Trufflehunter: Cache Snooping Rare Domains at Large Public DNS Resolvers

Audrey Randall
aurandal@eng.ucsd.edu
UC San Diego

Enze Liu
e7liu@eng.ucsd.edu
UC San Diego

Gautam Akiwate
gakiwate@cs.ucsd.edu
UC San Diego

Ramakrishna Padmanabhan
ramapad@caida.org
CAIDA, UC San Diego

Geoffrey M. Voelker
voelker@cs.ucsd.edu
UC San Diego

Stefan Savage
savage@cs.ucsd.edu
UC San Diego

Aaron Schulman
schulman@cs.ucsd.edu
UC San Diego

ABSTRACT

This paper presents and evaluates Trufflehunter, a DNS cache snooping tool for estimating the prevalence of rare and sensitive Internet applications. Unlike previous efforts that have focused on small, misconfigured open DNS resolvers, Trufflehunter models the complex behavior of large multi-layer distributed caching infrastructures (e.g., such as Google Public DNS). In particular, using controlled experiments, we have inferred the caching strategies of the four most popular public DNS resolvers (Google Public DNS, Cloudflare Quad1, OpenDNS and Quad9). The large footprint of such resolvers presents an opportunity to observe rare domain usage, while preserving the privacy of the users accessing them. Using a controlled testbed, we evaluate how accurately Trufflehunter can estimate domain name usage across the U.S. Applying this technique in the wild, we provide a lower-bound estimate of the popularity of several rare and sensitive applications (most notably smartphone stalkerware) which are otherwise challenging to survey.

CCS CONCEPTS

• **Networks** → **Naming and addressing**; **Network measurement**.

ACM Reference Format:
Audrey Randall, Enze Liu, Gautam Akiwate, Ramakrishna Padmanabhan, Geoffrey M. Voelker, Stefan Savage, and Aaron Schulman. 2020. Trufflehunter: Cache Snooping Rare Domains at Large Public DNS Resolvers. In *ACM Internet Measurement Conference (IMC '20), October 27–29, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3419394.3423640

1 INTRODUCTION

Mitigating harmful Internet behavior requires monitoring its prevalence; specifically, to direct efforts to restricting services and blocking certain types of traffic. While there is a range of approaches to measure the prevalence of widespread abuse (e.g., spam [54]), characterizing the amount of rare abuse—where a small number of users experience or cause a significant amount of harm—has remained elusive. For example, while a few surveillance apps [13] have been found on devices belonging to participants of clinical studies of intimate partner violence [22, 28], the prevalence of these apps in the broader population is still unknown.

Harmful Internet behavior manifests in many different forms, using different protocols and on different platforms. However, virtually all depend on making queries to the Domain Name Service (DNS). Thus, the prevalence of a given source of abuse can be characterized by the number of requests for its associated domain names. While in most cases, it is not possible as a third party to directly measure the number of global DNS queries for a given name, we can infer them indirectly using DNS cache snooping: a technique that probes DNS resolvers to observe if a domain is in the cache, therefore implying that a user must have previously accessed it.

In this work, we focus on techniques for cache snooping large public DNS resolvers. Due to their scale, public resolvers both provide large-scale measurement opportunities and, due to their aggregation, sidestep some of the traditional privacy concerns of cache snooping. For example, as of May 2020, APNIC's DNS service popularity measurements indicate that ∼10% of web users appear to use Google Public DNS (GPDNS) as their primary DNS resolver [39], while Cloudflare and OpenDNS each serve ∼1% of web users.

However, public DNS resolvers consist of many independent caches operating in independent Points-of-Presence (PoPs), which makes them among the most challenging DNS resolvers to cache snoop. Through controlled experiments, we infer the load-balanced multi-layer distributed caching architectures of the four most popular resolvers. To the best of our knowledge, we are the first to model the behavior of these caching architectures and how they relate to user accesses. Building on these models, we demonstrate that it is possible to snoop public DNS PoPs and estimate how many caches contain a specific domain. Surprisingly, we found that GPDNS appears to dynamically scale the number of caches that contain a particular domain name based on the number of users accessing it; we observed up to several thousand uniquely identifiable caches for one domain name (Section 5). This behavior is a likely explanation for the unusual caching behavior of GPDNS that was reported, but not explained, in prior work [59, 62].

We present Trufflehunter, a tool to snoop the caches of public DNS resolvers. We evaluate the accuracy of Trufflehunter's cache behavior modeling with a large-scale controlled experiment. Our relative error in estimating the number of filled caches for each resolver varies from 10% to -50%, with the exception of one unusual Cloudflare location where our error is 75% (Section 5). Trufflehunter's error varies depending on the caching architecture of the resolver: it can estimate the cache occupancy of OpenDNS and Cloudflare more accurately than Quad9 and GPDNS. This error may seem large, but because Trufflehunter consistently *underestimates* cache occupancy, it can provide a *lower-bounded* estimate of the prevalence of rare user behaviors (Section 6).

We demonstrate Trufflehunter with several case studies on abusive Internet phenomena. We found that some of the most concerning smartphone stalkerware apps have a minimum of thousands of simultaneous active users. We also found academic contract cheating of the services were significantly more popular than the others, and their popularity wanes during the summer.

Trufflehunter is open source and available at:

https://github.com/ucsdsysnet/trufflehunter

2 BACKGROUND

In this section, we describe how we can measure the prevalence of rare Internet user activity by probing the caches of public DNS resolvers. We begin by describing why public DNS services have become a key vantage point for observing uncommon behavior of Internet users. Then, we describe how the complex caching architecture of these services makes it possible to externally measure the minimum number of simultaneous users that have queried for a domain name. Finally, we outline how this complex caching architecture makes it challenging to accurately estimate the number of users that accessed a domain with cache snooping techniques.

2.1 A variety of users query public resolvers

Initially, the set of users that adopted public DNS were Internet power users who were privacy and security conscious. However, public DNS is now becoming a popular default configuration on networks and in software. This trend has caused a wide variety of users to adopt these services; indeed, many public DNS users today did not explicitly configure their devices to use public DNS. The adoption of these services is largely driven by two factors: (1) network operators and equipment vendors configuring networks and devices so that users default to using public DNS services as their primary or secondary resolver, and (2) developers hard-coding public DNS resolution into their software.

Enterprise network administrators have switched from running their own DNS resolvers to pushing users to public DNS resolvers because that can improve reliability [30]. Small-scale ISPs have also switched to public DNS to avoid the operation and maintenance cost of providing their own DNS resolver. For instance, GPDNS has a formal vetting process where ISPs can request to remove GPDNS's rate limits so they can have their entire customer base use GPDNS as their primary resolver [24]. Also, public DNS is often adopted by administrators and ISPs because it provides additional security measures for their users. For example, Quad9 and OpenDNS both block DNS requests for domains that are reported to be malicious on threat intelligence feeds [40, 56]. This security feature was reported to be the primary reason the NYC city-run public WiFi network switched to using Quad9 as its default DNS resolver [55]. Security is also cited as the primary reason that enterprises and schools have switched to using OpenDNS [52]. Public DNS resolvers have also been set as the default resolver in networking equipment as a means of improving performance. The most notable example of this trend is the "Google Home" WiFi router, which ships with its default configuration to resolve all DNS queries with GPDNS [21].

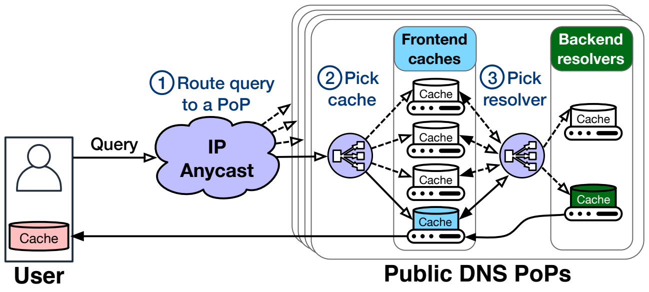


Figure 1: How Public DNS services cache responses

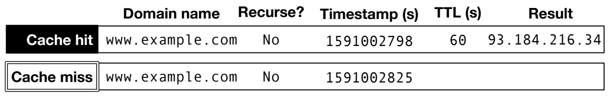
Software developers have also increased public DNS adoption by hard-coding their software to send DNS requests to public resolvers. The most notable instances are because public DNS services offer the latest DNS security features. For example, when Firefox deployed the privacy protections provided by DNS-over-HTTPS, it hard-coded Cloudflare's public DNS resolver as the default resolver for all of their U.S. users [18]. Additionally, the reliability, and wide availability, of public DNS makes it a common choice as a hard-coded backup resolver. For example, in 2017 Linux distributions started shipping with GPDNS hard-coded as a backup resolver in SYSTEMD [45, 65], and in 2019 added Cloudflare as well [37].

2.2 Complex caching can reveal many active users

We now describe how cache snooping public DNS resolvers can provide a lower bound on the number of users accessing domain names—without revealing who has accessed these domain names. Public DNS resolvers do not operate as a large contiguous cache with global coverage. Rather, they operate many fragmented DNS caches [2, 49, 71] and load-balance queries across many caching resolver instances [36, 64].[1] This architecture can have a negative effect on their performance: even if a user recently resolved a domain name with a public DNS resolver, subsequent requests to that domain may not be serviced from a cache. However, this performance limitation is also an opportunity for establishing a non-trivial lower-bound on the number of users accessing a domain.

To demonstrate how cache snooping public DNS services can reveal a non-trivial number of users, we start by explaining how a typical query is cached in a public DNS resolver's hierarchy (Figure 1). In particular, we focus on the three steps used to resolve a query on a public resolver, where the response to the query can be cached in one of many independent caches. This cache architecture is a generalization of how all of the large public DNS resolver caches that we study operate. Section 3 later describes the details of how caching works in each of the resolvers.

[1]GPDNS served 400 billion responses per day in 2015 [26].



	Domain name	Recurse?	Timestamp (s)	TTL (s)	Result
Cache hit	www.example.com	No	1591002798	60	93.184.216.34
Cache miss	www.example.com	No	1591002825		

Figure 2: Example of cache snooping responses

① Users direct their query to one of the public DNS service's PoPs by sending the request to one of the service's anycast IP addresses [49]. These addresses are announced by routers in PoPs distributed geographically around the globe.[2] This anycast DNS architecture is similar to the anycast load balancing that the root DNS servers use [8, 12, 47, 60]. In our experiments we found that, for large resolvers, PoPs operate their caches independently (Section 5).

② Then, within a PoP, a query is load-balanced to a pool of frontend caches for the backend caching DNS resolvers [17]. The load balancer selects from the pool of frontend caches based on a policy that distributes the load across these caches. These frontend caches can be isolated, creating the possibility of having more independent locations users' query responses can be cached.

③ If the selected frontend cache does not have an entry for the domain name, the query will be forwarded to one of a pool of backend resolvers via a second load balancer. Backend resolvers operate independent caches, introducing yet another opportunity for multiple users to have their queries cached independently.

After the backend resolves the domain name, all of the caches along the path in the hierarchy are filled with the response. First, the backend resolver fills its cache, and in some cases responds directly to the user. Then the frontend resolver fills its cache, and finally the response is sent to the user, which fills its local cache. The presence of user-local caches makes it easier to count the number of users accessing a rarely-used domain name: it effectively limits the number of cache entries that can be created in a public DNS resolver to one per user at a point in time. All major operating systems operate have DNS caches [3, 7, 35, 43, 50], including MacOS, Windows, and Linux. Additionally, many browsers operate their own local cache such as Safari, Chrome, and Firefox. Some home networks and organizations also run a caching forwarding resolver for all users on their entire network. These local caches will effectively limit a household or organization to filling only one cache in a public resolver, per domain, at any point in time.

Snooping Public DNS caches can provide a lower-bound on the number of active users. If we can estimate how many of these caches hold a particular domain at any point in time, we can obtain a lower-bound on the number of simultaneously active users of that domain. However, this estimate is strictly a lower-bound on the number of users that may have requested this domain—we cannot observe how many users had their query serviced by one of the caches. As we are limited to counting only one user per cache, snooping will be most useful for estimating the popularity of *rare* domains. It will not provide any new information about the prevalence of domains that are already known to be popular.

Does DNS security limit counting users with cache snooping? DNS security and privacy technologies, such as DNS-over-HTTPs, DNS-over-TLS, as well as DNSSEC, do not change the cache model for DNS described above. All queries made with these protections enabled will be served out of the same caches as insecure queries.

[2] As of May 2020, GPDNS has 33 PoPs worldwide, Cloudflare has 46 PoPs in the U.S., Quad9 has 27 PoPs in the U.S., and OpenDNS has 11 PoPs in the U.S. & Canada.

2.3 Public DNS cache snooping challenges

The complex multi-level caching hierarchy makes it feasible to estimate a non-trivial number of active users of a domain name. Unfortunately, it also makes it challenging to accurately estimate the number of caches that have been filled. Cache snooping a resolver with a single cache—as has been investigated in prior work—is straightforward. The most direct way of snooping a cache is to "probe" it by making a query for a particular domain name with the Recursion Desired (RD) flag unset. Unsetting Recursion Desired prevents the backend resolver from doing a recursive query to get the uncached answer, causing it instead to report a cache miss by not including a result in the answer. The DNS response from a cache probe contains limited information (Figure 2) that answers the following questions: is the domain name currently cached (indicated by the existence of a DNS Answer section having a nonzero TTL), and how long has it been cached (as inferred from the response timestamp and the remaining TTL in the response)? Cache snooping a large public DNS service to measure how many caches are occupied is significantly more difficult because each probe (non-recursive DNS query) returns information about only a single cache in the resolver.

From the limited information available in these single-cache responses, we somehow need to determine how many independent caches have been filled at the resolver. Note that each cache probe will provide the status of only one cache, in one PoP: we do not know what cache, nor do we know what PoP, the response came from. Additionally, we cannot tell if the response came from a frontend cache, a backend cache, or both (if they are synchronized). We also need to consider what happens when a cache entry is shared between caches. For instance, is the same TTL value stored in the frontend cache when copying from a backend resolver? What about when cache entries are shared between frontend resolvers? We will show in the next section that the data in the simple cache probes described above (Figure 2) is sufficient to estimate how many caches a domain is in at one time.

3 SNOOPING PUBLIC DNS CACHES

Cache snooping public DNS resolvers with Trufflehunter is possible because it can interpret the multi-level caching architecture's behavior by observing DNS responses. Specifically, it sends a collection of cache snooping probes (non-recursive DNS queries) for a domain name towards a resolver and deduces what the responses reveal about how many caches are occupied by that domain name. Different resolvers can use widely different architectures, however, introducing resolver-specific challenges in interpreting responses to cache snooping probes. In this section, we describe how we analyzed and modeled resolvers' caching architectures. We also show how Trufflehunter's inference technique will be tailored to each resolver's architecture.

In this section, we describe the cache inference logic we built in Trufflehunter based on our observations about how the caching architecture operates for the four largest public DNS resolvers. We

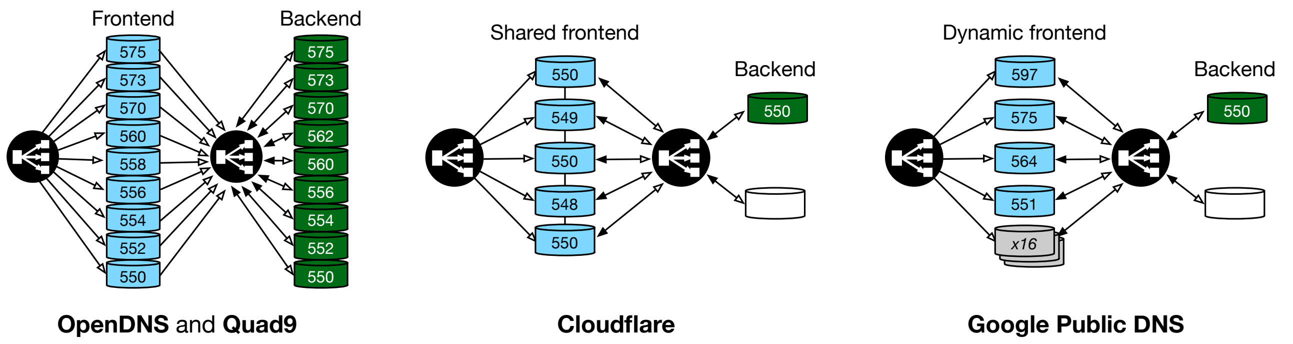


Figure 3: Inferred cache architectures of the four largest public DNS services (50 secs after filled with 600 sec maximum TTL)

explain how we inferred their behavior with a combination of controlled experiments—where we intentionally put a domain name in as many caches as possible in each resolver—and public information released by the resolvers. Each of the architectures presents their own opportunities or limitations for observing a non-trivial number of independent caches that have the same domain. We discovered that some resolvers employ caching strategies that allow us to measure a large number of user accesses, but others employ strategies that impose significant limitations on the number of caches we can observe. They also all present unique challenges that make it difficult to estimate the number of occupied caches with cache snooping.

Inferring behavior of DNS cache architectures

We infer the caching architecture of public resolvers by inserting DNS responses into as many caches as possible, and observing how often new cache entries appear to be made.

Cache-filling experiment. We send recursive queries for a unique domain name to one PoP of each public resolver once every two seconds. These queries were made by a single machine in AS 7922 (Comcast). Since our goal is to fill as many caches as possible, including the backend recursive resolver caches, we issue these queries with the Recursion Desired (RD) flag enabled. We controlled the domain name used in the experiment, allowing us to verify that certain responses were serviced by a backend recursive resolver. The behavior of the resolvers during this experiment will be similar to how the resolver's caches will look when a resolver has a constant stream of users requesting a domain name.

The data collected during this experiment are DNS responses from the resolvers. When resolvers operate independent caches, the primary indicator that a response is coming from a particular cache is the TTL in the response. We know that one of our queries caused a cache to be filled when the response contains the maximum TTL (i.e., the TTL returned by the authoritative nameserver). We know that a query was serviced from a cache—and therefore did not fill a new cache—if the TTL in the response is lower than the maximum TTL.[3] The TTL also reveals which cache the query was serviced from because TTLs of cached responses decrement one second per second: responses that were received N seconds apart, and also have a difference in their TTLs of N seconds, can be assumed to come from the same cache. Our observations of the pattern of TTLs in the responses form the basis for our technique (described in Section 4) to measure the number of filled caches with DNS cache snooping probes (i.e., repeated *non-recursive* DNS queries).

[3]Most DNS resolvers age the TTLs of cached DNS responses once every second.

TTL Line. Responses that originate from the same cache should fall on a line with a slope of -1 (since TTLs decrease once per second) on a graph of TTLs over time. We use the term "TTL line" to refer to this line. TTL lines originate from a point in time where we infer that a cache was filled because we observe a response with the maximum TTL (600 seconds in our controlled experiment).

Visualizing DNS resolver caching behavior. The results of this experiment are presented as follows. For each resolver, we plot a point for every DNS response we receive during the experiment. The x-value is the time the response was received, and the y-value is the TTL contained in the response. We also draw a TTL line each time we observe a response with the maximum TTL. We only plot the first 50 seconds of each experiment because that is sufficient to show the general caching behavior. To make it easier to understand what cache architecture is producing this behavior, and to provide a visual comparison between the architectures, we show the states of the three different cache architectures at the end of the 50-second period in Figure 3.

3.1 OpenDNS and Quad9

OpenDNS and Quad9 presented the most intuitive caching behavior of the public resolvers. They both appear to be operating independent frontend caches (Figure 3). This architecture means that Trufflehunter can observe at most N_b simultaneous active users of domain names, where N_b is the number of backend resolvers operating at a PoP.

Figure 4 depicts the results of the cache-filling experiment that demonstrates this behavior (we omit the plot for Quad9 because its behavior is similar). As OpenDNS received repeated queries over time, we observed nine responses with the maximum TTL (indicated by the vertical dotted lines). For each of these responses, we observed a query to the authoritative resolver. Therefore, we can conclude that a frontend cache did not have the entry cached, and the query was resolved by a backend resolver. All of the other responses that we received from OpenDNS had a TTL that fell on one of the TTL lines that originate from these nine responses with the maximum

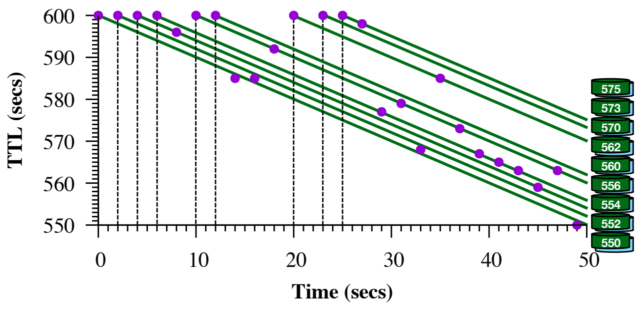


Figure 4: OpenDNS and Quad9 caching behavior

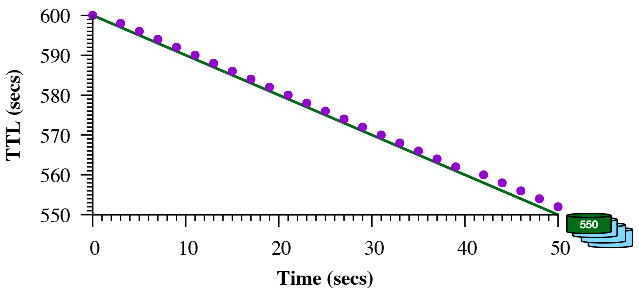


Figure 5: Cloudflare DNS shared caching behavior

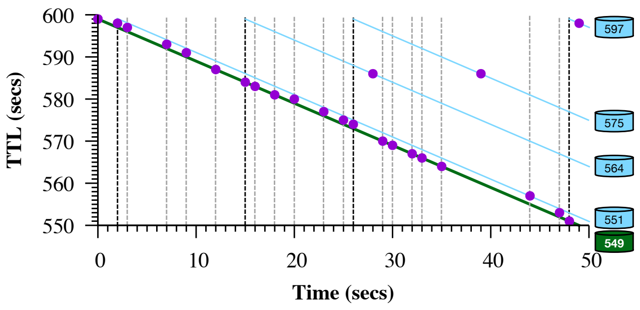


Figure 6: GPDNS's dynamic caching behavior

TTL (there is an inherent error of $+1, -1$ seconds that we address in Section 4.2). This behavior indicates that when a frontend cache does not have an entry, it copies the TTL from the response it gets from forwarding the query to a backend resolver, even if the backend resolver answers the query from its cache.

Estimating OpenDNS and Quad9 cache occupancy. Estimating the number of domain users active on Quad9 and OpenDNS requires estimating the number of independent backend resolvers that have the domain cached at any point in time. Recall that each TTL line in the recursive responses corresponds to one backend resolver having the domain name cached. Therefore, Trufflehunter can estimate this quantity by sending repeated cache probes for the domain, and counting the number of unique TTL lines it observes.

3.2 Cloudflare DNS

Cloudflare's DNS service is the only DNS resolver we evaluated that operates a shared frontend cache architecture, as shown in Figure 3. Specifically, it uses knotDNS's resolver, which has a shared backing database for its frontend caches (e.g., memcached) [27]. This architecture means that, unfortunately, the lower-bound of number of users accessing a domain on Cloudflare will be very conservative—at most one user within a TTL interval at a PoP. However, for domain names that are infrequently used, such as the ones we design Trufflehunter to observe, this limitation is not significant. Additionally, domains often have short TTLs and Cloudflare operates its resolvers from numerous PoPs, allowing us to provide meaningful lower-bound estimates.

Figure 5 shows the results of the cache-filling experiment for Cloudflare DNS. The recursive queries to Cloudflare all produce responses that fall on one TTL line that originates from the time the first query was made. However, this one TTL line is not perfect: it slowly deviates from a slope of -1. We believe this deviation is due to the errors in the TTL that accumulate as the frontend resolvers copy the cached response from the shared cache into their local cache.

Estimating Cloudflare cache occupancy. All Cloudflare resolvers in a PoP share one cache. They also slowly drift away from the "true" TTL line. As a result, it is not sufficient to simply send cache probes and count how many individual TTL lines we observe. Drift in TTL of cache responses will effectively extend the amount of time that a result resides in the cache, ultimately longer than the maximum TTL. If we only sample the cache infrequently, and at irregular intervals, we may conclude there are more user requests than there truly are.

Instead, Trufflehunter counts how many times the cache was filled by applying a peak-finding algorithm that will find the points where all caches were empty and then filled again (details in Section 4.2). It also only allows one of these peaks per maximum TTL of the domain name to account for any spurious peaks that may be due to errors from cache sharing.

3.3 Google Public DNS (GPDNS)

Google's public DNS resolver has a unique caching behavior that enables counting a large number of users that are actively requesting a domain name. Like other services, Google describes their caching architecture as load-balanced frontend caches and backend resolvers [23]. However, when a query hits a frontend cache that does not contain the domain name, GPDNS appears to create a new, independent frontend cache. This behavior means that it may be possible to count nearly all accesses to a domain name by cache snooping GPDNS's frontend caches. We suspect the presence of so many frontend caches is a design choice to scale the caching infrastructure dynamically based on the number of requests to a particular domain—and may be a reasonable way of protecting their infrastructure from DDoS attacks. The cloned caches are deleted when their "parent" backend cache expires.

Figure 6 shows the unique results of the cache-filling experiment for GPDNS. The initial query is recursively resolved by a backend resolver, and several future responses appear to be serviced from the same cache because they follow the same TTL line. Strangely, several of the responses contain a TTL that does not correspond with that initial TTL line, and they also do not have the maximum TTL (indicating a new cache has been filled). These caches also do not appear to be backend resolvers because we only see one request to our authoritative server per maximum TTL epoch. Previous work also noticed these caches,[4] but could not explain where they came from [59, 62].

This behavior appears to be a result of the unique way that Google shares query results from backend resolvers with frontend caches. When a query is load balanced to a frontend cache that does not have the domain name in cache, it will forward the query to a backend resolver. The backend resolver will then do the same thing as other public DNS resolvers: namely, send the response to the query to the user, then it will fill the frontend cache that was empty. The exact way that they fill the cache, however, is unique: the backend cache will fill the *maximum TTL* ($TTL_{max} - 1$ in the case of GPDNS [19]) in the frontend cache, rather than filling it with the current TTL of the cached entry at the backend cache. Therefore, each cache miss to a frontend cache fills a uniquely identifiable new frontend cache. We depict this behavior in Figure 6. Each cache hit to the backend resolver is marked with a vertical dotted line that marks that a frontend cache was filled at that time instant with the maximum TTL. When we draw TTL lines starting at these lines, we see that indeed four of them intersect the points that came from previously unexplained caches.

These cache entries may seem problematic because they effectively lengthen the duration that a domain entry is cached past the maximum TTL from the authoritative server. Fortunately, we observed that these cache entries are deleted as soon as the cache entry expires at the backend resolver.

Estimating GPDNS cache occupancy. While this cache-filling strategy creates an opportunity to count as many users as there are frontend caches, it also makes it difficult to cache snoop. The problem is that GPDNS fills a new frontend cache regardless of if the query is from a user (i.e., recursive) or a cache probe (i.e., non-recursive). Fortunately, it is possible to distinguish the caches created by cache probes from caches filled by users. Essentially, we account for all TTL lines that would be created by Trufflehunter's cache probes. When we observe a probe response that has a TTL on one of these lines, we simply discard it. Note that a consequence of this approach is that the more frequently we probe, the more frontend caches we may fill, and therefore the fewer users we can observe. To address this problem, we probe infrequently (i.e., five times per minute) relative to the duration of the maximum TTL (e.g., ten minutes), rather than probing as fast as possible.

Summary. This experiment demonstrates that public DNS resolvers often operate more than one independent cache in each PoP, creating the opportunity to estimate the number of users resolving a domain. We also describe how, by analyzing the TTLs obtained in DNS cache snooping, it is possible to count the number of occupied caches. In Section 5, we evaluate how well these cache snooping techniques work with a controlled experiment across most of the U.S. PoPs of these providers.

[4]One paper referred to them as "ghost caches" [59].

4 METHODOLOGY

In this section, we describe the details of the probing and analysis methodology Trufflehunter uses to observe all unique TTL lines for a specific domain. First, we describe how Trufflehunter probes public resolvers at multiple PoPs using CAIDA's distributed Archipelago infrastructure [9]. Then, we describe the technique Trufflehunter uses to count unique TTL lines from these probe responses. Combined with the cache behavior models described in the previous section, Trufflehunter produces estimates of the number of cached copies of a domain across many of the PoPs of a public DNS resolver.

4.1 Probing multiple PoPs

Since DNS queries to public resolvers are routed using anycast [10], we have multiple opportunities to improve our estimates of the users of a domain. As each PoP implements multiple levels of caching, and each of these caches in turn can be probed for a domain of interest, each additional PoP we probe can significantly increase the lower bound on the number of users Trufflehunter can observe.

Ark enables Trufflehunter measurements across many PoPs. Measuring many domains over multiple PoPs requires a geographically distributed measurement infrastructure that offers considerable flexibility in the number and type of DNS measurements that it can make. In this study, we focus on the U.S. due to the diversity of PoPs used by public resolvers in the country. We considered three choices from which we could host Trufflehunter—RIPE Atlas [67], public clouds such as Amazon AWS, and CAIDA's Archipelago (Ark) project [9]—and chose the Ark network to run our measurements since it offers diverse vantage points and the flexibility to implement and run continuous, longitudinal measurements. Though RIPE Atlas probes are more numerous and can contact more PoPs than the Ark nodes, more restrictions apply to their use; consequently we used Atlas probes only as controlled users for the smaller-scale experiments described in Section 5. We also considered using AWS, Google Public Cloud, and Microsoft Azure, but found that the Ark nodes have considerably wider coverage of PoPs. We deployed Trufflehunter on 43 Ark nodes distributed across the U.S.

DNS location requests identify which PoPs probes route to. While the existence of multiple PoPs per resolver enables improved measurement capabilities, it also introduces a layer of complexity: we need to identify to which PoP Ark nodes' DNS queries are routed. Doing so enables our analyses of filled caches for a domain at the per-PoP level (Section 5) and subsequent aggregation per resolver (Section 6). Fortunately, all four of the largest public resolvers provide ways to determine to which PoP requests are routed. Google makes the locations of GPDNS resolvers available in the form of a TXT record for the domain "`locations.publicdns.goog.`" Querying this record gives a map of resolver IP addresses to three-letter location codes. Another TXT record available at domain "`o-o.myaddr.l.google.com`" returns the non-anycast IP address of the resolver answering the query, which can then be looked up in the map. Similarly, TXT queries to "`debug.opendns.com`" return the three-letter airport code of the PoP that the query routed to in the answer. Quad9 and Cloudflare similarly make their locations available via CHAOS TXT queries to the domain "`id.server`" [31, 61].

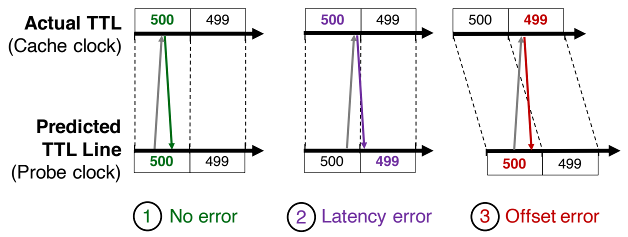


Figure 7: Errors matching cache probes to TTL lines

For all experiments in the remainder of the paper, we identify PoPs by the three-letter code of a nearby airport.

Data collection. Our deployment of Trufflehunter on Ark nodes continuously runs two sets of DNS measurements from each node:

1. Given a list of domains to search for, Trufflehunter performs a DNS request for each domain five times per minute towards each public resolver we study. These requests are made with the RD flag unset (non-recursive). We looked for evidence of rate limiting from our chosen resolvers to ensure we were not generating too onerous a load, but saw no instances of failures related to rate limiting (e.g., many `SERVFAIL` responses or query timeouts).

2. Trufflehunter makes a DNS location request once per minute to determine the PoP towards which it is currently routing queries, for each public resolver. While the Ark nodes usually only change PoPs on the scale of days, they do go through occasional periods of "fluttering," where the PoP they are routed toward changes more frequently (on the order of minutes).

4.2 Finding unique TTL lines using cache probing

Recall from Section 3 that estimating cache occupancy for the four public resolvers we study requires counting the number of unique TTL lines observed from cache probes of a domain at each PoP. We now describe potential errors that can affect our estimate of unique TTL lines and our methods to mitigate these errors. Trufflehunter uses these methods in conjunction with the methods described in Section 3 to estimate cache occupancy of a domain at each PoP. Together, these methods yield a lower bounded estimate of the number of caches that contain a domain at a PoP within a TTL epoch.[5]

Error correction method for GPDNS, Quad9, and OpenDNS. Intuitively, counting the number of unique TTL lines (as defined in Section 3) will yield the number of caches that contain a domain. However, correctly identifying TTL lines using cache probing is challenging since DNS TTLs only have precision to one second. This lack of precision can introduce off-by-one errors when comparing TTLs in responses originating from the same cache. This situation may lead to significant overestimates of the number of TTL lines, and therefore also active users. Our goal, however, is to present *lower-bounds* on the number of active users. We will now describe the nature of this problem in detail and describe the technique we developed to avoid overestimating the count of unique TTL lines.

To determine if a TTL returned by a cache probe lies on a particular TTL line, Trufflehunter needs two pieces of information: a sample of the actual TTL in the cache obtained by probing it, and a predicted TTL based on the probe clock's estimate of how much time has passed on the TTL line. Naively, one can determine if a TTL sample returned from the cache lies on the TTL line by checking if the predicted and actual TTLs are equal. However, TTLs in DNS responses only have precision to one second. Therefore, there can be sub-second measurement uncertainty. This uncertainty will lead to cases where Trufflehunter may overestimate the number of caches because TTLs do not lie on their predicted TTL line. Specifically, three cases can occur: the actual TTL matches the predicted TTL, the actual TTL is below the predicted TTL, or the actual TTL can be above the predicted TTL. The sources of uncertainty are as follows: the resolver's clock may not be synchronized with the probe clock, and there can be latency between when a resolver copies the TTL from its cache into a response, and when that response reaches the probing host. The effects of these sources of uncertainty are depicted in Figure 7.

First, consider the case where the resolver's clock is nearly synchronized with the cache probe's clock. In this case, TTL line prediction error will be due to the latency associated with cache probing. (1) There is no error when the TTL of the probe and cache stays the same between the time when a resolver generates its response and when the response arrives at the probing host.
(2) The TTL will be underestimated by one second when the probe TTL is decremented between the time when the resolver cache generates its response, and when it is received by the probing host.

Next, we consider the case where the clocks are not synchronized. (3) The actual TTL can be either overestimated, or underestimated, by one second. The error's direction depends on whether the probe clock is a fraction of a second ahead, or behind, the resolver's clock.

Trufflehunter uses the following heuristics to avoid overestimating the count of unique TTL lines due to TTL line prediction errors. If TTLs from cache probes lie on a single TTL line—with no probe TTLs falling on neighboring TTL lines (one second below or one second above)—we assume there are no errors. If we see a group of TTLs that lie on two neighboring TTL lines (one second apart), we assume there was an error, and we remove one line. If TTLs lie on a group of three or more neighboring TTL lines that are each one second apart, we remove the first and last lines in the group. Our rationale is that TTLs on the first TTL line may have been due to TTL overestimation, and the last line may have been due to TTL underestimation, but the TTLs on lines in the middle are likely composed of at least one correct measurement. This method can sometimes underestimate the count of TTL lines when lines that are one second above or below the predicted TTL line arise from other filled caches; however, this trade-off is consistent with our goal of presenting lower-bounds on the number of active users. We have found that this technique is reasonably accurate on resolvers with no confounding factors, such as OpenDNS. In Section 5, we evaluate the effectiveness of this technique using controlled experiments. We found it allows us to estimate the number of caches (TTL lines) within approximately 10% of the true value on the resolver with the least confounding variables (OpenDNS).

Error correction method for Cloudflare. Because Cloudflare only has one shared, distributed cache per PoP, we do not apply the above error method to estimate how many caches each PoP contains.

[5]TTL epoch is another term for the time period of the maximum TTL.

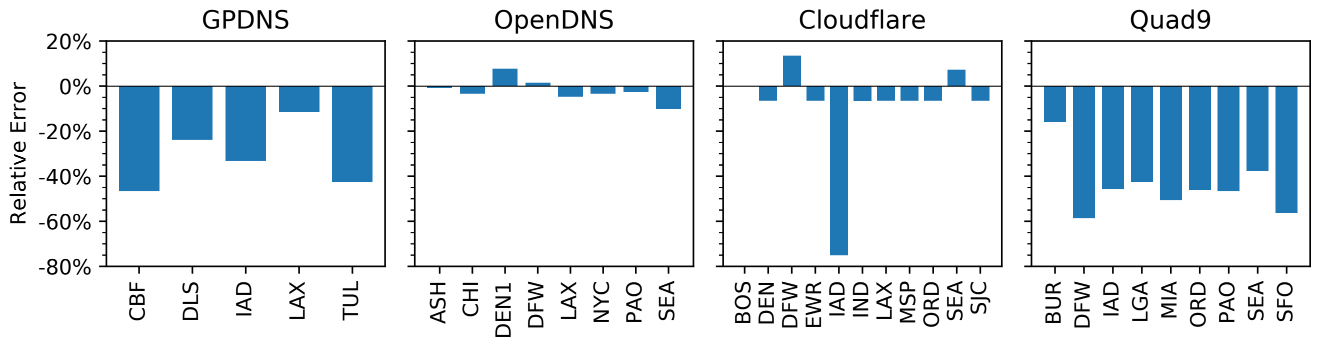


Figure 8: Cache estimation error

Instead, Cloudflare presents a different problem: its single externally visible cache is composed of many physical caches that all share the same record. The TTLs in this distributed cache drift away from the true TTL line over time, as the many individual caches that make up the single distributed cache share the record between themselves. As a result, TTLs from this cache get further away from the true TTL the longer the record is cached. Although it is easy to see visually that a group of DNS responses came from the same distributed cache, it is not trivial for an algorithm to do so: the drift allows a single cached entry to persist past the end of its TTL epoch. We use a peak-finding algorithm to find clusters of TTL lines that are near the true TTL line, essentially combining those TTL lines into one per TTL epoch.
cation

5 EVALUATION

In this section, we describe experiments to evaluate how accurately Trufflehunter estimates how many caches in a PoP contain a particular domain. In these controlled experiments, we use RIPE Atlas probes to mimic the behavior of many geographically distributed users querying for a domain from their local PoP. This allows us to quantify Trufflehunter's error in estimating how many *caches* in a PoP contain a domain within a TTL epoch, and therefore its error in estimating a *lower-bound* on the number of users that have queried for a domain within a TTL epoch. However, it does not reveal Trufflehunter's accuracy in estimating the number of users over multiple TTL epochs (Section 6.1). The results of these experiments demonstrate that our estimates are consistent with Trufflehunter's goal of providing lower-bounded estimates of a domain's prevalence.

5.1 Simulating users with RIPE Atlas probes

Our goal in this experiment was to emulate users from multiple geographic locations requesting a domain from various public resolvers at diverse PoPs. Since Trufflehunter is deployed from the Ark infrastructure, we sought an orthogonal infrastructure that could provide this ability. RIPE Atlas probes are deployed in diverse locations; moreover, since this experiment did not have to be performed longitudinally, the restrictions with RIPE Atlas probes that prevented us from using the platform to run Trufflehunter long-term did not apply (such as low rate-limits towards destinations, low availability of RIPE Atlas credits etc.).

Choosing Atlas probes. We initially considered 956 RIPE Atlas probes located in the U.S. for this experiment, but realized that some showed evidence of having DNS requests hijacked by ISPs. If a request was hijacked, a request would arrive at the authoritative nameserver for the domain, but the cache that got filled as a result would not belong to the resolvers we were trying to measure and would therefore be invisible to Trufflehunter. We therefore designed an experiment to filter probes whose requests are hijacked.

We first asked each RIPE Atlas probe to request a subdomain, which contained its probe ID and its targeted resolver's name, from each of the four public resolvers. When we examined the IPs that requested these domains from our authoritative nameserver, we identified those that came from ASes that do not belong to the four public resolvers. We determined that the RIPE Atlas probes fell into three categories: some never had their requests hijacked (reliable probes), some always had their queries hijacked (unreliable probes), and some, to our surprise, seemed to have some of their queries hijacked but not others (suspicious probes). Further investigation revealed that the suspicious probes were in small ASes that had multiple Internet providers. Our hypothesis is that one of these providers hijacks DNS queries and the other does not. Changes in routing could lead the probe's queries to switch between the hijacking provider and the non-hijacking provider. Another possibility might be that some ISPs hijack only a sample of DNS queries, not all of them.

We filtered 40 probes in this step, leaving 916 probes that could participate in the experiment.

5.2 Measuring cache fills from Ark nodes

We used the RIPE Atlas probes chosen from the previous step to repeatedly place a domain (whose authoritative nameserver is controlled by us) in the caches of the public resolvers. We simultaneously attempted to detect the presence of the domain with Trufflehunter. The RIPE Atlas probes placed the domain in cache by making recursive queries in bursts of ten minutes. These bursts were repeated at three hour intervals for a total of 48 hours. Note that this allows every individual cache to be filled up to sixteen times. Trufflehunter began searching for the domain several hours before the experiment began, when it was not yet expected to be in cache, and did not stop searching until many hours afterward. We did not detect the domain in any cache outside the duration of the experiment.

Figure 8 shows the accuracy of our estimate of the number of filled caches per PoP for each resolver. We calculate our error by calculating the percentage of the caches filled by Ripe probes that were *missed* by Trufflehunter. Missed caches are the caches filled by Ripe probes minus the caches observed by Trufflehunter. We count the caches filled by Ripe probes by enumerating the responses that have the maximum TTL for that domain. Both Trufflehunter nodes and the Ripe probes identify the PoP they are currently routed to by using the location queries from Section 4. Our error ranges from underestimating by approximately 10% on average on OpenDNS, to approximately 50% on average on Quad9. The difference in underestimation rate depends on the caching architecture of the resolver: some resolvers are easier to measure than others. We note several interesting points from the results.

First, on OpenDNS, our method for eliminating error caused by measurements that have a granularity of one second appears to be reasonably successful. Recall that we eliminate the first and last TTL lines from each group, since we predict that they are likely to be composed only of measurements that are one second off from the true values (Section 4.2). We speculate that the remaining error is due to the fact that our error-removing technique is only a heuristic: there are a few lines that we remove that are not erroneous, and a few that we allow to remain that *are* erroneous.

We use the same technique on Quad9, but get very different results. Upon further investigation, it turns out that although Quad9 uses one DNS load-balancer called "dns-dist" for its frontend caches, they use two different software packages, Unbound and PowerDNS, for their backend resolvers. Unbound defaults to not answering queries with the Recursion Desired flag disabled; it returns a status of `REFUSED` [51]. Therefore, when a RIPE Atlas probe places its domain into the cache of a backend resolver that does not answer non-recursive queries, that record becomes invisible to Trufflehunter. We therefore underestimate the true number of filled caches at Quad9 PoPs. On most PoPs we underestimated by $\sim$50%, except for the BUR PoP, where our controlled users' queries appear to have coincidentally hit PowerDNS more than Unbound resolvers. Quad9's use of two backend resolver implementations is an interesting challenge, and one that limits the accuracy of our current technique.

On GPDNS, we appear to underestimate the number of filled caches by up to 45%, but this result may be due to the fact that the true number of filled caches is very hard to determine. Any request that missed in a frontend cache and hit in a backend cache presumably filled the frontend cache. Unfortunately, since it would not have caused a request to the authoritative nameserver, we cannot count the true number of filled caches with perfect certainty. We appear to have observed more caches than either RIPE Atlas's or Trufflehunter's queries would account for. However, since our estimate is consistently lower than the probable true value, this outcome is consistent with Trufflehunter's goal of providing a lower-bounded estimate of domain prevalence.

On most Cloudflare PoPs, Trufflehunter's error varies from 15% to -5%. Although each PoP's shared cache can only be filled once during a single TTL epoch, Trufflehunter cannot differentiate TTL epochs with perfect accuracy because the TTLs of the records drift over time (Section 4.2). IAD is the exception with significantly higher error (75%). For IAD, Trufflehunter only observed only the final four out of sixteen times the Atlas probes filled the cache. We suspect that there may have been a problem with the DNS location queries (Section 4) during this experiment. While both Trufflehunter and the Atlas probes recorded that they were using IAD, they may have been routed to different PoPs during the first twelve cache fills.

We also tested if any of a resolvers' PoPs appeared to use significantly different caching strategies compared to the resolver-specific strategies we identified in Section 3. We did not observe strong evidence of PoP-level inconsistencies.

In summary, Trufflehunter's cache enumerations underestimate by approximately 10-50% (excluding Cloudflare's IAD) depending on the resolver's cache architecture. Furthermore, since Trufflehunter consistently underestimates, our error does not prevent us from providing a *lower-bounded* estimate of cache occupancy. This allows Trufflehunter to fulfill its aim of enabling relative comparisons of rare domain popularity.

6 CASE STUDIES

In this section, we apply our cache snooping technique to examine the use of three categories of abusive Internet phenomena: stalkerware, contract cheating services, and typo-squatting domains.

Our goal in this section is to provide lower-bounded estimates on the number of *users* of various domains using our estimates of the number of *caches* filled with these domains. During a single TTL epoch, local caches prevent a user from filling more than one resolver cache (Section 2), so every filled cache represents at minimum a single user. But when attempting to enumerate users, we would like to present estimates across *multiple* TTL epochs (such as the number of users in a day). This is more challenging—it is hard to distinguish between one user making multiple queries in distinct epochs and multiple users making individual queries. If the rate at which users make requests can be determined, it is possible to deduce the number of individual users from the observed filled caches [57]. Unfortunately, the intervals at which a user visits a website are not usually deterministic. Without a vantage point that can grant insight into user behavior, estimating the number of unique visitors to a site is difficult. It is reasonable to assume that a user will not make multiple requests for a domain within a single TTL epoch, because of the caching behavior of operating systems and browsers (Section 2). But the number of unique website visitors cannot be estimated at time scales larger than a TTL epoch. We therefore present our estimates of website traffic in the form of web requests per day, summed over all resolvers and PoPs.

However, domains that are associated with *applications* instead of websites are often accessed automatically, without user interaction, at regular time intervals. Such regularity provides the opportunity to estimate the number of unique application users with better accuracy than we can estimate unique website visitors. Unfortunately, this insight still does not allow us to distinguish between unique users in multiple TTL epochs. As a result, we use the maximum number of users ever observed during a single TTL epoch as a conservative lower bound estimate of the number of users of a given application. This "maximum users per epoch" metric sums users observed across all PoPs and resolvers during that epoch. We assume that an individual user is unlikely to make DNS requests to either multiple PoPs or multiple resolvers during one TTL epoch.

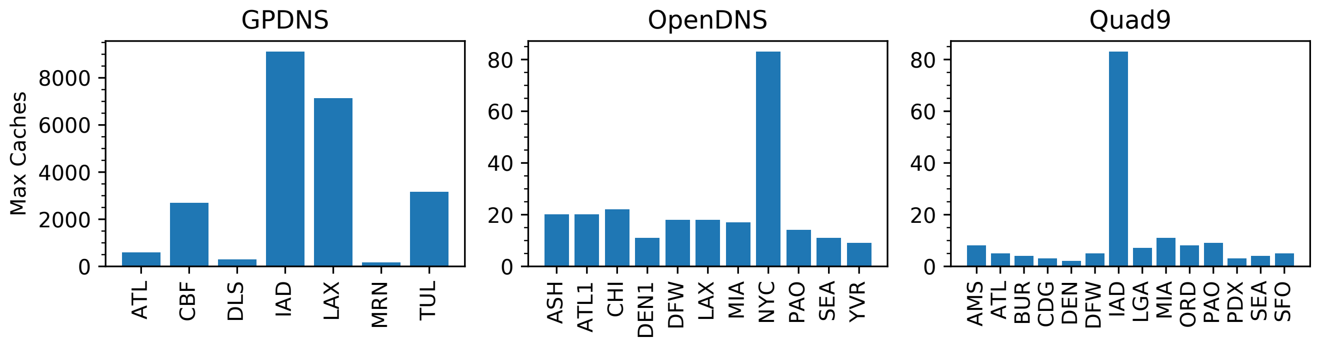


Figure 9: Maximum caches observed per PoP per resolver

6.1 Limits on observed users

Recall that our lower bound estimates of the number of users of a domain depend upon the number of caches at each PoP that can potentially contain a domain. If there is only a single shared cache at a PoP (as in Cloudflare's case, for example), we can only estimate at most one user per TTL epoch at that PoP. However, as the number of available caches increases, so does the opportunity to observe more users filling caches.

Trufflehunter can observe at most one user or web request per cache per TTL epoch. We can quantify our error in estimating how many caches have been filled over time (Section 4.2), but we cannot estimate how many more users have accessed a cache after it has been filled by the first user, until the cache expires and is refilled. This is the reason that Trufflehunter can only provide a lower-bounded estimate of the users of an application or visitors to a domain.

We used the data collected by Trufflehunter between March 6, 2020, and May 30, 2020 (per the methodology described in Section 4.1) for the domains in our study to estimate the number of individual caches that each PoP contains.[6] We counted the maximum caches that Trufflehunter saw during this period filled with any of the domains we studied during any single TTL epoch. Figure 9 shows these results. We do not include Cloudflare in Figure 9 since the maximum number of caches in any PoP is always one due to Cloudflare's use of a single shared cache per PoP. We note that GPDNS appears to have thousands of caches, consistent with our model of GPDNS in Section 3. This allows us to observe thousands of users per TTL epoch at each GPDNS PoP. On the other end of the scale, Cloudflare only has one cache per PoP. We do note, however, that although Cloudflare has the fewest caches per PoP, it has the most PoPs in the U.S. out of the resolvers we studied (46). This allows us to observe a non-trivial number of Cloudflare users across the U.S., and Cloudflare users do contribute to our total estimates of users (Figure 10). We also observe that across resolvers, larger PoPs like IAD and NYC have more caches, as might be expected.

[6]We elided a handful of days from our data because a small number of Ark nodes constructed some queries incorrectly on these scattered days, and may have poisoned the resolvers' caches by placing domains in them.

6.2 Stalkerware

We first apply Trufflehunter to estimate the prevalence of stalkerware. The term "stalkerware" covers a wide range of software used in the context of intimate partner violence (IPV). It is installed by the abuser onto the target's device, usually a cell phone. Apps vary widely in their range of capabilities, which can include tracking location, recording messages sent by text or other messaging apps, recording audio of phone calls and ambient sound, spoofing texts to the target, and more. The abuser then accesses the target's information by visiting an online dashboard, which is updated regularly by the app. Stalkerware broadly falls into two categories: "dual-use" apps, designed for a benign purpose and repurposed as spyware, and "overt" apps, which hide their presence on the target device and often have more dangerous capabilities than dual-use apps. Some are explicitly marketed for catching an unfaithful partner or spouse, although this messaging recently appears to have become more subtle or disappeared entirely [41, 42]. Since even overt applications are now advertised for legal or legitimate uses such as parental control, it must be noted that we have no way to tell whether Trufflehunter is observing stalkerware used in the context of IPV, or stalkerware installed for other reasons.

In the context of IPV, previous research has taken a clinical approach to studying stalkerware, and has uncovered little evidence of the use of overt applications. Most digital surveillance previously uncovered appears to be facilitated by either dual-use apps or misconfigured settings [13, 28].

However, a clinical approach has limited scalability: the researchers were able to speak with fewer than fifty IPV survivors. In contrast with clinical studies, a quick Google search reveals dozens of overt stalkerware applications, as well as anecdotes and articles describing what targets of the overt applications' capabilities have gone through. The contrast begs the question: how many people have overt stalkerware installed on their phones? Even if overt stalkerware is not the most common vector for digital stalking, it is a dangerous phenomenon worth understanding in more detail.

Our application user measurement technique is uniquely suited to measuring stalkerware because it leverages the knowledge that some apps make requests automatically at well-defined intervals. Stalkerware apps often exhibit this behavior, without requiring any

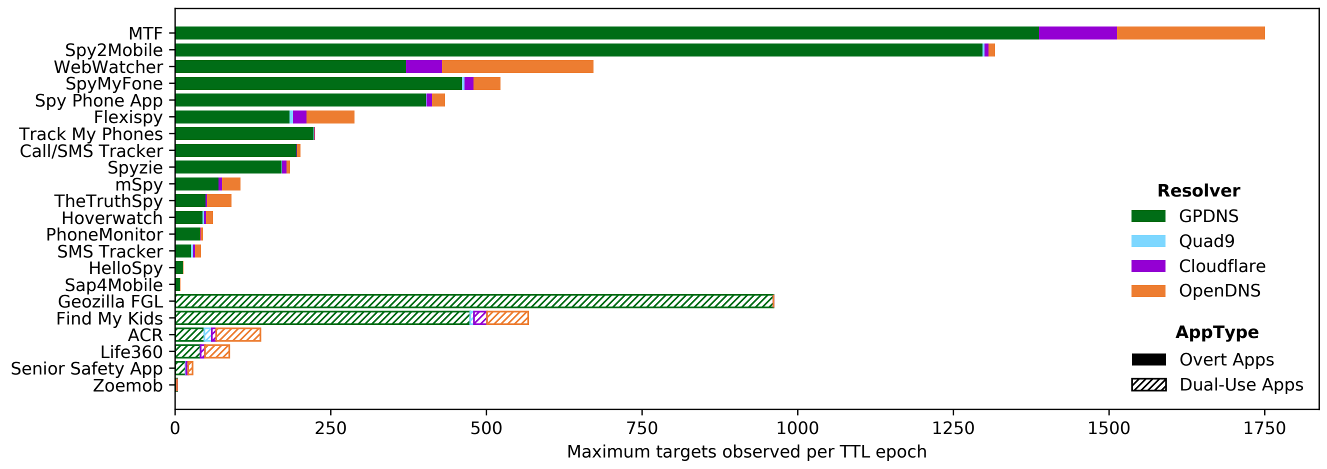


Figure 10: Stalkerware targets visible per TTL epoch

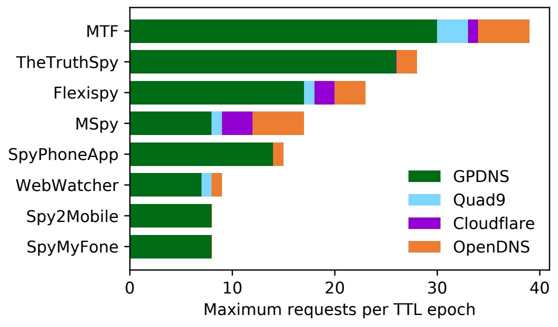


Figure 11: Web requests per TTL epoch for stalkerware dashboards. Note that not all stalkerware apps have dashboards.

user interaction. In fact, since they are installed on devices whose owners are unaware of their presence, this behavior is a necessity.

6.2.1 Profiling Stalkerware Applications. To model stalkerware request behavior, we examined the network traces of approximately 60 stalkerware apps on an Android Pixel phone. We found these stalkerware apps using a combination of searching the Google Play store and Google search results: previous work [13] indicates that spyware is surprisingly easy to find this way. We discarded around 40 apps that did not function correctly or were not usable as spyware. The remainder are a combination of dual-use and overt apps.

We then installed each application one at a time and recorded its network trace for a few hours. During each recording, we occasionally sent messages, made calls, installed apps, and simulated other behavior that the stalkerware claimed to track. We identified most of the domains that each app requested in this manner. We then installed all of the applications at once and recorded the phone's network activity during specific activities, such as sending a text, making a phone call, or rebooting the phone. The goal was to determine if any apps reacted to the target's activity on the phone by making network requests, or if the apps simply send information at regular intervals regardless of target behavior. We found that most apps did not appear to respond to target actions, with the exception of reboots: most apps made network requests directly after the phone was restarted. Finally, we recorded the phone's network traffic for a total of eighteen days, while attempting to use the phone like a normal device. Using this data, we determined which apps make requests at regular intervals. For the apps that do not, we make the most conservative assumption: that they make requests at most once per TTL epoch.

We make the simplifying assumption that targets have one device with stalkerware installed. We also assume that an individual device will not access more than one PoP or more than one resolver during a single TTL epoch. Therefore, for each domain used by an app, we first calculate the sum of all the caches we observed to be filled with that domain across all PoPs and resolvers for every TTL epoch. We then divide this sum of filled caches for every TTL epoch by the app request rate we calculated with our network traces. This yields the number of targets visible during each epoch. Finally, we calculate the maximum number of targets ever visible during one TTL epoch as a lower-bounded estimate of how many targets of stalkerware exist in the U.S.

6.2.2 Estimates. Figure 10 shows the maximum targets ever observed in a single epoch for 22 stalkerware apps between March 6, 2020, and May 29, 2020. Overt apps are shown as solid colors and dual-use apps have hatches. Each app is broken down by the resolver at which the targets were observed. Most targets are observed in GPDNS's caches, since GPDNS has the most caches per PoP. Interestingly, during the particular TTL epoch when the maximum users were observed, some apps, such as WebWatcher, SpyPhoneApp, and others, were not present in Quad9's caches. We speculate that this could be due to Quad9's use of Unbound resolver software, which prevents Trufflehunter from observing some filled caches.

We estimate that a minimum of 5,758 people are targeted by overt stalkerware in the U.S. today. The most popular overt app, Mobile Tracker Free, accounts for a third of observed targets (1,750). In contrast to most subscription-based overt stalkerware, Mobile

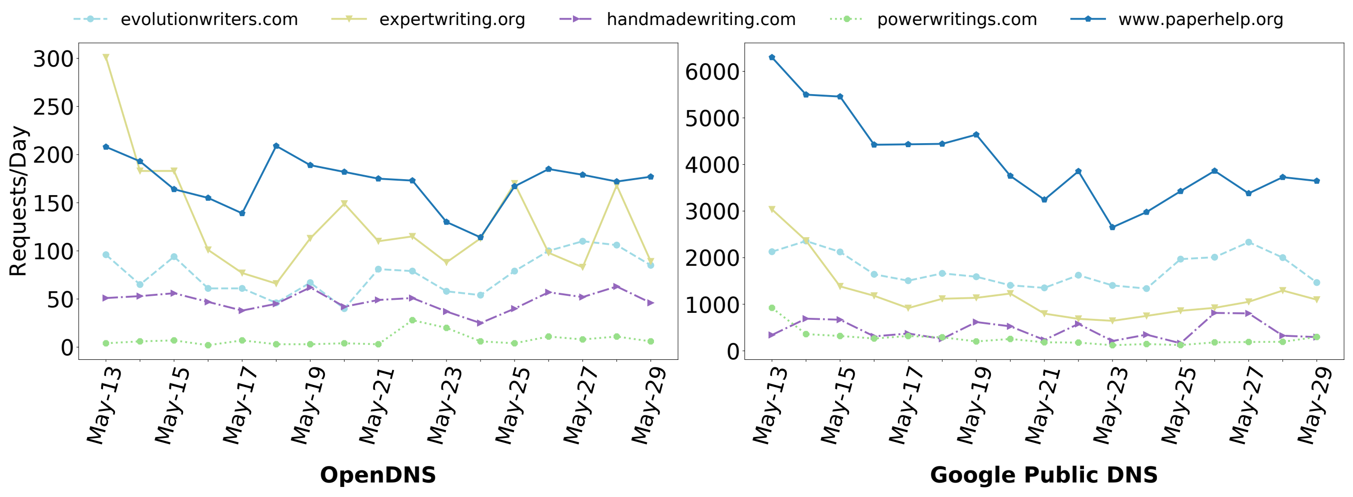


Figure 12: Web requests per day of contract cheating services

Tracker Free and Spy2Mobile (the second most frequently observed app) are both free, which likely accounts for their high popularity. Additionally, Spy2Mobile is one of only two overt apps we studied that is available on the Google Play store (the other is Call/SMS Tracker). All other overt apps must be downloaded from third-party websites.

We also used Trufflehunter to observe web requests for the dashboard websites that attackers use to view targets' information. Because web requests made by attackers do not exhibit periodic behavior, Figure 11 shows the maximum number of web requests ever visible during one epoch. We note that the popularity of app dashboards does not always correspond to the prevalence of the app, likely due to the features the app provides. For example, abusers might check Spy2Mobile's dashboard less frequently than MTF's because Spy2Mobile primarily provides location data, while MTF also records messages, phone calls, and more.

6.3 Contract Cheating Services

We next use Trufflehunter to examine users visiting "contract cheating" domains. Contract cheating services offer to complete students' homework assignments, projects, and in some cases entire classes for a fee. It is an increasingly popular method of cheating since it does not rely on plagiarism and is therefore more difficult to detect [15, 69]. The specific services provided include essay-writing services, "agency sites" that use auction models to match students to contractors who can complete assignments, copy-edit services, and more [38]. Since cheating is by its nature something that students are reluctant to admit to, it is difficult to measure using indirect means such as surveys.

We identified a set of ten popular contract cheating websites based upon search results and online discussions and recommendations. From May 3–29, 2020, we used Trufflehunter to track activity to these sites. Figure 12 shows the daily sum of web requests observed over time across the two resolvers with the most activity. Interestingly, we note a decrease in some services towards the end of May, perhaps because schools and universities that use semester systems were transitioning to summer break.

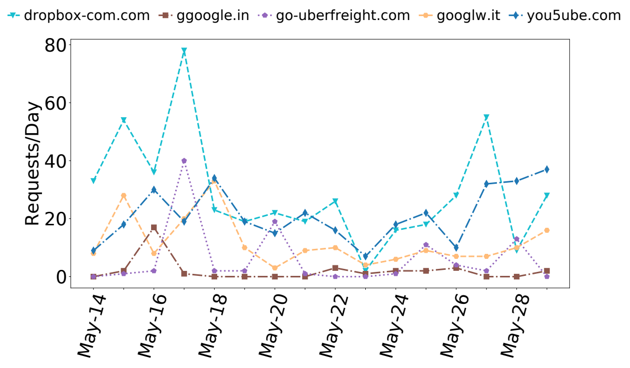


Figure 13: Web requests per day for typo-squatting domains on GPDNS

6.4 Typo Squatting Domains

Finally, we use Trufflehunter to estimate users visiting typo-squatting domains. Typo-squatting is the abusive practice of registering a domain similar to a popular domain so that users will be tricked into mistyping or clicking on it. Some forms of typo-squatting simply seek to show unwanted advertisements to the user, which is irritating but not generally harmful. Others distribute malware, or imitate children's websites and redirect to adult websites, or trick users into entering credentials which are then stolen [70].

To find typo-squatting domains that were likely to be active, we first tried to resolve the typo squatting domains listed in Tian et al.'s study [68]. We attempted to remove domains that did not appear malicious at the time of the study, such as paypal-cash.com, which is now an apparently legitimate domain owned by PayPal. Figure 13 shows the snooping results for the five most-accessed typo-squatting domains from May 3–29, 2020, on GPDNS, the resolver that showed the most activity. Given that these domains were in use two years before our study, it would not be surprising if all were on blacklists. We found that some domains were still surprisingly active, with dozens of resolutions per day.

We also looked for several domains used by hack-for-hire services in spear phishing attempts [46]. Of all of these, 18 still resolved to an IP address as of May 2020, and only one made an infrequent appearance in any resolver's cache.

7 RELATED WORK

Various aspects of recursive DNS resolver behavior, including caching [32, 33, 48], client proximity [6, 14, 44, 53], and vulnerabilities [29, 63, 66] have been studied. We focus our attention on studies that investigated caching behavior in public DNS resolvers and prior applications of cache snooping.

With public DNS recursive resolvers increasing in popularity [11], their caching and load-balancing behavior has received attention from the community. Callejo et. al. observed that public DNS resolvers including GPDNS, OpenDNS, Level3, and Cloudflare are responsible for 13% of the DNS requests in their online-advertising-based measurement campaign [11]. Public DNS resolvers use anycast [49] and can be present at multiple PoPs [17]. Some studies have observed that public resolvers have multiple caches for load-balancing [36, 64], which can be fragmented [2, 49, 71]. While these studies have investigated different aspects of the caching behavior of public resolvers, ours is the first to enable DNS cache snooping on them.

Several studies have used DNS cache snooping to measure various domains. Wills et al. used this technique in 2003 to measure the popularity of various web domains [72] and in 2008, two other studies used DNS cache snooping for similar purposes [4, 57]. Rajab et al. measured the relative footprints of various botnet domains [1, 58], and Kührer et al. used cache snooping to analyze which "open" resolvers found in an Internet-wide scan were actively providing service to clients [34]. All these studies assume that the resolvers they are probing have a single cache; our work has demonstrated that this assumption is no longer valid, especially for public DNS resolvers. Since these efforts did not focus upon potentially sensitive domains, they were able to probe the caches of arbitrary "open" resolvers. However, recent work has shown that millions of open resolvers are misconfigured residential devices that are unintentionally open [5, 16, 34, 62, 63], and are therefore not suitable for use in our study.

More recently, Farnan et al. used cache snooping on recursive resolvers belonging to VPN providers to analyze which domains are accessed through VPNs [20]. They target recursive resolvers belonging to VPN providers, which do not appear to have the complex caching architectures we observed in public resolvers.

Our work is the first to successfully demonstrate that public DNS resolvers can yield meaningful estimates of active users in a privacy-conscious way due to their underlying caching properties.

8 ETHICS

Since DNS cache snooping can reveal if a domain was recently accessed by the users of a DNS resolver, some ethical questions arise that we address below.

First, if a DNS resolver is used only by a few users, cache snooping may identify with fine granularity which domains these users accessed, impinging upon their privacy [25]. We avoid this issue by targeting our measurements only at large, public resolvers with thousands of users. Doing so allows us to measure how often anonymous users access rare domains and yet learn little that could aid in deanonymizing individual users. We refrain from probing caches of other "open" resolvers, since these are often misconfigured residential devices that may be serving only a few users [62, 63].

Second, domain names could contain user identifiers that potentially enable identifying a user's activity in a resolver. For example, a service may embed usernames into a unique service subdomain that may be periodically requested from a user's device. While a potential side channel for a resolver of any size, in our work we do not probe any domain that contains user-specific identifiers, and no individual user's information is exposed by our measurements.

Third, some users may be motivated to use large public resolvers because of the increased resilience to cache snooping they seemingly provide. Since the technique we describe in this paper identifies a side channel using the combination of rare applications and the caching architecture of large resolvers, we will be notifying the public resolvers of our findings.

Finally, the applications that we use in our case studies are by their nature sensitive. Since we are not able to identify individual users, though, we cannot associate the use of sensitive applications with any particular user. At most we can identify that these applications are in use at the coarse granularity of a large geographic region served by a PoP.

9 CONCLUSION

This paper introduces Trufflehunter, a privacy-preserving DNS cache snooping that models the caching architecture of public resolvers to provide lower-bounded estimates of cache occupancy. Applied to four large public DNS resolvers, Trufflehunter achieves a 10% to -50% error. Trufflehunter may be applicable to more distributed public DNS resolvers than the four we studied in the paper. Indeed, we observed the same caching strategy in Quad9 and OpenDNS. Trufflehunter provides lower-bounded estimates of domain usage, therefore it is best suited for relative comparisons of rare domain popularity, rather than for estimating an absolute number of users per domain. To demonstrate this capability, we showed how to estimate the prevalence of rare and sensitive applications on the Internet, which are otherwise difficult to measure from a third-party perspective.

ACKNOWLEDGEMENTS

We thank our shepherd John Heidemann and the anonymous reviewers for their insightful suggestions and feedback. We also thank Young Hyun and KC Claffy at CAIDA for providing access and support for their ARK infrastructure. We are also grateful to Cindy Moore for managing software and systems used in this project. This work was supported in part by National Science Foundation grants CNS-1629973 and CNS-1705050, Department of Homeland Security grant AFRL-FA8750-18-2-0087, the Irwin Mark and Joan Klein Jacobs Chair in Information and Computer Science, and generous support from Facebook and Google.

REFERENCES

[1] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. 2006. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *Proc. ACM Internet Measurement Conference (IMC)*.

[2] Bernhard Ager, Wolfgang Mühlbauer, Georgios Smaragdakis, and Steve Uhlig. 2010. Comparing DNS Resolvers in the Wild. In *Proc. ACM Internet Measurement Conference (IMC)*.

[3] Ron Aitchison. 2011. *Pro Dns and BIND 10*. Apress.

[4] Hüseyin Akcan, Torsten Suel, and Hervé Brönnimann. 2008. Geographic Web Usage Estimation By Monitoring DNS Caches. In *Proc. International Workshop on Location and the Web (LOCWEB)*.

[5] Rami Al-Dalky, Michael Rabinovich, and Kyle Schomp. 2019. A Look at the ECS Behavior of DNS Resolvers. In *Proc. ACM Internet Measurement Conference (IMC)*.

[6] Rami Al-Dalky and Kyle Schomp. 2018. Characterization of Collaborative Resolution in Recursive DNS Resolvers. In *Proc. Passive and Active Measurement Conference (PAM)*.

[7] Fatemah Alharbi, Jie Chang, Yuchen Zhou, Feng Qian, Zhiyun Qian, and Nael B. Abu-Ghazaleh. 2019. Collaborative Client-Side DNS Cache Poisoning Attack. In *Proc. IEEE Conference on Computer Communications (INFOCOM)*.

[8] Marc Blanchet and Lars-Johan Liman. 2015. RFC 7720: DNS Root Name Service Protocol and Deployment Requirements.

[9] CAIDA. 2020. *Archipelago (Ark) Measurement Infrastructure*. https://www.caida.org/projects/ark/

[10] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitendra Padhye. 2015. Analyzing the Performance of an Anycast CDN. In *Proc. ACM Internet Measurement Conference (IMC)*.

[11] Patricia Callejo, Rubén Cuevas, Narseo Vallina-Rodriguez, and Ángel Cuevas. 2019. Measuring the Global Recursive DNS Infrastructure: A View From the Edge. In *Proc. IEEE Access*.

[12] Sebastian Castro, Duane Wessels, Marina Fomenkov, and Kimberly Claffy. 2008. A Day at the Root of the Internet. *ACM Computer Communication Review (CCR)* (2008), 41–46.

[13] Rahul Chatterjee, Periwinkle Doerfler, Hadas Orgad, Sam Havron, Jackeline Palmer, Diana Freed, Karen Levy, Nicola Dell, Damien McCoy, and Thomas Ristenpart. 2018. The Spyware Used in Intimate Partner Violence. In *Proc. IEEE Symposium on Security and Privacy (SP)*. 441–458.

[14] Fangfei Chen, Ramesh K. Sitaraman, and Marcelo Torres. 2015. End-User Mapping: Next Generation Request Routing for Content Delivery. In *Proc. ACM SIGCOMM*.

[15] Robert Clarke and Thomas Lancaster. 2006. Eliminating the successor to plagiarism? Identifying the usage of contract cheating sites. In *Proc. International Plagiarism Conference*.

[16] David Dagon, Niels Provos, Christopher P. Lee, and Wenke Lee. 2008. Corrupted DNS Resolution Paths: The Rise of a Malicious Resolution Authority. In *Proc. Network and Distributed Systems Security (NDSS) Symposium*.

[17] Wouter B. de Vries, Roland van Rijswijk-Deij, Pieter-Tjerk de Boer, and Aiko Pras. 2019. Passive Observations of a Large DNS Service: 2.5 Years in the Life of Google. (2019), 190–200.

[18] Selena Deckelmann. 2020. Firefox continues push to bring DNS over HTTPS by default for US users. https://blog.mozilla.org/blog/2020/02/25/firefox-continues-push-to-bring-dns-over-https-by-default-for-us-users/

[19] Frank Denis. 2012. Performance: How Long Does a Second Actually Last? https://dzone.com/articles/performance-how-long-does

[20] Oliver Farnan, Alexander Darer, and Joss Wright. 2019. Analysing Censorship Circumvention with VPNs Via DNS Cache Snooping. In *Proc. IEEE Security and Privacy Workshops (SPW)*.

[21] Rom Feria. [n.d.]. Hiding from Data Collectors. https://rom.feria.name/hiding-from-data-collectors-9485dcb93b22.

[22] Diana Freed, Sam Havron, Emily Tseng, Andrea Gallardo, Rahul Chatterjee, Thomas Ristenpart, and Nicola Dell. 2019. "Is my phone hacked?" Analyzing Clinical Computer Security Interventions with Survivors of Intimate Partner Violence. In *Proc. ACM Conference on Human-Computer Interaction*.

[23] Google. 2018. *Google Public DNS: Performance Benefits*. https://developers.google.com/speed/public-dns/docs/performance?hl=zh-cn

[24] Google. 2020. Google Public DNS FAQ. https://developers.google.com/speed/public-dns/faq#isp

[25] Luis Grangeia. 2004. *DNS Cache Snooping or Snooping the Cache for Fun and Profit*. Technical Report. Securi Team-Beyond Security.

[26] Yunhong Gu. 2014. Google Public DNS and Location-Sensitive DNS Responses. https://webmasters.googleblog.com/2014/12/google-public-dns-and-location.html.

[27] Ólafur Guðmundsson. [n.d.]. Introducing DNS Resolver, 1.1.1.1 (not a joke). https://blog.cloudflare.com/dns-resolver-1-1-1-1/.

[28] Sam Havron, Diana Freed, Rahul Chatterjee, Damon McCoy, Nicola Dell, and Thomas Ristenpart. 2019. Clinical Computer Security for Victims of Intimate Partner Violence. In *Proc. USENIX Security*.

[29] Amir Herzberg and Haya Shulman. 2013. Fragmentation Considered Poisonous, or: one-domain-to-rule-them-all.org. In *IEEE Conference on Communications and Network Security (CNS)*.

[30] Michael Horowitz. 2007. *OpenDNS provides added safety for free*. https://www.cnet.com/news/opendns-provides-added-safety-for-free/

[31] joenathanone. 2017. *Hacker News forum: Quad9 location request*. https://news.ycombinator.com/item?id=15712940

[32] Jaeyeon Jung, Arthur W. Berger, and Hari Balakrishnan. 2003. Modelling TTL-based Internet Caches. In *Proc. IEEE Conference on Computer Communications (INFOCOM)*.

[33] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. 2002. DNS Performance and the Effectiveness of Caching. In *Proc. IEEE/ACM Transactions on Networking*.

[34] Marc Kührer, Thomas Hupperich, Jonas Bushart, Christian Rossow, and Thorsten Holz. 2015. Going Wild: Large-Scale Classification of Open DNS Resolvers. In *Proc. ACM Internet Measurement Conference (IMC)*.

[35] Amit Klein and Benny Pinkas. 2019. DNS Cache-Based User Tracking. In *Proc. Network and Distributed Systems Security (NDSS) Symposium*.

[36] Amit Klein, Haya Shulman, and Michael Waidner. 2017. Counting in the Dark: DNS Caches Discovery and Enumeration in the Internet. In *Proc. IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*.

[37] Ignat Korchagin and Lennart Poettering. 2019. *Git commit: resolved: use Cloudflare public DNS server as a default fallback*. https://github.com/systemd/systemd/commit/def3c7c791e7918a889c2b93dee039ab77b3a523

[38] Thomas Lancaster and Robert Clarke. 2015. *Contract Cheating: The Outsourcing of Assessed Student Work*.

[39] Abner Li. 2018. Google's Public DNS turns '8.8.8.8 years old,' teases 'exciting' future announcements. https://9to5google.com/2018/08/13/google-public-dns-8-8-8-8-years-future-announcements/.

[40] Owen Lystrup. 2020. *OpenDNS Enforces Threat Intelligence at the Speed of Automatic*. https://umbrella.cisco.com/blog/opendns-custom-api-operationalizes-threat-intelligence

[41] Internet Archive Wayback Machine. 2018. Mobile Spy App for Personal Catch Cheating Spouses. https://web.archive.org/web/20180216084527/http://hellospy.com/hellospy-for-personal-catch-cheating-spouses.aspx?lang=en-US

[42] Internet Archive Wayback Machine. 2020. Catch Cheating Spouses With TheTruthSpy. https://web.archive.org/web/20200523174940/https://thetruthspy.com/catch-cheating-spouses-with-thetruthspy/

[43] Linux Programmer's Manual. 2020. *GetHostByName – Linux manual page*. https://www.man7.org/linux/man-pages/man3/gethostbyname_r.3.html

[44] Zhuoqing Morley Mao, Charles D. Cranor, Fred Douglis, Michael Rabinovich, Oliver Spatscheck, and Jia Wang. 2002. A Precise and Efficient Evaluation of the Proximity Between Web Clients and Their Local DNS Servers. In *Proc. USENIX Annual Technical Conference*.

[45] Xavier Mertens. 2017. *Systemd Could Fallback to Google DNS?* https://isc.sans.edu/forums/diary/Systemd+Could+Fallback+to+Google+DNS/22516/

[46] Ariana Mirian, Joe DeBlasio, Stefan Savage, Geoffrey M. Voelker, and Kurt Thomas. 2019. Hack for Hire: Exploring the Emerging Market for Account Hijacking. In *Proc. International World Wide Web Conference (WWW)*.

[47] Paul V. Mockapetris. 2020. Domain Names - Implementation and Specification. https://tools.ietf.org/html/rfc1035

[48] Giovane C. M. Moura, John Heidemann, Ricardo de O. Schmidt, and Wes Hardaker. 2019. Cache Me If You Can: Effects of DNS Time-to-Live. In *Proc. ACM Internet Measurement Conference (IMC)*.

[49] Giovane C. M. Moura, John Heidemann, Moritz Müller, Ricardo de O. Schmidt, and Marco Davids. 2018. When the Dike Breaks: Dissecting DNS Defenses During DDoS. In *Proc. ACM Internet Measurement Conference (IMC)*.

[50] Yu Ng. 2014. *In the World of DNS, Cache is King*. https://blog.catchpoint.com/2014/07/15/world-dns-cache-king/

[51] NLNet Labs. 2020. *Unbound configuration file*. https://nlnetlabs.nl/documentation/unbound/unbound.conf/

[52] OpenDNS. 2015. *FAQ: Why did Cisco buy OpenDNS?* https://www.opendns.com/cisco-opendns/

[53] John S. Otto, Mario A. Sánchez, John P. Rula, and Fabián E. Bustamante. 2012. Content Delivery and the Natural Evolution of DNS: Remote DNS Trends, Performance Issues and Alternative Solutions. In *Proc. ACM Internet Measurement Conference (IMC)*.

[54] Andreas Pitsillidis, Chris Kanich, Geoffrey M. Voelker, Kirill Levchenko, and Stefan Savage. 2012. Taster's Choice: A Comparative Analysis of Spam Feeds. In *Proc. ACM Internet Measurement Conference (IMC)*.

[55] Quad9. 2018. *Quad9 Enabled Across New York City Guest and Public WiFi*. https://www.quad9.net/quad9-enabled-across-new-york-city-guest-and-public-wifi/

[56] Quad9. 2020. *Quad9: Internet Security And Privacy In a Few Easy Steps*. https://www.quad9.net

[57] Moheeb Abu Rajab, Fabian Monrose, Andreas Terzis, and Niels Provos. 2008. Peeking Through the Cloud: DNS-Based Estimation and Its Applications. In *Proc. Applied Cryptography and Network Security Conference (ACNS)*.

[58] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. 2007. My Botnet Is Bigger Than Yours (Maybe, Better Than Yours): Why Size Estimates Remain Challenging. In *Proc. USENIX Workshop on Hot Topics in Understanding Botnets*.

[59] Tarcan Turgut Rohprimardho and Roland M. van Rijswijk-Deij. 2015. *Peeling the Google DNS Onion*. Technical Report.

[60] root-servers.org. 2020. Root Server Technical Operations Association homepage. https://root-servers.org/

[61] Chris Scharff. 2018. *Have problems with 1.1.1.1? *Read Me First**. https://community.cloudflare.com/t/have-problems-with-1-1-1-1-read-me-first/15902

[62] Kyle Schomp, Tom Callahan, Michael Rabinovich, and Mark Allman. 2013. On Measuring the Client-Side DNS Infrastructure. In *Proc. ACM Internet Measurement Conference (IMC)*.

[63] Kyle Schomp, Tom Callahan, Michael Rabinovich, and Mark Allman. 2014. Assessing DNS Vulnerability to Record Injection. In *Proc. Passive and Active Measurement Conference (PAM)*.

[64] Lior Shafir, Yehuda Afek, Anat Bremler-Barr, Neta Peleg, and Matan Sabag. 2019. DNS Negative Caching in the Wild. In *Proc. ACM SIGCOMM Conference Posters and Demos*.

[65] Redhat Customer Solutions. 2017. *systemd-resolved falls back to Google public DNS servers*. https://access.redhat.com/solutions/3083631

[66] Sooel Son and Vitaly Shmatikov. 2010. The Hitchhiker's Guide to DNS Cache Poisoning. In *Proc. International Conference on Security and Privacy in Communication Systems (SECURECOMM)*.

[67] RIPE NCC Staff. 2015. Ripe Atlas: A Global Internet Measurement Network. *Internet Protocol Journal* (2015).

[68] Ke Tian, Steve T. K. Jan, Hang Hu, Danfeng Yao, and Gang Wang. 2018. Needle in a Haystack: Tracking Down Elite Phishing Domains in the Wild. In *Proc. ACM Internet Measurement Conference (IMC)*.

[69] Mary Walker and Cynthia Townley. 2012. Contract cheating: a new challenge for academic honesty? *Journal of Academic Ethics* 10, 1 (March 2012), 27–44.

[70] Yi-Min Wang, Doug Beck, Jeffrey Wang, Chad Verbowski, and Brad Daniels. 2006. Strider Typo-Patrol: Discovery and Analysis of Systematic Typo-Squatting. In *Proc. USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*.

[71] Nicholas Weaver, Christian Kreibich, Boris Nechaev, and Vern Paxson. 2011. Implications of Netalyzr's DNS Measurements. In *Proc. Workshop on Securing and Trusting Internet Names (SATIN)*.

[72] Craig E. Wills, Mikhail Mikhailov, and Hao Shang. 2003. Inferring Relative Popularity of Internet Applications by Actively Querying DNS Caches. In *Proc. ACM Internet Measurement Conference (IMC)*.