O. Schmidt, John Heidemann, Wouter B. de Vries, Moritz Müller, Lan Wei, and Christian Hesselman. 2016. Anycast vs. DDoS: Evaluating the November 2015 Root DNS Event. In *Proceedings of the ACM Internet Measurement Conference*. https://doi.org/10.1145/2987443.2987446

[24] Giovane C. M. Moura, John Heidemann, Moritz Müller, Ricardo de O. Schmidt, and Marco Davids. 2018. Datasets from "When the Dike Breaks: Dissecting DNS Defenses During DDoS". (May 2018). Web page https://ant.isi.edu/datasets/dns/Moura18a_data.

[25] Giovane C. M. Moura, John Heidemann, Moritz Müller, Ricardo de O. Schmidt, and Marco Davids. 2018. *When the Dike Breaks: Dissecting DNS Defenses During DDoS (extended)*. Technical Report ISI-TR-725b. USC/Information Sciences Institute. https://www.isi.edu/%7ejohnh/PAPERS/Moura18a.html (updated Sept. 2018).

[26] Giovane C. M. Moura, John Heidemann, Moritz Müller, Ricardo de O. Schmidt, and Marco Davids. 2018. When the Dike Breaks: Dissecting DNS Defenses During DDoS (extended). In *Proceedings of the ACM Internet Measurement Conference*. https://www.isi.edu/%7ejohnh/PAPERS/Moura18b.html

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

[28] NL Netlabs. 2018. NL Netlabs Documentation - Unbound - undbound.conf.5. https://nlnetlabs.nl/documentation/unbound/unbound.conf/.

[29] OpenDNS. 2018. Setup Guide: OpenDNS. https://www.opendns.com/setupguide/. https://www.opendns.com/setupguide

[30] Jianping Pan, Y Thomas Hou, and Bo Li. 2003. An overview of DNS-based server selections in content distribution networks. *Computer Networks* 43, 6 (2003), 695–711.

[31] Jeffrey Pang, Aditya Akella, Anees Shaikh, Balachander Krishnamurthy, and Srinivasan Seshan. 2004. On the Responsiveness of DNS-based Network Control. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement (IMC '04)*. ACM, New York, NY, USA, 21–26. https://doi.org/10.1145/1028788.1028792

[32] Jeffrey Pang, James Hendricks, Aditya Akella, Roberto De Prisco, Bruce Maggs, and Srinivasan Seshan. 2004. Availability, Usage, and Deployment Characteristics of the Domain Name System. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement (IMC '04)*. ACM, New York, NY, USA, 1–14. https://doi.org/10.1145/1028788.1028790

[33] Paul Vixie and Gerry Sneeringer and Mark Schleifer. 2002. Events of 21-Oct-2002. http://c.root-servers.org/october21.txt.

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

[35] Nicole Perlroth. 2016. Tally of Cyber Extortion Attacks on Tech Companies Grows. New York Times Bits Blog, http://bits.blogs.nytimes.com/2014/06/19/tally-of-cyber-extortion-attacks-on-tech-companies-grows/.
[36] Alec Peterson. 2017. EC2 resolver changing TTL on DNS answers? Post on the DNS-OARC dns-operations mailing list, https://lists.dns-oarc.net/pipermail/dns-operations/2017-November/017043.html.
[37] Quad9. 2018. Quad9 | Internet Security & Privacy In a Few Easy Steps. https://quad9.net.
[38] RIPE NCC. 2017. RIPE Atlas Measurement IDS. https://atlas.ripe.net/measurements/ID. ID is the experiment ID: TTL60: 10443671, TTL1800: 10507676, TTL3600: 10536725, TTL86400: 10579327, TTL3600-10min: 10581463, A:10859822, B: 11102436, C :11221270, D:11804500, E: 11831403, F: 11831403, G: 12131707, H:12177478 , I: 12209843.
[39] RIPE NCC Staff. 2015. RIPE Atlas: A Global Internet Measurement Network. *Internet Protocol Journal (IPJ)* 18, 3 (Sep 2015), 2–26.
[40] RIPE Network Coordination Centre. 2018. RIPE Atlas - Raw data structure documentations,https://atlas.ripe.net/docs/data_struct/.
[41] Root Server Operators. 2015. Events of 2015-11-30. http://root-servers.org/news/events-of-20151130.txt.
[42] Root Server Operators. 2016. *Events of 2016-06-25.* Technical Report. Root Server Operators. http://www.root-servers.org/news/events-of-20160625.txt
[43] Root Server Operators. 2017. Root DNS. http://root-servers.org/.
[44] José Jair Santanna, Roland van Rijswijk-Deij, Rick Hofstede, Anna Sperotto, Mark Wierbosch, Lisandro Zambenedetti Granville, and Aiko Pras. 2015. Booters—An Analysis of DDoS-as-a-Service Attacks. In *Proceedings of the 14th IFIP/IEEE Interatinoal Symposium on Integrated Network Management.* IFIP, Ottowa, Canada.
[45] D. Schinazi and T. Pauly. 2017. *Happy Eyeballs Version 2:Better Connectivity Using Concurrency.* RFC 8305. Internet Request For Comments. https://doi.org/10.17487/RFC8305
[46] Ricardo de O. Schmidt, John Heidemann, and Jan Harm Kuipers. 2017. Anycast Latency: How Many Sites Are Enough?. In *Proceedings of the Passive and Active Measurement Workshop.* Springer, Sydney, Australia, 188–200. http://www.isi.edu/%7ejohnh/PAPERS/Schmidt17a.html
[47] Bruce Schneier. 2016. Lessons From the Dyn DDoS Attack. blog https://www.schneier.com/essays/archives/2016/11/lessons_from_the_dyn.html.
[48] Kyle Schomp, Tom Callahan, Michael Rabinovich, and Mark Allman. 2013. On measuring the client-side DNS infrastructure. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference.* ACM, 77–90.
[49] Somini Sengupta. 2012. After Threats, No Signs of Attack by Hackers. *New York Times* (Apr. 1 2012), A1. http://www.nytimes.com/2012/04/01/technology/no-signs-of-attack-on-internet.html
[50] SIDN Labs. 2017. .nl stats and data. http://stats.sidnlabs.nl.
[51] Matthew Thomas and Duane Wessels. 2015. A study of caching behavior with respect to root server TTLs. DNS-OARC. https://indico.dns-oarc.net/event/24/contributions/374/
[52] Unbound. 2018. Unbound Documentation. https://www.unbound.net/documentation/unbound.conf.html.
[53] Lan Wei and John Heidemann. 2017. Does Anycast Hang up on You?. In *IEEE International Workshop on Traffic Monitoring and Analysis (Dublin, Ireland.*
[54] Weinberg, M., Wessels, D. 2016. Review and analysis of attack traffic against A-root and J-root on November 30 and December 1, 2015. In: DNS OARC 24 – Buenos Aires, Argentina. https://indico.dns-oarc.net/event/22/session/4/contribution/7.
[55] Maarten Wullink, Giovane CM Moura, Moritz Müller, and Cristian Hesselman. 2016. ENTRADA: A high-performance network traffic data streaming warehouse. In *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP.* IEEE, 913–918.
[56] Yingdi Yu, Duane Wessels, Matt Larson, and Lixia Zhang. 2012. Authority Server Selection in DNS Caching Resolvers. *SIGCOMM Comput. Commun. Rev.* 42, 2 (March 2012), 80–86. https://doi.org/10.1145/2185376.2185387

A WHICH TTL AFFECTS THE CACHE: REFERRALS OR ANSWERS?

Cache estimations in §3 and §4 assumes each record has a clear TTL. However, glue records bridge zones that are in different authoritative servers and can result in two different TTL values for a record. This section examines which TTL takes precedence.

A.1 The Problem of Glue Records

DNS employs *glue* records to identify name servers that are in the served zone itself. For example, the Canadian national domain .ca is served from ca-servers.ca, but to look up ca-servers.ca, one must know where .ca is. To break this cycle, records to identify these severs ("glue records") are placed in the parent zone (the root, in this case). With these records in two places, they may have different TTLs. We next evaluate which TTL takes priority in this case.

If we ask the Root DNS servers for the NS records of .ca, we obtain the answer shown in Listing 1:

```
$ dig ns ca @b.root-servers.net
;; AUTHORITY SECTION:
ca.     172800  IN      NS      any.ca-servers.ca.
ca.     172800  IN      NS      j.ca-servers.ca.
ca.     172800  IN      NS      x.ca-servers.ca.
ca.     172800  IN      NS      c.ca-servers.ca.
```

Listing 1: Referral answer to dig ns ca @any.ca-servers.ca.

Notice that the answer for the NS records is in the "Authority Section": this signals that this answer is known as "referral": a type of answer in which "in which a server, signaling that it is not (completely) authoritative for an answer, provides the querying resolver with an alternative place to send its query." [15]. In this case , the AA bit in the answer is clear.

Given that the Root servers are not *are not authoritative for .ca*, a recursive can ask the same query directly to any of the .ca servers. Listing 2 shows the results:

```
; authanswer
$ dig ns ca @any.ca-servers.ca.
;; ANSWER SECTION:
ca.     86400   IN      NS      c.ca-servers.ca.
ca.     86400   IN      NS      j.ca-servers.ca.
ca.     86400   IN      NS      x.ca-servers.ca.
ca.     86400   IN      NS      any.ca-servers.ca.
```

Listing 2: Answer to dig ns ca @any.ca-servers.ca.

Notice that the records on this response from any.ca-servers.ca. are presented in "ANSWER SECTION" (and not on "AUTHORITY SECTION"), and the AA bit is set.

Even though the records are the same as in the case of the Roots, their TTL is different: 2 days and 1 day, respectively. The same behavior can be observed for the A records of the .ca servers (dig A ca @b.root-servers.net and dig A ca @any.ca-servers.ca.).

We refer to this as *TTL duplication*: the same record is defined in both authoritative servers and on its respective parent node server as a glue record.

This duplication poses a challenge for recursive resolvers: which value should the trust? The one from the glue or the other from the authoritative name server?

In theory, the answer from the authoritative should be the one used, since they are the servers who are authoritative for the zone. The roots, in this example, are not "completely" authoritative [15]. This behavior can be observed in various zones as well, not only at the roots.

Assuming that the records are the same as for .ca, in this section we investigate, in the wild, which one resolvers get to use: the values provided by the parent, or the authoritative. According to RFC2181 [10], a recursive should use the authoritative answer instead (§5.4.1).

Given the importance of TTLs in DNS resilience (cache duration), we investigate in this section which values recursives trust: the parent or the authoritative name server.

	NS record	A Record	Source
Total Answers	128,382	98,038	
TTL>3600	38	46	Unclear
TTL=3600	260	233	Parent
60<TTL<3600	6,890	4,621	Parent,others
TTL=60	60,803	46,820	Authoritative
TTL<60	60,391	46,318	Authoritative

Table 5: Distribution of TTL of DNS answers for cachetest.nl

.

A Records of amazon.com

domain	IP	TTL	
amazon.com.	205.251.242.103	60	
amazon.com.	176.32.103.205	60	
amazon.com.	176.32.98.166	60	
Records amazon.com			
domain	NS	TTL-auth	TTL-.com
amazon.com	ns1.p31.dynect.net.	3600	172,800
amazon.com	pdns6.ultradns.co.uk.	3600	172,800
amazon.com	ns2.p31.dynect.net.	3600	172,800
amazon.com	ns4.p31.dynect.net.	3600	172,800
amazon.com	pdns1.ultradns.net.	3600	172,800
amazon.com	ns3.p31.dynect.net.	3600	172,800
A records of NS records			
domain	IP	TTL	
ns1.p31.dynect.net.	208.78.70.31	86400	
pdns6.ultradns.co.uk.	204.74.115.1	3600	
ns2.p31.dynect.net.	204.13.250.31	86400	
ns4.p31.dynect.net.	204.13.251.31	86400	
pdns1.ultradns.net.	204.74.108.1	3600	
ns3.p31.dynect.net.	208.78.71.31	86400	

Table 6: amazon.com NS and TTLs on 20180803

A.2 Experiments

We use the same setup used in §3 and configured the TTL values of our name servers (ns[1,2].cachetest.nl) as following:

- Referral value (at the .nl zone): TTL value of 3600 s
- Answer value (the authoritative servers (ns[1,2].cachetest.nl): TTL =60 s

We then use ~10,000 Atlas probes (yielding ~15,000 vantage points) to query their local recursives for the NS records of cachetest.nl (Ripe measurement ID: 15642129). We did the same for A records, in another measurement (ID: 15628702). We then analyze the distribution of TTL values of the answers obtained.

Table 5 shows the distribution of TTLs for this two experiments. As can be seen, the large majority of answers (about 95%) receive the TTL values as provided by the authoritative servers, and not the one provided by the parent .nl server. As such, we can conclude that, with these two experiments, for A and NS records, most recursives will forward their clients the TTL provided by the authoritatives of a zone.

A.3 Looking At a Recursive's Cache

In §A.2 we showed that, in the wild, most recursives will serve their clients with the TTL provided by authoritative, and not by referrals.

In this section, we look into the caching of Unbound DNS server when we query for the NS records from amazon.com. Similarly to .ca example of the previous section, these records have two TTL values 3600 s a their authoritative, 172,800 s at the parent .com server, as can be seen in Table 6.

To determine which specific record two popular recursives software use, we analyze both Unbound (version 1.5.8) and BIND (9.10.3) cache contents. We start the servers with cold cache and send only one query: `dig ns amazon.com`. We them dump the contents of the cache of these servers and see which values they store.

Listing 3 and Listing 4 show the contents of the cache of both BIND and Unbound after send the aforementioned query. As can be seen, the values stored in the cache are the ones (decremented by few seconds, when we dumped its contents) provided by the authoritative server, and not by the parents authoritative as referrals.

```
; authanswer
amazon.com.
3595    NS      ns1.p31.dynect.net.
3595    NS      ns2.p31.dynect.net.
3595    NS      ns3.p31.dynect.net.
3595    NS      ns4.p31.dynect.net.
3595    NS      pdns1.ultradns.net.
3595    NS      pdns6.ultradns.co.uk.
```

Listing 3: BIND cache dump excerpt for amazon.com

```
;rrset 3594 6 0 7 3
amazon.com.  3594  IN  NS  pdns1.ultradns.net.
amazon.com.  3594  IN  NS  ns1.p31.dynect.net.
amazon.com.  3594  IN  NS  ns3.p31.dynect.net.
amazon.com.  3594  IN  NS  pdns6.ultradns.co.uk.
amazon.com.  3594  IN  NS  ns4.p31.dynect.net.
amazon.com.  3594  IN  NS  ns2.p31.dynect.net.
```

Listing 4: Unbound cache dump excerpt for amazon.com

A.4 Discussion and Implications

Understanding how glue records affect TTLs is important to show that zone operators are in fact in charge of how long their NS records will remain in their clients cache. They should choose these values carefully, as discussed in §8.

B TTL MANIPULATIONS ACROSS DATASETS

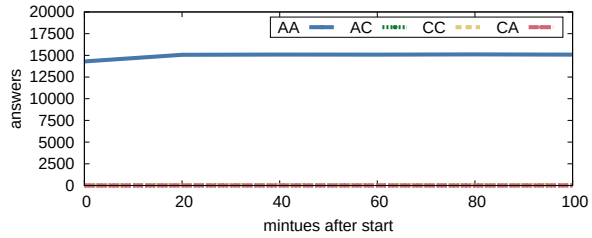
In §3.4, we saw that there are a number of AC-type answers—queries that are answered by the authoritative even though we expected them to be served from a cache. We now investigate the relation between the number of AC-answers with regards the TTL used for the DNS records.

Figure 13 shows response types over time for our four different TTLs. Analyzing these figures, we can see that with exception of a 60 s, where all queries go to the AA, the number of AC answers is relatively constant. This consistency suggests cache fragmentation across the datasets. We also see that AA and CC values alternate, given as caches entries expires new queries are forwarded to the authoritatives.

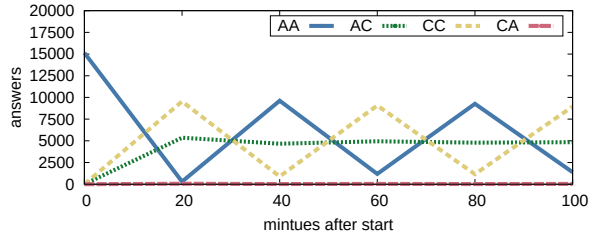
C LIST OF PUBLIC RESOLVERS

We use the following list of public resolvers in §3.4. This list is obtained by searching DuckDuckGo for "public" and "dns" on 2018-01-06.

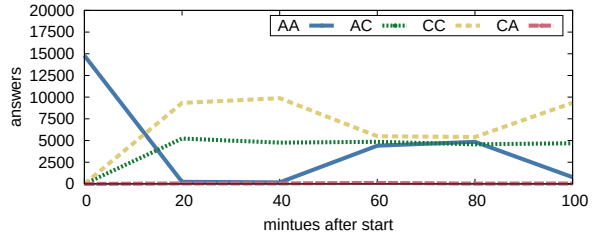
198.101.242.72	Alternate DNS
23.253.163.53	Alternate DNS
205.204.88.60	BlockAid Public DNS (or PeerDNS)
178.21.23.150	BlockAid Public DNS (or PeerDNS)
91.239.100.100	Censurfridns
89.233.43.71	Censurfridns



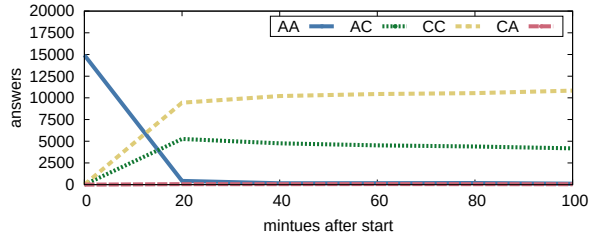
(a) TTL 60 s



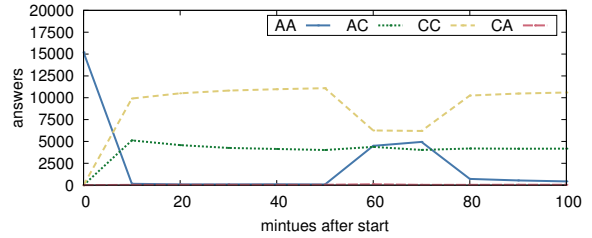
(b) TTL 1800 s



(c) TTL 3600 s



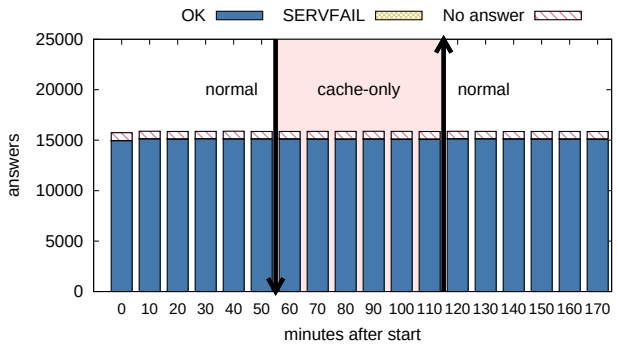
(d) TTL 86400 s (1 day)



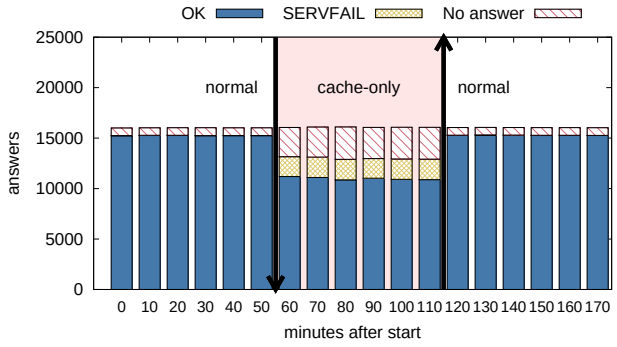
(e) TTL 3600 s (1 hour), Probing Interval T=10min

Figure 13: Fraction of query answer types over time

2001:67c:28a4::	Censurfridns
2002:d596:2a92:1:71:53::	Censurfridns
213.73.91.35	Chaos Computer Club Berlin
209.59.210.167	Christoph HochstÃďtter
85.214.117.11	Christoph HochstÃďtter
212.82.225.7	ClaraNet
212.82.226.212	ClaraNet
8.26.56.26	Comodo Secure DNS
8.20.247.20	Comodo Secure DNS
84.200.69.80	DNS.Watch
84.200.70.40	DNS.Watch
2001:1608:10:25::1c04:b12f	DNS.Watch
2001:1608:10:25::9249:d69b	DNS.Watch
104.236.210.29	DNSReactor
45.55.155.25	DNSReactor
216.146.35.35	Dyn
216.146.36.36	Dyn
80.67.169.12	FDN
2001:910:800::12	FDN
85.214.73.63	FoeBud
87.118.111.215	FoolDNS
213.187.11.62	FoolDNS
37.235.1.174	FreeDNS
37.235.1.177	FreeDNS
80.80.80.80	Freenom World
80.80.81.81	Freenom World
87.118.100.175	German Privacy Foundation e.V.
94.75.228.29	German Privacy Foundation e.V.
85.25.251.254	German Privacy Foundation e.V.
62.141.58.13	German Privacy Foundation e.V.
8.8.8.8	Google Public DNS
8.8.4.4	Google Public DNS
2001:4860:4860::8888	Google Public DNS
2001:4860:4860::8844	Google Public DNS
81.218.119.11	GreenTeamDNS
209.88.198.133	GreenTeamDNS
74.82.42.42	Hurricane Electric
2001:470:20::2	Hurricane Electric
209.244.0.3	Level3
209.244.0.4	Level3
156.154.70.1	Neustar DNS Advantage
156.154.71.1	Neustar DNS Advantage
5.45.96.220	New Nations
185.82.22.133	New Nations
198.153.192.1	Norton DNS
198.153.194.1	Norton DNS
208.67.222.222	OpenDNS
208.67.220.220	OpenDNS
2620:0:ccc::2	OpenDNS
2620:0:ccd::2	OpenDNS
58.6.115.42	OpenNIC
58.6.115.43	OpenNIC
119.31.230.42	OpenNIC
200.252.98.162	OpenNIC
217.79.186.148	OpenNIC
81.89.98.6	OpenNIC
78.159.101.37	OpenNIC
203.167.220.153	OpenNIC
82.229.244.191	OpenNIC
216.87.84.211	OpenNIC
66.244.95.20	OpenNIC
207.192.69.155	OpenNIC
72.14.189.120	OpenNIC
2001:470:8388:2:20e:2eff:fe63:d4a9	OpenNIC
2001:470:1f07:38b::1	OpenNIC
2001:470:1f10:c6::2001	OpenNIC
194.145.226.26	PowerNS

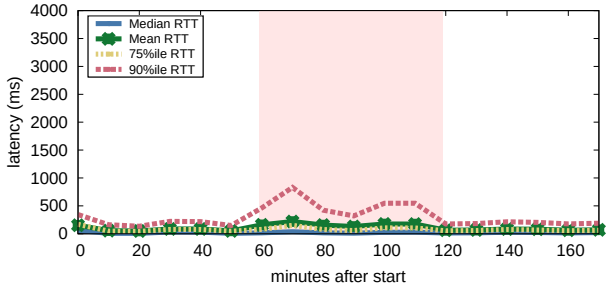


(a) Experiment D: 1800-50p-10min-ns1; arrows indicate DDoS start and recovery

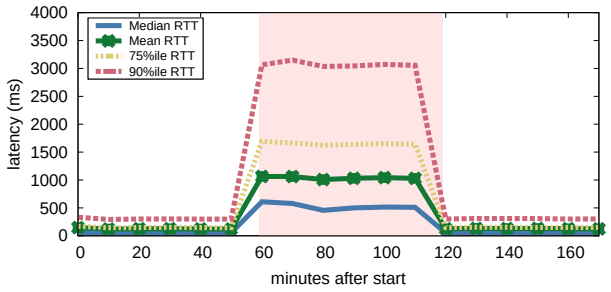


(b) Experiment G: 300-75p-10min-2Nses; arrows indicate DDoS start and recovery

Figure 14: Answers received during DDoS attacks (extra graphs).



(a) Experiment D: 50% packet loss on 1 NS only (1800 s TTL)



(b) Experiment G: 75% packet loss (300 s TTL)

Figure 15: Latency results; Shaded area indicates the interval of an ongoing DDoS attack (extra graphs)

77.220.232.44	PowerNS
9.9.9.9	Quad9
2620:fe::fe	Quad9
195.46.39.39	SafeDNS
195.46.39.40	SafeDNS
193.58.251.251	SkyDNS
208.76.50.50	SmartViper Public DNS
208.76.51.51	SmartViper Public DNS
78.46.89.147	ValiDOM
88.198.75.145	ValiDOM
64.6.64.6	Verisign
64.6.65.6	Verisign
2620:74:1b::1:1	Verisign
2620:74:1c::2:2	Verisign
77.109.148.136	Xiala.net
77.109.148.137	Xiala.net
2001:1620:2078:136::	Xiala.net
2001:1620:2078:137::	Xiala.net
77.88.8.88	Yandex.DNS
77.88.8.2	Yandex.DNS
2a02:6b8::feed:bad	Yandex.DNS
2a02:6b8:0:1::feed:bad	Yandex.DNS
109.69.8.51	puntCAT

D EXTRA GRAPHS FROM PARTIAL DDOS

Although we carried out all experiments listed in Table 4 (§5), the body of the paper provides graphs for key results only. In this section, we include the graphs for experiments D and G.

Figure 14 shows the number of answers for experiments D and G, while Figure 15 shows the latency results. We can see at Figure 14a that when 1 NS fails (50%), there is no significant changes in the number of answered queries. Besides, most of the users will not notice in terms of latency (Figure 15a).

Figure 14b shows the results for Experiment G, in which the TTL is 300 s (thus half of the probing interval). Similarly to experiment I, this experiment should not have caching active given the TTL is shorter. Moreover, it has a packet loss rate of 75%. At this rate, we can see that the far majority (72%) of users obtain an answer, with a certain latency increase (Figure 15b).

E ADDITIONAL INFORMATION ABOUT SOURCES OF RETRIES

In §6.2, we summarized sources of retries. Here we provide additional experiments to understand how many retries occur in current DNS software. These experiments confirm prior work by Yu *et al.* [56] still holds. We examine two popular recursive resolver implementations: BIND 9.10.3 and Unbound 1.5.8.

To evaluate these implementations we deploy our own recursive resolvers, and record traffic with the authoritatives (cachetest.net, not cachetest.nl as in the rest of the paper) up and then with them inaccessible. For consistency, we only retrieve AAAA records for sub.cachetest.net. The recursives operates normally, learning about the authoritatives from .net and querying one or both based on its internal algorithms. All queries are made with a cold cache. In total,

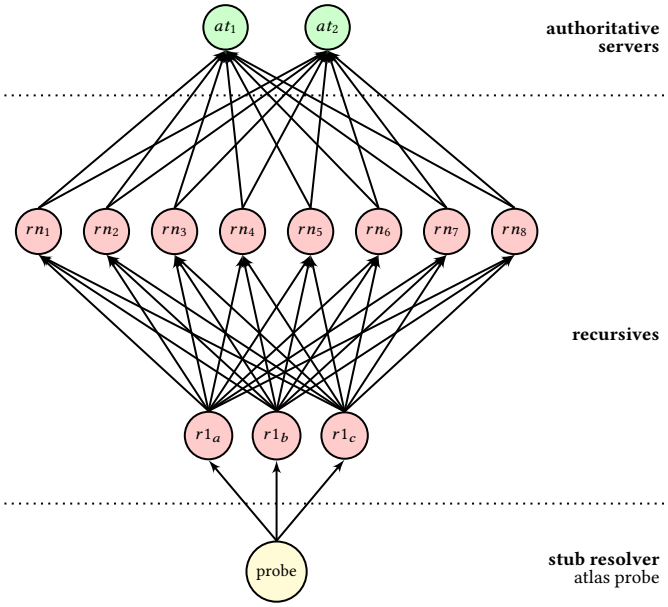


Figure 17: probe 284777 on dataset i. all $r1$'s are connected to all rn's, which are connected to all at's.

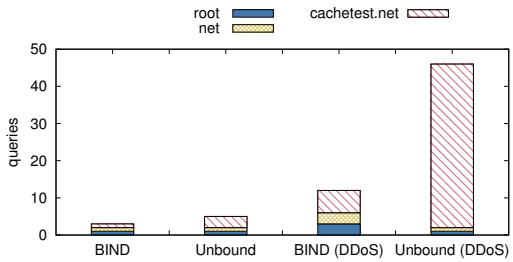


Figure 16: Histogram of queries from recursives resolvers

we send 100 queries per experiment and present in this section the results.

We start by analyzing traffic monitored at the recursive resolvers. We consider only queries sent from the recursives towards any of the authoritative servers in the cachetest.net hierarchy: the Root DNS servers, the .net authoritative servers, and the authoritative servers for cachetest.net.

Figure 16 shows the results of this experiment, with normal behavior in the left two groups, and the inaccessible DDoS attempts in the right two. From this figure we see that, in normal operation, a few queries are required to look up the target domain: the server looks up AAAA records for both nameservers and then queries the AAAA record for the target (Unbound also queries for the AAAA records of the authoritative name serves of cachetest.net, which are not in the .net glue neither in the cachetest.net zone, which explains the difference with BIND, which does not.)

Under normal operations, it takes BIND 3 DNS queries (1 to the root, one to .net, and one to cachetest.net authoritative servers) to resolve the AAAA records of sub.cachetest.net. Unbound, in turn, takes 5 queries—it sends 2 more to the authoritatives of cachetest.net asking for AAAA records of their NS records, which do not exist (thus this difference can be neglected since it may be specific to our setup).

The two groups on the right of Figure 16 show much more aggressive queries during server failure. Both BIND and Unbound send far more DNS queries to all authoritative servers available. We see that BIND sends 12 queries instead of 3 it does under normal operations (4× increase), while Unbound sends 46 queries in total (14.6× increase): Unbound keeps on trying to resolve the domains and the AAAA records of the authoritative name servers (the latter does not exist, which it is specific to our domain, 30 in total out of 46). If we disregard these queries, Unbound would then increase the total number of queries from 6 to 16, a 6.4× increase. (We repeated this experiment 100 times and all results were consistent.)

Another difference between BIND and Unbound is that when cachetest.net is unreachable, BIND will ask both parents (the Roots and .net) for its records again (shown by increase in the root and net portions of bind DDoS vs. bind non-DDoS in Figure 16). Unbound does not retry these queries.

This experiment is consistent with our observation that we see 7× more traffic than usual during near-complete failure. We do not mean to suggest that these retries are incorrect—recursives need to retry to account for packet loss.

F ANALYSIS OF RETRIES FROM A SPECIFIC PROBE

In §6 we examined changes in legitimate traffic during a DDoS event, showing traffic greatly increases because of retries by each of possibly several recursive resolvers. That analysis looked at aggregate statistics. We next examine a *specific* probe to confirm our evaluation.

We consider RIPE Atlas Probe 28477, in Experiment I. This probe is located in Australia. It has three "local recursives" ($R1_a$, $R1_b$, $R1_c$) and 8 "last layer" recursives (the ones that ultimately send queries to authoritatives, Rn_1–Rn_8), as can be seen in Figure 17.

In this section, we analyze all the incoming queries arriving on our authoritatives for AAAA records of 28477.cachetest.nl. We do not analyze NS queries, since we cannot associate them to this probe.

We then analyze data collected at both the client side (Ripe Atlas probes) and the server side (Authoritative servers). We show the summary of the results in Table 7. In this table, we highlight in light color the probing intervals T in which the simulated DDoS started (dataset I, 90% packet drop on both authoritative servers – Table 4).

Client Side: Analyzing this table, we can learn the following from the Client side:

During *normal operations*, we see 3:3:3 at the client side (3 queries, 3 answers, and 3 $R1_n$ used in the answers). By default, each Ripe Atlas probe sends a DNS query to each $R1n$.

During the *DDoS*, this slightly change: we see 2 answers from 2 $R1$s – the other either times out or SERVFAIL. Thus, at 90% packet loss on both authoritatives, still 2 of 3 queries are answered at the client (and there is no caching since the TTL of records is shorter than the probing interval).

This is in fact a good result given the simulated DDoS.

Authoritative side: Now, let's consider the authoritative side:

T	**Client View** (Atlas probe)			**Authoritative Server View**						
	Queries	**Answers**	# $R1$	**Queries**	**Answers**	# ATs	# Rn	# ($Rn - AT$)	**Max.** (% total) (Que($Rn - AT$))	**Queries** (top-2 Rn's)
1	3	3	3	3	3	2	2	3	1 (33.3%)	2;1
2	3	3	3	5	5	2	4	5	1 (20.0%)	2;1
3	3	3	3	6	6	2	6	6	1 (16.7%)	1;1
4	3	3	3	5	5	2	2	3	2 (40.0%)	4;1
5	3	3	3	3	3	2	3	3	1 (33.3%)	1;1
6	3	3	3	6	6	2	5	5	2 (33.3%)	2;1
7	3	2	2	29	3	2	2	4	10 (34.5%)	15;14
8	3	2	2	22	6	2	7	9	7 (31.8%)	10;7
9	3	2	2	21	1	2	3	6	9 (42.9%)	16;3
10	3	2	2	11	4	2	3	4	6 (54.5%)	8;2
11	3	2	2	21	3	2	4	6	15 (71.4%)	17;2
12	3	2	2	23	1	2	8	8	15 (65.2%)	15;2
13	3	3	3	3	3	2	3	3	1 (33.3%)	1;1
14	3	3	3	4	2	2	4	4	1 (25.0%)	1;1
15	3	3	3	8	8	2	7	7	7 (87.5%)	1;1
16	3	3	3	9	9	2	3	3	6 (66.7%)	6;3
17	3	3	3	9	9	2	5	5	5 (55.6%)	5;1

Table 7: Client and Authoritative view of AAAA queries (probe 28477, measurement I)

In *normal operation* (before the DDoS), we see that the 3 queries sent by the clients (client view) lead to 3 to 6 queries to be send by the R_n, which reach the authoritatives. Every query is answered, and both authoritatives are used (# ATs).

We also see that the number of R_n used before the attack changes from 2 to 6 (# *Rn*). This choice depends on how $R1_*$ choose their upper layer recursives (R_{n-1}) and ultimately how R_{n-1} chooses *Rn*.

We also can observe how each *Rn* chooses each authoritative (# Rn-AT): this metric show the number of unique combinations of R_n and authoritatives observed. For example, for $T = 1$, wee see that three queries reach the authoritatives, but only 2 *Rn* were used. By looking at the number of unique recursive-AT combination, we see that there are three. As a consequence, one of the *Rn* must have used both authoritatives.

During the DDoS, however, things change: we see that the still 3 queries send by probe 28477 (client side) now turn into 11–29 queries received at the authoritatives, from the maximum of 6 queries before the DDoS. The client remains unaware of this.

We see three factors at play here:

(1) **Increase in number of used** *Rn*: we see that the number of number of used *Rn* also increases a little (3.6 average before to 4.5 during). (However, not all *Rn* are used all the time during the DDoS. We have not enough data of why this happens – but surely is how the lower layer recursives chooses these ones, or how they are set up.)
(2) **Increase in the number of authoritatives used by each** *Rn*: Each *Rn*, in turns, contacts an average of 1.13 authoritatives before the attack (average of UniqRn-AT/UniqRn). During the DDoS, this number increases to 1.37.
(3) **Retrials by each recursive**, either by switching authoritatives or not: we found this to be the most important factor for this probe. We show the number of queries sent by the top 2 recursives (QueriesTop2Rn) and see that compared to the total number of queries, they comprise the far majority.

For this probe, the answer of why so many queries is a combination of factors, being the retrials by *Rn* the most important one. From the client side, there is no strong indication that the client's queries are leading to such an increase in the number of queries to authoritatives during a DDoS.