On Measuring the Client-Side DNS Infrastructure

Kyle Schomp, Tom Callahan, and
Michael Rabinovich
Case Western Reserve University, Cleveland,
OH, USA
{kyle.schomp,tom.callahan,michael.rabinovich}@case.edu

Mark Allman
International Computer Science Institute,
Berkeley, CA, USA
mallman@icir.org

ABSTRACT

The Domain Name System (DNS) is a critical component of the Internet infrastructure. It allows users to interact with Web sites using human-readable names and provides a foundation for transparent client request distribution among servers in Web platforms, such as content delivery networks. In this paper, we present methodologies for efficiently discovering the complex client-side DNS infrastructure. We further develop measurement techniques for isolating the behavior of the distinct actors in the infrastructure. Using these strategies, we study various aspects of the client-side DNS infrastructure and its behavior with respect to caching, both in aggregate and separately for different actors.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Miscellaneous; C.4 [**Performance of Systems**]: Measurement Techniques

Keywords

Internet Measurement; Domain Name System (DNS)

1. INTRODUCTION

DNS plays a foundational role in today's Internet. From its initial function of providing the mapping between human-readable names (e.g., "amazon.com") and obtuse network-level addresses, it evolved to form a basis for building scalable and agile Web platforms. In particular, by changing name-to-address bindings dynamically and providing different bindings to different clients, Web sites can transparently distribute client load among replicated Web servers or redirect client requests from their own servers to content delivery networks (CDNs), while CDNs and similar platforms can direct incoming requests to specific nodes in the platform.

With the crucial role DNS plays, the complexity of the DNS infrastructure—especially the client-side query-resolving aspect—has increased dramatically. No longer are address lookups a simple matter of end devices querying a local DNS resolver, which in turn queries authoritative nameservers on clients' behalf. Rather, facilitated by the simplicity of the stateless and connectionless protocol, DNS has developed into a complex ecosystem often involving several layers of shared resolvers which can, in turn, peer with additional resolvers. The path a DNS query takes through this infrastructure is often complex and hidden. This complexity makes it difficult to understand the behavior of the resolving infrastructure and to attribute responsibility for distinct behaviors to the individual actors.

This study targets the above challenge and makes the following contributions. First, we develop a set of methodologies for discovering the client-side DNS infrastructure efficiently. Given the vastness of this infrastructure and a short lifetime of some of its actors [8,13], probing strategies that improve the rate of discovery can facilitate subsequent measurements. Second, we develop measurement techniques for teasing apart the behavior of the actors within the system, some of whom cannot be accessed directly. Third, we apply our methodologies and strategies to assess some aspects of the client-side DNS infrastructure and its behavior with respect to caching, both in aggregate and separately for different actors. Our key observations from this assessment include the following:

- We double, from 15 to 32 million, previous estimates of the number of open resolvers on the Internet.
- We find evidence of wide adoption of complex resolution topologies including large shared pools of resolvers at certain layers in the infrastructure.
- We observe that DNS queries frequently travel large distances *within* the resolving infrastructure—both in terms of geography and network delay. We find 20% of open resolvers experiencing at least 100 msec of delay before their queries leave the resolution infrastructure.
- To the best of our knowledge, we contribute the first assessment of how various actors treat the time-to-live (TTL) settings given by authoritative nameservers to set the behavior of DNS caches. Despite being a simple notion, we find that different actors handle the TTL differently. The overall effect is that in many cases the TTL is distorted before reaching the original requesting client. We find that only 19% of all open resolvers consistently return correct TTL values to all our probes. A 2004 study [17] reports a wide violation of TTLs by end-clients while a 2012 study from a client site with "honest" resolvers shows a much lower violation rate by clients [6]. We expand upon these studies by demonstrating not only which actors cause violation but also how they behave regarding the TTL setting and, thus, cause other actors to violate TTL.
- We assess the time an unused record stays in the cache of various actors within the resolving infrastructure, which in particular determines whether the TTL or cache capacity is

IMC'13, October 23–25, 2013, Barcelona, Spain.
Copyright 2013 ACM 978-1-4503-1953-9/13/10 ...$15.00.
http://dx.doi.org/10.1145/2504730.2504734 .

origin：dns 请求发送设备
odns: 从origin接收DNS请求的设备
rdns：和权威解析服务器请求的设备。

the cause of eviction. We find scant evidence of a general capacity limitation problem .

2. RELATED WORK

The over-arching methodology we use to study the DNS ecosystem—as developed in the next three sections—involves actively discovering and characterizing DNS resolvers that will answer queries from arbitrary hosts throughout the Internet. In this manner, we can determine how the client-side DNS infrastructure behaves with regard to a wide range of test queries. The closest related work to ours is [8], which scans open resolvers in order to assess answer rewriting occurring on DNS paths. That work further contributes the idea of building a mapping between resolvers found by probing Internet hosts and the resolvers that ultimately contact an authoritative DNS server. We build upon that technique to attribute DNS behavior to specific actors. Our work extends the probing methodology presented in [8] to effectively discover resolver pools and examines many resolver characteristics not discussed in that paper.

Efficiently scanning the IPv4 address space for service discovery (including DNS) while avoiding complaints is discussed in [13]. While [13] explores reducing the burden of probing, we focus on reducing the number of probes required without losing insight. Our additional methodological contributions are in probing strategies that (i) increase the discovery rate, (ii) identify pools of recursive resolvers, and (iii) soundly assess specific behavior of the various actors in the system.

We also consider this work related to [2], which performs DNS lookups from several vantage points in order to compare performance among various local resolvers and public DNS services. [2] finds that ISP-provided resolvers often outperform public DNS services in query latency. Another performance-centric study is [14], which characterizes the difference in observed DNS performance for common DNS names from a variety of vantage points. Additionally, [14] reports on the behavior of the DNS time-to-live, which we also explore in more depth in this paper. Finally, [22] reports on several facets of DNS servers, including security configuration and support for DNS protocol components such as DNSSEC [5].

Several studies [10, 18, 23] show that information gleaned from DNS resolvers may be used to measure various aspects of the Internet such as popularity of Web sites and inter-host delays. Our work supports these efforts by developing effective discovery strategies and showing the diversity of behavior in differing implementations.

Several prior studies consider the number of open resolvers on the Internet [13, 22], the distance between clients and their resolvers [4, 11, 15, 20] and TTL violations [6, 17, 21]. We contrast our findings with these studies throughout this paper.

3. CLIENT-SIDE DNS INFRASTRUCTURE

The architecture of the client-side DNS infrastructure varies across providers. The actors can be loosely grouped into three roles: (i) "ingress" servers that receive DNS queries directly from user devices, (ii) "egress" servers that directly communicate with authoritative DNS servers (ADNS), which maintain hostname to IP address mappings for their respective domains, and (iii) hidden servers that act as intermediaries between the ingress and egress but are not exposed to either clients requesting mappings or authoritative servers providing mappings. To avoid confusion, in the rest of the paper we use the following terminology to describe the various components of the client-side DNS infrastructure that is charged with obtaining a hostname to IP address binding from an ADNS. The actors are also illustrated in Figure 1.

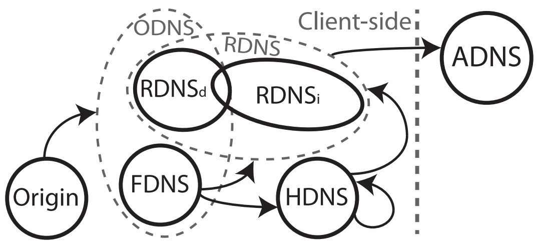

Figure 1: Structure of the client-side DNS infrastructure.

- Origin devices are either user devices or the sources of our DNS requests sent to probe the client-side DNS infrastructure.
- ODNSes ("open DNS") are ingress servers that accept requests from any host.
- RDNSes ("recursive DNS resolvers") are egress resolvers that communicate directly with authoritative DNS servers.
- FDNSes ("forwarding ODNS") refers to an ODNS that does not itself resolve a query, but rather forwards the request along to another resolver. The FDNS servers are a subset of the ODNS servers, or the ingress points that origin devices query.

转发器

- RDNS$_d$es ("direct RDNS") are RDNSes that are also ODNSes. In other words, an RDNS$_d$ is both an ingress and egress server.

间接解析器:代表转发器进行解析的权威解析器。

- RDNS$_i$es ("indirect RDNS") are RDNSes observed at the authoritative DNS server resolving queries on behalf of an FDNS. Note: The RDNS$_i$ and RDNS$_d$ sets overlap. That is, we find some RDNS servers that both (i) accept and resolve queries from arbitrary origin devices and (ii) act on behalf of a set of FDNS servers we detect in our survey.
- HDNSes ("hidden DNS") are servers which operate between FDNSes and RDNSes. Since these servers are neither ingress nor egress servers, they are invisible externally. While we cannot directly detect these servers, their existence is confirmed by DNS operators [9] and therefore we must keep them in mind as their actions may impact our results.

A typical example path through the maze of DNS-related devices has a client computer (Origin) starting the process by sending a DNS request to a home routing device (FDNS), which in turn forwards the request through a chain of zero or more HDNSes and ultimately to an RDNS$_i$. The RDNS$_i$ sends the request to the appropriate ADNS. Note that the RDNS$_i$ may cooperate with other RDNS$_i$es and subsequent requests from the same FDNS may be handled by a different RDNS$_i$. We call such structures "RDNS pools". We discuss RDNS pools in more detail in §6. This typical example accounts for nearly all of our experimental observations.

流程示例

4. METHODOLOGY OVERVIEW

In this section we sketch our general methodology and datasets. The specific methodology for each of our experiments is described in subsequent sections. Our measurements cover only a fraction of the Internet and therefore we must consider bias, namely, the degree to which the DNS infrastructure we discover and assess is representative of the broader Internet. We defer this question to §8—after we have further developed experiments that can be brought to bear on the question.

只扫了部分ip

ODNS: open dns

Scan	Format	Start	Dur. (days)	ODNSes	RDNSes
S_1	Random IP	2/29/12	17	1.09M	69.5K
S_2	Random IP	7/3/12	32	1.98M	72.6K
S_3	Random /24	8/5/12	17	841K	43.9K
S_4	Scan on First Hit	10/4/12	25	17.6M	72.1K
S_5	Rescan of S_3	11/16/12	9	892K	29.9K
S_6	Scan on First Hit	2/26/13	31	11M	65.8K

Table 1: Dataset characteristics

Criterion	No. ODNSes	% ODNSes
RomPager	258K	24%
Basic auth realm	265K	24%
PBL Listed by SpamHaus	566K	51%
PBL Listed by ISP	180K	17%
Wrong port	529K	48%
Total	849K	78%

Table 2: Home network device criteria

基于自己注册控制的域名进行扫描

Non-Interference With Normal Operation: While we are investigating the various components of the DNS ecosystem, we use our registered domain. Our probing rates are limited to insure we do not interfere with normal operation of any of the components of the system, and, although some of our techniques involve cache injection, all DNS requests are for subdomains of our own domain and we dot not interfere with any actual name-to-address bindings in use.

探测方法
1、注册一个域名
并部署次域名的权威解析器
2、利用 PlanetLab 100 个
节点请求注册域内的各种主机名。
3、通过自己的ADNS
中收集到的各种响应，
得到愿意处理probe响应 -- ODNS
4、

Discovering DNS Infrastructure: To examine the client-side DNS infrastructure, we must have an efficient method for finding both ODNSes and RDNSes. Discovering DNS infrastructure is a challenge because many of the components have policy restrictions preventing the acceptance of DNS requests from arbitrary hosts. Our basic discovery technique is an extension of the process described in [8]. 1 We registered a domain name[1] and deployed an ADNS for this domain. We then 2 leverage approximately 100 PlanetLab [7] nodes as our origins and randomly probe the IP address space with DNS requests for various hostnames within our domain. 3 By embedding the probed IP address in the hostname request and observing the queries arrive at our ADNS, we collect the IP addresses that are willing to handle our probes—thus discovering ODNSes. The IP addresses from which the queries ultimately arrive at our ADNS illuminate the set of RDNSes. Finally, since the ADNS has the addresses of both the RDNS and ODNS, we can associate FDNSes with the RDNSes they use for DNS resolution. Thus, we can elicit a response from an RDNS that will not respond to direct probes by indirectly probing via the FDNS.

Attribution of Behavior: When measuring DNS behavior, it is often necessary to identify the actor responsible for the behavior, e.g., when a violation of the DNS protocol is detected. A key contribution of this paper is measurement techniques to isolate FDNS behavior from RDNS and HDNS behavior. Through cache injection[2] on FDNSes, to which we found a sizable fraction of FDNSes are susceptible, we short-circuit HDNS and RDNS from processing a measurement probe. Therefore, any artifacts are the sole result of the FDNS. Similarly, we develop a technique of coordinated probing through two or more FDNSes to determine the behavior of a shared RDNS in near isolation from FDNS behavior. We validate the latter technique using the RDNS$_{di}$es —which we can probe both through an FDNS and directly—as ground truth of RDNS behavior. Estimating from our experiments, over 77% of RDNS$_i$es will not respond to direct DNS requests from external hosts and are assumed hidden from an outside observer; despite that, our technique provides the ability to assess their behavior.

ODNS Lifetimes: We note that during our measurements we find that ODNSes are often short-lived—with around 40% becoming unreachable within one day (see §5.1.1). Since the duration of our experiments is typically longer than one day, we rediscover the resolvers anew for each new experiment we discuss below. Hence, techniques for quick rediscovery are important. We describe these techniques below and use them to collect different datasets for different experiments, as summarized in Table 1. We describe the details of each scan as needed throughout the remainder of the paper. Our datasets are publicly available [19]. http://dns-scans.eecs.cwru.edu/ 数据集已隐藏。

5. METHODOLOGY DETAILS

We first turn our attention to discovering various components of the client-side DNS infrastructure. To facilitate our exploration of discovery methodologies, we use two datasets. The first dataset is from the S_2 scan in Table 1 and represents the probing of 255M unique random IP addresses using 267M DNS requests from 7/3/2012 to 8/3/2012. Our S_2 scan discovered 1.9M ODNSes and 73K RDNSes. The second dataset is from the S_3 scan in Table 1 and was collected between 8/5/2012 and 8/21/2012 using a methodology based on completely scanning random /24 IP address blocks. This scan represents a probing of 465K random /24 address blocks—11.9M IP addresses—via 121M DNS requests. The S_3 dataset includes 841K ODNSes and 44K RDNSes. The number of probes exceeds the number of IP addresses because some ODNSes use RDNS pools, which we attempt to discover through repeated probes to ODNSes (see §5.2 for details).

ODNSes appear to be mostly home network devices. During the S_1 scan of random IP addresses we gather detailed information about roughly 1.09M ODNSes. We find that 78% are likely residential networking devices as they meet at least one of the following criteria as shown in Table 2: (*i*) HTTP probes to the device show that the Web server reports itself as RomPager, which is a well-known software package for creating Web interfaces in embedded devices, (*ii*) HTTP probes to the device show the use of basic authentication with a realm indicating a likely home device (e.g., "3068 DSL-2641R"), (*iii*) the IP address is listed in the Spamhaus PBL, or (*iv*) the device replies to a DNS probe from a port other than the port to which the probe was directed. We speculate that the final criteria is caused by a low-end NAT device that is performing translation on its own packets. Together, these indicators provide supporting evidence that the ODNSes we discover are overwhelmingly low-end network devices residing in residential settings.

5.1 ODNS Discovery

The fundamental aspect of discovery is finding ODNSes since, as we will show, these are the windows into the client-side DNS infrastructure. Several projects leverage full scans of the Internet address space [1, 13] to understand the prevalence of open resolvers. However, we are interested not only with discovering the existence of ODNSes, but also with understanding their characteristics and behavior, which entails sending far more requests than discovery would dictate (as detailed in subsequent sections). Additionally, we find—as previously developed in the literature [8, 13]—the window of accessibility for ODNSes to be in general fairly short (see below). Therefore, we must do in-depth probing in conjunction

[1] dnsresearch.us

[2] A technique for inserting records into a DNS cache against the spirit of the protocol.

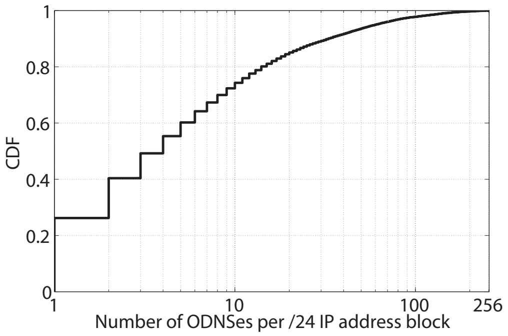

Figure 2: Distribution of ODNSes per /24 IP address block, excluding empty blocks.

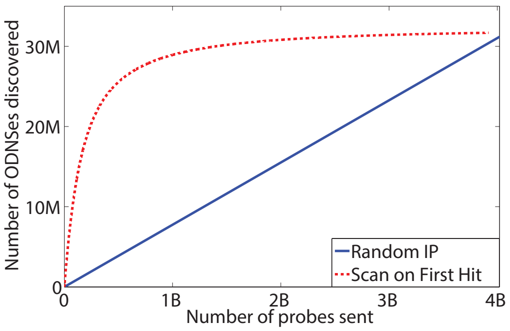

Figure 3: ODNS discovery rate versus DNS requests sent (extrapolation from the S_3 scan to the Internet scale).

with ODNS discovery as returning to the given address later may well be fruitless. Finally, our probing rate has to result in a manageable load on the ADNS—both the server itself and the hosting network—where ultimately the measurement traffic converges. For our ADNS, the resource constraints and desire to finish experiments in a reasonable time frame necessitates a partial scan. The key questions that arise from this choice involve (i) understanding the effectiveness of randomly probing arbitrary IP addresses with DNS requests in the hope of stumbling upon ODNSes, and (ii) whether there are probing strategies to improve the efficiency of this process.

Our first observation is that ODNSes are unevenly distributed throughout IP space. As sketched above, in S_3 we choose and probe random /24 address blocks. We find that only 14% of these blocks contain ODNSes. Further, as Figure 2 shows, the distribution of ODNSes among the blocks that have some ODNSes is uneven. We find a small number of "dense" blocks with many ODNSes. For instance, the top 10% of the address blocks each contain over 30 ODNSes while 40% of the blocks have no more than two ODNSes. The average across all blocks with at least one ODNS is approximately 13 ODNSes per /24 block.

Discovery within such a sparse address space requires extensive scanning. For collecting a sample of ODNSes with a partial scan, we examine two methods of ODNS discovery. The first method is a random scanning of IP addresses labeled "Random IP". The second method, "Scan on First Hit", acts like "Random IP" but, once an ODNS is discovered, proceeds to scan the entire /24 block in which the ODNS resides. This latter method utilizes the above observation of uneven ODNS distribution among /24 blocks to increase the ODNS discovery rate.

To compare the two methods fairly, we simulate both of them based on the same dataset from the S_3 scan using the following methodology. We consider the Internet's 2^{32} IP addresses divided into 2^{24} /24 blocks. We mark a random 14% of /24 blocks as "productive" —which as previously discussed is the fraction of /24 blocks found to contain at least one ODNS server—and in each productive block we mark a number of IP addresses as ODNS according to the distribution from Figure 2. "Random IP" is then simulated by selecting randomly without replacement from the full 2^{32} address range and counting the rate of discovering ODNS. For "Scan on First Hit", we again select an address randomly without replacement. If the selected address is an ODNS, we count not only this address, but also all addresses marked as ODNS in the encompassing /24 block and remove the block from the address pool for further selection.

Figure 3 shows the discovery rate for both methods. We find a drastically higher initial discovery rate using the "Scan on First Hit" strategy, which maintains its advantage for all scan sizes until the techniques converge to discover the entire set of ODNSes with a complete scan. The discovery rate of "Scan on First Hit" decreases over time. The reason is that the more dense a /24 address block is the higher the probability of finding an ODNS; therefore, dense /24 address blocks have a greater chance of being discovered early. The purely random scan shows steady progress across the entire scan but is overall less productive for limited scans. As noted above, only 14% of the /24 blocks contain ODNSes. So, the random scan misses opportunities to learn about the "neighborhood" when finding an ODNS and chances are that neighborhood is populated with additional ODNSes.

While the "Scan on First Hit" strategy discovers more ODNSes with fewer probes, it has a downside in that it introduces a bias in the set of discovered ODNSes. Blocks with higher concentrations of ODNSes have a greater chance of being discovered, thus biasing the resulting dataset towards ODNS in well-populated address blocks. Thus, when using this efficient discovery method, one must consider implications of its bias. We consider effects of this bias on our results in §8.

5.1.1 Rediscovery and Whitelisting

ODNSes have previously been found to be short lived [8,13] and we confirm these results. In our S_6 scan conducted from 2/26/2013 through 3/28/2013 we repeatedly probe discovered ODNSes for a period of 1M seconds after discovery. Details of the S_6 scan and the intervals at which the ODNSes were probed are discussed in §7. As shown in Figure 4, 40% of the discovered ODNSes answer queries for no more than one day and 80% of the ODNSes cease answering queries within one week. As noted above, there is evidence that the ODNSes we find are predominantly home network devices. Therefore, we suspect that short ODNS lifetimes are due to DHCP lease expirations. Thus, we conclude that our lists of ODNSes become stale and biased quickly and for this reason we discover ODNSes anew for each phase of our study.

Rescanning can be an expensive and time consuming process. Fortunately, we find that ODNSes demonstrate a tight IP spatial cohesion: while an individual ODNS can be short lived, productive /24 blocks tend to remain productive. We rescanned the same /24 address blocks from the S_3 scan between 11/16/2012 and 11/24/2012, nearly three months after the S_3 scan ended; this scan is labeled S_5 in Table 1. We also find that 76% of the 67K produc-

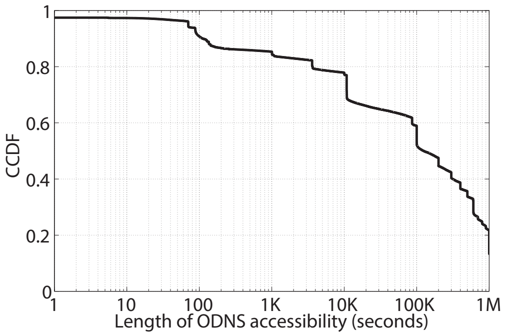

Figure 4: Distribution of the duration of ODNS accessibility.

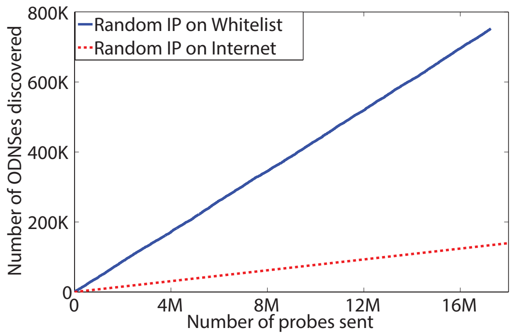

Figure 5: ODNS discovery rate using whitelisting in the S_5 scan compared to the discovery rate of the S_2 scan.

tive /24 address blocks during the former scan remain productive during the repeat scan. This spatial cohesion over time enables the use of "whitelisting" to rescan just those /24 address blocks which were previously productive. Using the same simulation methodology we employ to explore Random IP vs. Scan on First Hit above (Figure 3) we study re-scanning previously productive /24 blocks. Figure 5 shows the discovery rate for rescanning the 67K productive /24 address blocks from the S_5 scan using Random IP—i.e., scanning random IP addresses from the whitelisted /24 address blocks—in contrast to random IP selection from the entire Internet address space based on the S_2 scan. Clearly, rescanning using whitelisting is more efficient than random scanning. We also note that whitelisting may be used in conjunction with the "Scan on First Hit" strategy to generate a whitelist containing dense /24 address blocks. Rescanning using such a whitelist would likely have a much higher discovery rate than Figure 5 suggests.

5.2 RDNS Discovery

RDNS discovery provides more of a challenge than ODNS discovery for two reasons. First, unlike ODNS discovery, RDNS discovery is an indirect process whereby the characteristics and behaviors of the ODNSes may impact the process. Second, the RDNS resolver topologies are complex, unlike the ODNS population, which is by definition just a set of simple servers. In particular, an ODNS may forward DNS queries to a *pool* of resolvers, which may optionally utilize another layer of resolvers before the queries egress the infrastructure and are visible at our ADNS. For example, Google's public DNS utilizes a two-level topology that hashes the requested hostnames to particular egress resolvers to improve their cache effectiveness [9]. Unfortunately, we can only discover the egress RDNSes and do not have a technique for discovering HDNSes in the middle of the infrastructure.

转发器影响流程：
1）转发不解析请求

We use a two-pronged approach for RDNS discovery. First, for a given ODNS we send multiple DNS requests for hostnames within our domain in an attempt to spread those requests throughout the RDNS pool—if such exists. For this we use unique hostnames such that each request must move through the entire infrastructure and end up at our ADNS. Second, our ADNS returns a variable-length chain of CNAME records to queries from RDNSes[3]. We know from experience that the RDNS that sends a DNS request is often *not* the same RDNS that resolves the CNAME redirections. We use both these mechanisms until we stop discovering new members of the observed RDNS pool. Specifically, our strategy is as follows.

一个请求会经过：客户端-转发器-直/间接解析器-权威服务器

- When a first probe to a newly discovered ODNS arrives at our ADNS through a previously discovered RDNS, the ADNS responds with a special A record indicating that no new RDNS discovery has occurred.

A记录---RDNS系列

- However, when this first query arrives from a previously unknown RDNS, the ADNS responds to a query with a CNAME record of a new subdomain. After receiving the subsequent query for this new subdomain we repeat the process four additional times. When this batch of five CNAME queries leads to the discovery of at least one new RDNS then the entire process is repeated with five additional CNAMEs. This process continues until no new RDNS is found, at which point a special A record is returned to the client indicating that new RDNSes were discovered.

权威的A记录，即为间接解析器；

- When the A record returned—through the ODNS—indicates new RDNS were discovered, the client sends five more probes for distinct subdomains to this ODNS. Note that these subsequent probes may trigger a series of CNAME responses by our ADNS as described above. As long as the A record from the ADNS indicates new RDNS discovery, probing extends with another batch of five probes.

从权威发现间接解析器的过程

- When the A record returned to the client indicates that no new RDNSes were found, the discovery process terminates.

Our S_2 scan uses the above procedure. Furthermore, to enable exploration of alternate scanning strategies—as well as to discover RDNS pools in §6—our S_3 scan uses a modified version of the above procedure that triggers a new CNAME batch as long as new RDNSes are discovered *for the current ODNS* rather than consulting the full set of RDNSes from all probing.

We test this basic RDNS discovery mechanism with four ODNS probing strategies: "Random IP", "Scan On First Hit", and "Random /24 Block" described earlier, plus "Aborted Random Block", which is a scan of random /24 address blocks that terminates after the first ODNS in that block is found. The idea behind the last strategy is that the ODNSes in a /24 block will all share the same RDNS infrastructure and so the first ODNS will trigger the discovery of the lion's share of the RDNSes. The results for all four strategies reflect simulations driven by the dataset collected by the S_3 scan.

Figure 6 shows the discovery rates of our four methods. The "Scan on First Hit" method has a higher rate than the alternate strategies for two reasons. First, we find that ODNSes within the same /24 address block do not all use the same RDNS or RDNS pool—contrary to our intuition. Therefore, learning about the "neighborhood" is beneficial not only for ODNS discovery but also for RDNS discovery. This accounts for "Scan on First Hit"

[3] A CNAME record indicates a "canonical" name for the hostname queried. On receiving this record, the resolver will issue a new query for the name contained in the CNAME record.

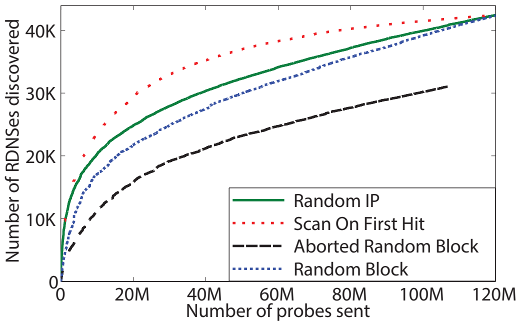

Figure 6: RDNS discovery rate versus DNS requests sent, simulated from the S_3 scan.

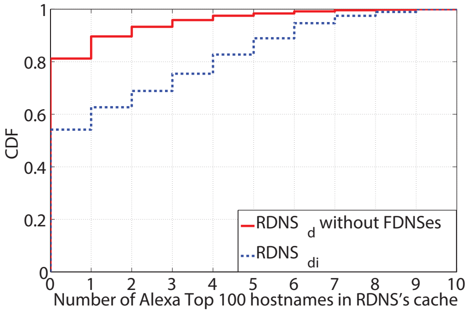

Figure 7: Number of the Alexa top 100 Web sites in the caches of RDNSes.

achieving a higher RDNS discovery rate than Random IP. Second, because the vast majority of /24 blocks do not contain any ODNSes, much of the scanning in Aborted Random Block and Random Block is wasted. We note that Aborted Random Block was unable to discover 13K of the 43.9K RDNSes within the S_3 dataset. The undiscovered RDNSes were not reachable through the first ODNS found within each /24 block. The "Scan on First Hit" technique provides the best ODNS and RDNS discovery rate in terms of scanning infrastructure cost and time.

5.2.1 *RDNS$_d$ Evaluation*

The RDNS$_d$es deserve special attention here. These servers have been counted during both ODNS and RDNS discovery. Yet, it is unclear if all RDNS$_d$es serve clients. They could, for instance, be misconfigured authoritative DNS servers that happen to be willing to accept external queries for external domains as discussed in [10]. We find that 51% of the RDNS$_d$es in the S_2 scan are used by at least one FDNS, i.e., are in fact RDNS$_{di}$es. The remaining 49% could be resolvers which might be accessed by origins directly, or whose client FDNSes are hidden from our scans' view or have been missed by our scans.

To determine if the 49% (17K) of RDNS$_d$es which are *not* used by any FDNSes in our dataset from the S_2 scan are actually acting as resolvers for some client population we query them for the top 100 Web sites as listed by Alexa [3]. If the RDNS$_d$es are resolvers, then they are likely handling DNS requests from their clients for some of these Web sites. Therefore, some of these popular hostnames should be in the RDNS$_d$es' caches. We detect if a record is in the cache by sending a DNS request for the hostname to the RDNS$_d$ and comparing the returned time-to-live (TTL) value with the TTL we expect to be set by the Web site's ADNS—which we established separately. A TTL value in a DNS response that is less than the ADNS assigned TTL indicates that the Web site's record is in the RDNS$_d$'s cache, suggesting that some real client previously requested the record. Figure 7 shows the distribution of the number of popular Web site records that appear to be in the caches of RDNS$_d$es *without* FDNSes and in the caches of RDNS$_{di}$es. Although we later show that RDNSes are prone to inaccurate reporting of TTLs, the difference between the two curves indicates a difference in the behavior of the two sets of RDNSes. We opt to remove RDNS$_d$es without FDNSes from our analysis since their purpose is unclear. Instead, we focus the remainder of our study upon RDNS$_i$es which have a clear purpose within the client-side DNS infrastructure.

5.3 Techniques for Untangling Behavior

We now discuss techniques we developed to tease apart the behavior of FDNS from RDNS.

5.3.1 *Measuring FDNS*

As we previously note, the vast majority of ODNSes we discover are in fact FDNSes (over 95% across all datasets). Gaining an understanding of FDNSes in isolation from the remainder of the client-side DNS infrastructure is a challenge. Fortunately, we find that a fraction of FDNSes allow a primitive form of cache injection which we leverage to gain insight into the behavior of this FDNS subset. Specifically, these FDNSes do not perform any of the following security checks on DNS responses which could prevent cache injection: (i) change and/or verify the transaction ID, (ii) verify the source IP address, and (iii) verify the destination port number. The absence of such checks makes it straightforward to follow a request to an FDNS with an acceptable response which only involves the FDNS and not the rest of the infrastructure.

Hence, we can study these FDNSes in isolation from HDNSes and RDNSes using the following procedure. We begin by sending a DNS request for a hostname within our domain to the FDNS under study and then immediately issue a DNS response for the same hostname to the FDNS that binds the requested name to IP address X. On the other hand, when the request arrives to our ADNS in a normal way, the latter answers with a response containing IP address Y. Then, any subsequent requests made by our probing host that are responded to with IP address X must have come from the FDNS's cache and the FDNS is effectively isolated from the rest of the client-side infrastructure, which never touched record X. We stress that the FDNSes we study may be a biased set since the set only includes FDNSes that exhibit the cache injection vulnerability.

5.3.2 *Measuring RDNS*

Since typically we cannot query RDNS$_i$es directly, we utilize FDNSes as our window into RDNS$_i$ behavior. This, however, poses a problem as FDNSes may alter DNS requests, responses and caching phenomena, hence obscuring RDNS behavior. A second response for a domain name through a single FDNS may be returned from either the FDNS's cache or the RDNS$_i$'s cache, ambiguously. Fortunately, it is common to find multiple FDNSes which use the same RDNS$_i$. We leverage a general experimental strategy which requires at least two FDNSes per RDNS$_i$ to succeed—F_1 and F_2. While the exact details of the technique vary with different experiments and will be detailed separately, the general framework is as follows. We begin by requesting a unique subdomain of our domain from F_1. Our ADNS responds to this query with a randomly generated record, which should be cached

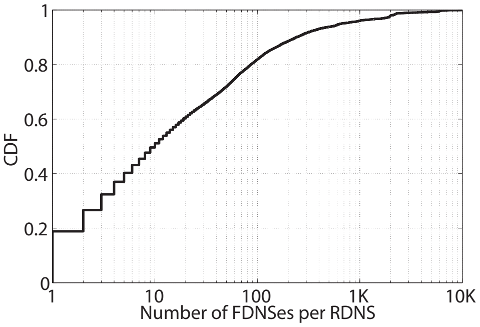

Figure 8: Number of FDNSes per RDNS$_i$ in the S_3 scan.

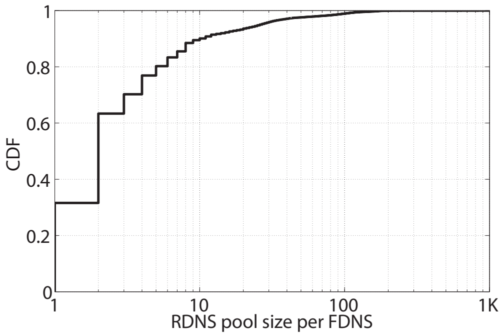

Figure 9: Distribution of the RDNS pool size for each FDNS.

at the RDNS$_i$ on the return path to F_1. Then, after a predetermined amount of time, we query for the same subdomain through F_2. If the RDNS$_i$ still has the record in its cache, the response from F_2 will match the response from F_1. In this case, we further know that the record is from the RDNS$_i$'s cache and not from F_2 because the request was previously unseen by F_2. If the record is no longer in the RDNS$_i$'s cache, the request will arrive at our ADNS, which will respond with a different record. In this way, we eliminate FDNS caching behavior when studying RDNS$_i$ caching behavior.

This technique relies upon discovering two FDNSes which use the same RDNS$_i$ at roughly the same time. Figure 8 shows the number of FDNSes per RDNS$_i$ in the S_3 dataset. Over 80% of the RDNS$_i$es are used by more than one FDNS and the coordinated probing technique has a chance of succeeding. Further, 50% of RDNS$_i$es appear with at least 10 FDNSes in the dataset, vastly increasing the chances of successful measurement.

FDNS behavior may still distort the measurement by altering records. We discuss ways to mitigate this problem—such as using all available FDNSes— in §7.

6. TOPOLOGY

In this section, we address the size and structure of the client-side DNS infrastructure.

6.1 Estimating Global ODNS Population

Extrapolating from our limited scans of IP space, we estimate that there are approximately 32M ODNSes on the Internet today. We arrive at this result from two independent scans. First, we find almost 2M ODNS within a set of 254.7M probed IPs in the "Random IP" scan S_2 where addresses are chosen randomly from the complete 2^{32} address space. Therefore, we estimate the population size as $2M/254.7M \times 2^{32} = 33M$ ODNSes across the Internet. Second, from the "Random /24" scan S_3, the fraction of productive /24 address blocks (those with at least one ODNS) is 0.141 and a productive block contains on average 13 ODNSes. Therefore, the ODNS population across the entire Internet is $0.141 \times 13 \times 2^{24} = 31M$ (a similar number is also obtained from simulations in Figure 3).

These estimates significantly exceed previous results of complete Internet scans [13] and estimates [22], which show around 15M responding DNS resolvers. One of our *partial* scans using the "Scan on First Hit" strategy directly identifies 17.6M ODNSes—more than found in previous full scans. Additionally, [1], a complete Internet scan, reports 33M open resolvers as of May 2013 which agrees with our estimate. The results show the population of ODNSes on the Internet has increased since previous studies.

6.2 RDNS Pool Sizes

The ODNSes we find in both our S_2 scan and S_3 scan utilize RDNSes in roughly 99% of the cases—i.e., they are in fact FDNSes. Moreover, approximately 70% of FDNSes use an RDNS pool in both scans. Per §5.2, we use repeated DNS requests and CNAME chaining triggered by per-ODNS discovery of a new RDNS to identify RDNS pools used by an FDNS. Figure 9 shows the size distribution of the discovered RDNS pools in the S_3 scan. The plot shows that 10% of FDNSes use RDNS pools consisting of more than 10 servers. Also, note that these pools can encompass multiple providers, e.g., an ISP's own DNS infrastructure and OpenDNS, which could occur when either the FDNS is configured to use both, e.g., one as the primary DNS server and the other as the secondary DNS server, or the ISP is utilizing some alternate DNS infrastructure for some queries.

6.3 Distance between FDNSes and RDNSes

As discussed in §5.1.1, ODNSes—and consequently FDNSes—are predominantly in residential settings. Many Internet platforms—notably content delivery networks—rely on the assumption that client machines are close to their RDNS in that the platforms direct the client to a nearby node within the platform based on the location of the client's RDNS. To test this assumption, we use a geolocation database [16] to calculate the distance between FDNSes and the RDNSes they employ for resolution. We perform this on the S_6 dataset. Figure 10 presents the distribution of these distances[4]. Since an FDNS may use several RDNSes, we plot the minimum, maximum, and mean distance between an FDNS and all of the RDNSes it utilizes. We find that FDNSes are often quite close to their RDNSes with the median being approximately 100 miles. However, one in ten FDNSes appear to be over 8K miles from at least one of their RDNSes; these FDNSes experience a high cache miss cost and potentially incorrect redirections by content delivery networks to nodes that are not nearby. Prior studies [4, 11] also consider the geographical distance between the client hosts and their DNS resolvers and report different results. In particular, [11] observes shorter distances between FDNSes and RDNSes, with 60% of clients to be within 17 miles from their resolvers, while [4] reports fewer client-resolver pairs at the low distance range, with only 25% of pairs being within 500 miles from each other, and also fewer extremely distant pairs, with just 5% of pairs more than 2000 miles apart. The differences with our experiment could be due to different vantage points. Both [11] and [4]

[4] The accuracy of our results rests upon the accuracy of the MaxMind geolocation database which varies by country. For example, MaxMind claims 81% accuracy within 25 miles for IP addresses inside the USA. See [16] for accuracy in other countries.

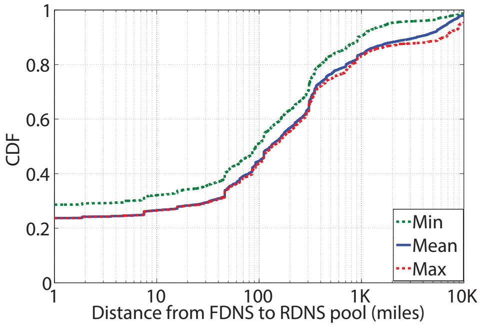

Figure 10: Distribution of the geographic distance from FDNS to RDNS.

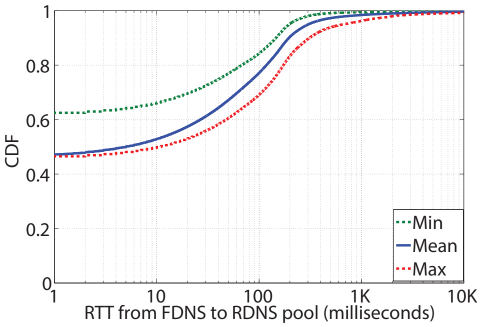

Figure 11: Distribution of round trip time between FDNS and RDNS.

instrumented a Web page and passively observed FDNS/RDNS pairs via DNS requests and subsequent HTTP connections, where as our study is an active scan which exhaustively explores all FDNS/RDNS pairs. Therefore, we discover FDNS/RDNS pairs that may not appear in prior studies.

Additionally, we found that some FDNSes respond to ICMP pings. In the S_6 scan, we obtained (1) the round trip time from our measurement point to the 22% of the FDNSes that were responsive to a ping and also (2) the round trip time between our measurement point and the respective RDNS through the FDNS for 6.3M FDNS/RDNS pairs. The later leverages our coordinated probing technique. We add a record to the RDNS' cache via a probe from the first FDNS. We then obtain the distance from our measurement point to the RDNS through the second FDNS by querying for the same record through the second FDNS. The difference between (2) and (1) is the round trip time in milliseconds between the second FDNS and the RDNS. We also repeat the process by swapping the roles of the two FDNS servers. We perform this measurement for each FDNS pair we discover using the same RDNS during discovery. Using this technique, we are able to obtain the round trip time from FDNS to RDNS for 5.6M FDNS/RDNS pairs across 1.3M unique FDNSes. In the case of multiple measurements per FDNS/RDNS pair, we choose the minimum delay value as it most accurately reflects the actual network delay between the FDNS and RDNS. We plot the results in Figure 11. The median round trip time is about 10 ms, however nearly 20% of the FDNSes experience delays in excess of 200 ms to at least one of their RDNSes.

7. CACHING BEHAVIOR

Caching aids the scalability of the DNS system and hides delay for hostname resolution. Additionally, DNS caching has important performance and security implications.

In terms of performance, DNS caching complicates Internet sites' traffic engineering ability because a single hostname-to-address binding may pin all clients behind a given resolver to the selected server for an extended period of time. Not only does this handicap sites' control over client request distribution but it also complicates the removal of unneeded infrastructure without risking failed interactions from clients using old bindings. DNS nominally provides sites with the capability to bound these effects by specifying a time-to-live (TTL) value within DNS responses to limit the amount of time recipients can reuse the response. However, recipients are known to disobey TTL [6, 17, 21]. With regard to security, caching determines the lingering effect of a successful record injection (e.g., using the Kaminsky attack [12]).

The extent of these phenomena depends on two inter-related aspects: how long a resolver keeps a record in its cache and how the resolver treats the TTL. We first explore the aggregate behavior of all components of the client-side DNS infrastructure, as this will be the view of the clients leveraging the infrastructure. We then use the measurement techniques described above to tease apart the behavior of the components in isolation to gain insight into which components are responsible for various behavior.

The results in this section come from the S_6 scan (see Table 1). The scan covers 79M IP addresses with 1.3B DNS requests from 2/26/2013 through 3/28/2013. The S_6 scan encompasses 11M ODNSes and 65.8K RDNSes—46K of which are RDNS$_i$es. The S_6 dataset uses the "Scan on First Hit" methodology to increase the ODNS discovery rate and thus the probability of finding multiple FDNSes which use the same RDNS$_i$ at roughly the same time. This assists with the coordinated probing strategy sketched in §5.3.2.

7.1 Aggregate Behavior

To investigate aggregate behavior of the DNS resolution infrastructure, we perform in-depth probing of 2.4M FDNSes during the S_6 scan. We did not test all of the FDNSes because of limitations in the amount of traffic our ADNS can handle. Therefore, we limit the number of FDNSes which can be concurrently assessed to 25K per measurement origin (PlanetLab node). When a measurement origin finds a new FDNS we skip in-depth measurement if 25K FDNSes are presently being assessed.

We examine how FDNSes and RDNS$_i$es behave when presented with DNS records with different TTL values: 1, 10, 30, 60, 100, 120, 1,000, 3,600, 10,000, 10,800, 86,400, 100,000, 604,800 and 1,000,000 seconds. For each TTL value, we re-probe at various intervals to determine whether the records are still available. We use intervals slightly below the TTL values (by two seconds), to check if the record is retained for the full TTL. We also use intervals slightly above the TTL values (by two seconds), to check if the record is retained *longer* than the TTL. For this experiment, our ADNS returns a random IP address. Thus, if subsequent DNS requests for the same hostname return the same IP address we know—with high likelihood—the request was satisfied by a cache somewhere within the resolving infrastructure, either at the FDNS, HDNSes, or RDNS$_i$.

Our first observation is that some of the responses arrive at the client with incorrect TTL values, i.e., different from those set by our ADNS. Furthermore, even when the client receives the correct TTL for the *initial* request for a hostname, we find *subsequent* requests result in responses with incorrect TTL values. We interpret the latter as due to the DNS actor distorting TTL not when con-

Behavior	Percentage of Measurements
Honest	19%
Lie on Initial	38%
Lie on Subsequent	9%
Constant TTL	7%
Increment TTL	1%

Table 3: Aggregate TTL Behavior

Expected (sec)	% <	% >	Mode Lie	
			Value	% of All Lies
1	0%	11%	10000	35%
10-120	$\leq$ 1%	$\leq$ 8%	10000	$\geq$ 37%
1000	1%	3%	10000	62%
3600	2%	2%	10000	51%
10000	5%	0%	3600	40%
10800	8%	0%	3600	27%
86400	16%	0%	21600	36%
100000	22%	0%	21600	27%
604800	22%	0%	21600	26%
1000000	64%	0%	604800	67%

Table 4: Aggregate TTL Deviations

veying the record back to the requester but when storing the record in its cache. Table 3 summarizes our results for aggregate behavior. In total, 19% of the FDNSes and their underlying infrastructure always report the correct TTL value. Meanwhile, 38% of the FDNSes respond with an incorrect TTL value to an initial request and 9% of FDNSes respond with an incorrect TTL value to the subsequent requests. The fact that at least 62% of FDNSes honestly report the TTL on the first DNS request will be important in studying RDNS$_i$es below.

Beyond changing the TTL of DNS records, we find some FDNSes fail to correctly decrement the TTL over time[5]. We find that 7% of the FDNSes return a constant TTL value, never decrementing. An additional 1% of the FDNSes actually begin to *increment* the TTL after it counts down to zero! Both these behaviors eventually result in a TTL which disagrees with the initial setting by the ADNS. Downstream devices—regardless of how they treat the TTL—may unwittingly use such records in violation of the intent of the ADNS.

Table 4 shows the TTL deviations we observe including those in response to both the initial and subsequent requests. The table shows the percentage of cases when TTL is less than ("< %") and greater than ("> %") the ADNS-assigned value. In addition, we show the most common TTL violation (mode lie) and the percentage of the lies the mode represents. For example, we find 11% of FDNSes deviate from a 1 second authoritative TTL. Further, 35% of the lies are for a TTL of 10,000 seconds. The prevalence of lies increases for both small and large TTL values. For instance, most resolvers appear to cap the TTL at one week (604,800 seconds).

We next consider how long a record remains cached and accessible in the client-side DNS infrastructure. Using repeated probes at the intervals mentioned above, we record the latest time at which a record is returned to our probing host. While it is possible that the record was still in some cache at this point and resolution merely took a different path through the infrastructure, our experiment reflects the behavior a user of the system would experience. Figure 12 shows the length of time records remain in some cache within the resolving infrastructure. We present results for records with TTLs of 30 seconds, a short TTL value similar to those used by content delivery networks (e.g., Akamai uses 20 seconds and Limelight uses 350 seconds). Additionally we use records with TTLs of 1 million seconds, a value chosen to determine how long the infrastructure will retain rarely used records. The "All" lines reflect the longest record lifetime we observe from a given FDNS. However, a number of FDNSes become unreachable in the course of the experiment. When this happens due to IP address reassignments as discussed in §5.1.1, the records may still be in the cache. Thus, the "All" line may be an underestimate of the length of time the infrastructure retains DNS records. Consequently, we report results separately for the 189K FDNSes that remain accessible throughout the experiment (the "Accessible" line). The true cache duration lies somewhere between these two lines.

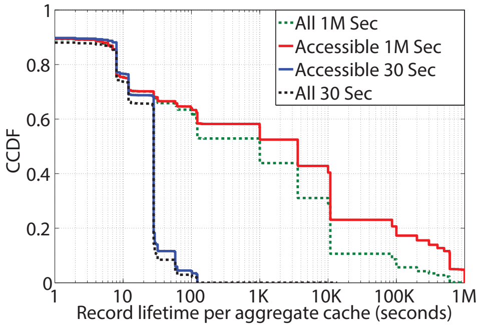

Figure 12: Aggregate cached record lifetimes.

Our results show that 90% of FDNSes and their supporting infrastructure retain records with a TTL of 30 seconds for no longer than the TTL—with 60% of the FDNSes retaining the record for the full 30 seconds. We find that 10% of FDNSes retain the record for longer than the TTL—with 4% retaining the record for over 100 seconds. This indicates that in general short TTLs are relatively effective in controlling DNS caching behavior. These results deviate from the findings from a 2004 passive study that shows TTL violations on the order of hours were not uncommon [17]. Furthermore, the records assigned a TTL of 1 million seconds show much longer retention, with 40% active for more than 1 hour. This indicates that short cache retention of the records with TTLs of 30 seconds is due to the TTL setting rather than cache capacity constraints.

7.2 FDNS Behavior

We now study the behavior of FDNSes in isolation by using the record injection technique described in §5.3.1. We find 683K FDNSes in the S_6 dataset (6%) accept our injected response records and were thus amenable to our measurement.

We utilize the same experimental technique as in §7.1 with the exception that we follow up the initial DNS request for a hostname with a DNS response that binds the hostname to IP address X. The same binding is never returned by our ADNS, so whenever X appears in a DNS response we know the request was satisfied from the FDNS' cache. If the DNS response contains an address other than X we know X is no longer in the FDNS' cache. We perform the experiment for the same TTL values and re-probe intervals as in §7.1.

Table 5 summarizes our general findings on FDNS' TTL behavior. First, 60% of the FDNSes never lie with respect to the TTL. However, we find that 12% of FDNSes lie in response to the initial request and 30% lie in response to the subsequent requests. As we

[5] A resolver is supposed to decrement the TTL to account for time spent in the resolver's cache.

Behavior	Percentage of Measurements
Honest	60%
Lie on Initial	12%
Lie on Subsequent	30%
Constant TTL	26%
Incrementing TTL	10%

Table 5: FDNS TTL Behavior

Expected (sec)	% <	% >	Mode Lie	
			Value	% of All Lies
1	0%	31%	10000	88%
10-3600	≤ 1%	19%	10000	≥ 95%
10000	1%	0%	60	92%
10800	19%	0%	10000	97%
86400	19%	0%	10000	97%
100000	19%	0%	10000	97%
604800	19%	0%	10000	97%
1000000	25%	0%	10000	75%

Table 6: FDNS TTL Deviations

note above, we interpret this latter behavior as due to the FDNS distorting TTL not when conveying the record back to the origin but when storing the record in its cache. We also determine that 26% of FDNSes report a constant TTL value without ever decrementing it. Finally, 10% of the FDNSes began to increment the TTL value upon decrementing it to zero.

Table 6 shows the TTL deviations we observe for FDNSes, including both initial and subsequent deviations. The table shows the same general trend as aggregate behavior but with more deviations at the low end of TTL spectrum and less prevalence of capping the TTL at 1 week. The number of cases where the authoritative short TTL is replaced by a 10,000 second value jumps by roughly 50% for the authoritative TTL of 1 second as compared to 10 seconds. Thus, the ADNS often will retain better control over FDNS caching by assigning a TTL larger than 1 second.

We now consider how long a record remains accessible in the cache of FDNSes. Using repeated probes at the intervals mentioned above, we record the latest time at which the injected record is returned to our probing host. Again, we present results for records supplied by our ADNS with the TTLs set to 30 seconds and 1 million seconds. Figure 13 shows how long the records are retained in the caches. See §7.1 for an explanation of the "All" and "Accessible" lines. For this experiment, the "Accessible" line contains the results for 22.5K (3.3%) of the 683K tested FDNSes.

The results show that roughly 40% of the FDNSes retain the record with a TTL of 30 seconds for longer than the TTL. Additionally, 28% of FDNSes hold the record for over 100 seconds, more than twice the TTL. This deviates from the aggregate results in Figure 12 indicating that FDNSes which allow cache injection are more likely to retain records past the TTL value than the general FDNS population. We find records with a TTL of 1 million seconds are retained for at least 10,000 seconds in 50% of the FDNSes.

7.3 RDNS$_i$ Behavior

We now turn to the behavior of RDNS$_i$es in near isolation from FDNSes. RDNS$_{di}$es are particularly convenient in that we can examine their behavior in perfect isolation since there is no other actor to interfere with our results. We have results for 4.9K RDNS$_{di}$es. We use the same experimental technique as in §7.1 with the same TTL values and re-probe intervals. Table 7 shows our general results while Table 8 shows the TTL deviations we observe. RDNS$_{di}$es do not exhibit either the constant TTL nor the pathological TTL incrementing behavior we observe in FDNSes and the aggregate data. However, there is more TTL distortion with virtually no consistently honest RDNS$_{di}$es.

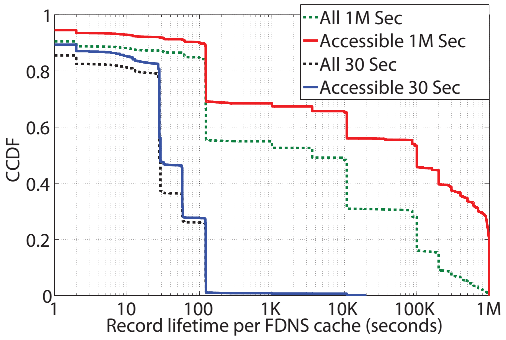

Figure 13: Distribution of record availability in FDNS caches for records with TTLs of 30 and 1 million seconds.

Behavior	Percentage of Measurements
Honest	2%
Lie on Initial	80%
Lie on Subsequent	18%
Constant TTL	0%
Incrementing TTL	0%

Table 7: RDNS$_{di}$ TTL Behavior

For RDNS$_i$es that do not respond to direct probes, we leverage the coordinated probing strategy to assess their violations. As described in §5.3.2, we begin by requesting a unique hostname via FDNS F_1. Our ADNS responds to this query with two IP addresses, one random and the second uniquely bound to the RDNS$_i$ from which the DNS request arrives. We assign the same set of TTL values as in §7.1. Both records pass through RDNS$_i$ on the return path to F_1 and may be cached for subsequent requests. Next, our experiment requests the same hostname via F_2. If the RDNS$_i$ still has the records in its cache and is leveraged by both F_1 and F_2, the random IP address from the response to F_1 will be re-used and hence the response to F_2 will show the same address.

However, at this point, any TTL deviations cannot be attributed to the RDNS$_i$, the FDNSes or even some HDNS in the path. To gain confidence in our attribution of TTL deviations to RDNS$_i$, we leverage two pieces of information. First, as noted in the previous two sections, a significant number of FDNSes are honest on the initial DNS response. We find 62% of FDNSes are honest on the

Expected (sec)	% <	% >	Mode Lie	
			Value	% of All Lies
1-120	0%	22%	3600	≥ 52%
1000	3%	19%	3600	53%
3600	3%	7%	86400	69%
10000	16%	7%	3600	53%
10800	16%	7%	3600	52%
86400	16%	0%	3600	72%
100000	40%	0%	86400	59%
604800	40%	0%	86400	59%
1000000	88%	0%	604800	54%

Table 8: RDNS$_{di}$ TTL Deviations

Behavior	Percentage of Measurements
Honest	36%
Lie on Initial	55%
Lie on Subsequent	5%
Constant TTL	5%
Incrementing TTL	0%

Table 9: RDNS$_i$ TTL Behavior

initial response in §7.1 and 88% of FDNSes are honest on the initial response in §7.2. Second, we can utilize more than two FDNSes in coordinated probing to mitigate the effect of FDNS lies. Instead of F_1 and F_2 representing single FDNSes in the above description of the experiment, we utilize up to 10 FDNSes that we divide into two sets. We send the same request through each FDNS at roughly the same time.

If any FDNS responds with the correct TTL value, then we conclude the RDNS$_i$ must be truthful since every component was truthful in this case. On the other hand, if no FDNS responds with the correct TTL, then it is probable that the RDNS$_i$ is responsible for the lie. In this situation, there are three scenarios. First, some of the FDNSes are in the set of FDNSes that were honest in initial responses *and* the TTL values from these FDNSes all agree. In this ideal case, their responses collectively identify the actual TTL the RDNS$_i$ provides. In the second scenario, the TTL values arriving at honest FDNSes do not agree. One potential cause of this case is HDNSes interposing between some of the FDNSes and the RDNS$_i$. In this case, we assume that the RDNS$_i$ returns the most common TTL value among the honest FDNSes. In the final scenario, none of the FDNSes are honest. In this case, we assume that the RDNS$_i$ returns the most common TTL value among all the FDNSes.

If the majority of FDNSes access an RDNS$_i$ through the same HDNS, then our experiment will conflate the behavior of the RDNS$_i$ and the HDNS. However, we note that if an RDNS$_i$ is *only* accessible through a single HDNS, then learning the RDNS$_i$'s behavior in isolation is moot because only the aggregate behavior of both components will ever impact client devices in the real system.

We validate our technique for determining RDNS$_i$ TTL behavior using RDNS$_{di}$es, which allow us to obtain ground truth by direct probing. The results using our coordinated probing technique agree with the ground truth in 98% of the cases, and not only in detecting whether an RDNS$_i$ is honest but also in determining the quantitative TTL violations (as we will show, most lies are from a small fixed set of TTLs).

In our dataset, there are 46K RDNS$_i$es and we conduct in-depth probing for 22K of them (due to logistical issues, as sketched above). Table 9 shows our findings. We determine that 36% of RDNS$_i$es are honest. Further, 55% of RDNS$_i$ lie on the initial response to F_1, but only 5% of RDNS$_i$es lie in response to subsequent requests from F_2 indicating that the behavior of caching a different TTL than initially returned is less prevalent in RDNS$_i$es than FDNSes. This supports our conjecture that FDNSes, being mostly home-based devices, represent more primitive implementations of DNS. In addition to incorrect TTLs, we find 8% of RDNS$_i$es return constant TTL values without decrementing. The TTL deviations from RDNS$_i$es, merged with the results for RDNS$_{di}$es are shown in Table 10.

We now consider how long a record remains cached and accessible at RDNS$_i$es. Again, we use the experimental setup described earlier in this section with one exception. Instead of querying from F_2 immediately after receiving the response from F_1, we wait before repeating the query (using the same intervals as in §7.1 up to

Expected (sec)	% <	% >	Mode Lie	
			Value	% of All Lies
1-120	$\leq$ 1%	$\leq$ 1%	300	$\geq$ 34%
1000	1%	0%	900	29%
3600	1%	0%	80	19%
10000	2%	0%	3600	35%
10800	2%	0%	7200	20%
86400	5%	0%	21600	32%
100000	11%	0%	86400	55%
604800	11%	0%	86400	53%
1000000	49%	0%	604800	71%

Table 10: RDNS$_i$ TTL Deviations

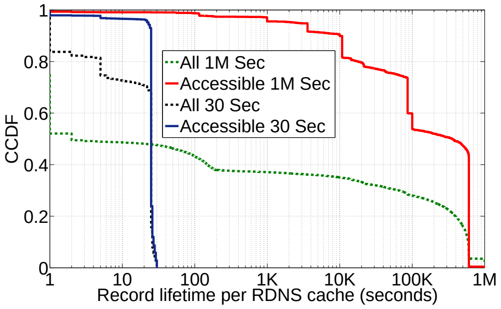

Figure 14: Distribution of record availability in RDNS$_i$ caches for records with TTLs of 30 and 1 million seconds.

the TTL value[6]). For each RDNS$_i$, we track the latest time at which the record is available.

Again, when FDNSes become unavailable in the course of the experiment we cannot detect how much longer a record stays in the RDNS's cache. Thus, Figure 14 shows the duration of record retention separately for all measured RDNSes (the "All" lines, representing 22K tested RDNS$_i$es and underestimating the result) and for those RDNSes that remained accessible throughout the experiment (the "Accessible" lines, representing 8.8K and 2.4K RDNS$_i$es for the 30 second TTL and the 1 million second TTL, respectively).

Concentrating on the "Accessible" lines as more reliable, we show that DNS records with a TTL of 1 million seconds stay in the cache for a long time, with the records still present 10K seconds (2.8 hours) after being inserted in 90% of the cases. Furthermore, step-wise drops indicate record evictions at fixed values of time in cache, indicating some configuration parameters rather than capacity eviction, and a more gradually descending line in the aggregate behavior (Figure 12) is likely due to FDNS affects. The record with a TTL of 30 seconds remains in the cache for the full TTL in 94% of the tested RDNS$_i$es.

8. DATASET REPRESENTATIVENESS

Finally, we return to the issue of bias in our datasets first mentioned in §4. Since our scans do not encompass the entire Internet, it is possible that our results are not demonstrative of the entire population of FDNSes and RDNS$_i$es due to biases in our scanning methodology. In particular, our results on FDNS behavior encompass a subset of FDNSes, i.e., those which allow cache injection. Similarly, our results for RDNS$_i$es include only those RDNS$_i$es

[6] Unfortunately, we neglected to measure intervals beyond TTL and hence do not check if RDNSes cache records beyond the authoritative TTL as we did for FDNSes.

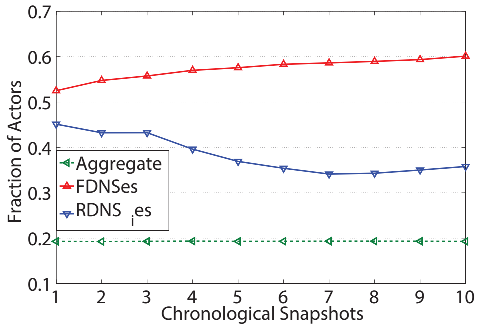

Figure 15: Fraction of honest actors over the discovery process.

for which we discover at least two FDNSes. For these two, we demonstrate here that our dataset is representative of the respective subsets of the whole population. On the other hand, we can validate the aggregate behavior against the whole population.

We assess representativeness of our datasets by calculating the fraction of actors which honestly report the TTL value for all TTL values we utilize. We divide our datasets into ten slices ordered by the time of discovery. For the aggregate behavior and FDNS behavior, the 10 slices each include an identical number of measured FDNSes while for the RDNS$_i$ behavior, the 10 slices each include an identical number of measured RDNS$_i$es. We then calculate a cumulative snapshot of the fraction of honest actors found in the first n slices for 1–10 slices. The cumulative rate should flatten out if the dataset is typical of the broader population.

Figure 15 shows the results. The fraction of honest actors in the aggregate data remains constant throughout the 10 snapshots, indicating that we quickly discover a representative sample in this study. In particular, these results indicate that the "Scan on First Hit" method of discovery used in this study and which has a bias potential, does not bias this particular metric. The fraction of honest RDNS$_i$es decreases over time but appears to converge to a constant value by the 7^{th} snapshot. This shows that (1) honest RDNS$_i$es are discovered at a higher rate towards the beginning of the scan and (2) we discover a sufficient number of RDNS$_i$es to be representative of the general population. Finally, the fraction of honest FDNSes increases throughout the 10 snapshots though the growth is flattening. This result indicates that our dataset is not sufficient to capture a representative set of FDNSes that allow cache injection. The fraction of honest FDNSes in the true population is likely higher than what we report in this paper.

A larger question of representativeness is whether open DNS resolvers are representative of the overall population of DNS resolvers users employ. In other words, do the behaviors we find in ODNSes more broadly apply to first hop resolvers in general? Our methodology does not afford any way to directly assess this question given that we would have to do so from inside many edge networks to gain an understanding of resolver behavior for first hop resolvers that are not arbitrarily accessible. This problem does not apply to our RDNS results because given the window that the myriad ODNSes provide we are readily able to get "inside" the RDNSes' networks and assess their behavior.

9. CONCLUSION

In this paper, we present a set of methodologies for efficiently discovering the client-side DNS infrastructure and, once discovered, teasing apart the behavior of the actors within the system including components that cannot be directly probed. Using these methodologies, we assess various aspects of the client-side DNS infrastructure and its behavior with respect to caching, both in aggregate and separately for different actors. In terms of the infrastructure, we double previous estimates of the number of open resolvers on the Internet, find evidence of wide use of shared resolver pools, and observe significant distances DNS messages travel within the infrastructure. In terms of caching behavior, we show how long various actors retain records and how they report TTL values. In general, we observe that the authoritative TTL value is frequently modified by the client-side DNS infrastructure. We show that large TTLs are reduced in 64% of the cases, and small TTLs are increased in 11% of our measurements. We tease apart these behaviors and attribute the former behavior predominantly to RDNS$_i$es and the latter behavior predominantly to FDNSes. Additionally, we find that cache evictions due to capacity limits occur infrequently in RDNS$_i$es even for rarely accessed records. At the same time, while the TTL is frequently mis-reported to clients, resolvers themselves do not retain records much past authoritative TTL. We observe that records are returned past TTL in only 10% of the cases, even for records with a relatively short TTL of 30 seconds.

Acknowledgments

This work was supported in part by NSF through grants CNS-0831821, CNS-1213157 and CNS-0831535. The authors would like to thank the anonymous reviewers – and in particular our shepherd, Meeyoung Cha – for their assistance in improving the paper.

10. REFERENCES

[1] Open Resolver Project. `http://openresolverproject.org/`.

[2] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig. Comparing DNS Resolvers in the Wild. In *10th ACM SIGCOMM IMC*, pages 15–21, 2010.

[3] Alexa. `http://www.alexa.com/topsites`.

[4] H. A. Alzoubi, M. Rabinovich, and O. Spatscheck. The Anatomy of LDNS Clusters: Findings and Implications for Web Content Delivery. In *22d Int. WWW Conf.*, 2013.

[5] R. Arends. DNS Security Introduction and Requirements, 2005. RFC 4033.

[6] T. Callahan, M. Allman, and M. Rabinovich. On Modern DNS Behavior and Properties. *ACM SIGCOMM CCR*, 43(3):7–15, 2013.

[7] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM CCR*, 33(3):3–12, 2003.

[8] D. Dagon, N. Provos, C. Lee, and W. Lee. Corrupted DNS Resolution Paths: The Rise of a Malicious Resolution Authority. In *NDSS*, 2008.

[9] I. Google. `https://developers.google.com/speed/public-dns/docs/performance#loadbalance`.

[10] K. Gummadi, S. Saroiu, and S. Gribble. King: Estimating Latency Between Arbitrary Internet End Hosts. In *2nd ACM SIGCOMM Workshop on Internet Measurment*, pages 5–18. ACM, 2002.

[11] C. Huang, D. Maltz, J. Li, and A. Greenberg. Public DNS System and Global Traffic Management. In *IEEE INFOCOM*, pages 2615 –2623, 2011.

[12] D. Kaminsky. Black Ops 2008: It's the End of the Cache As We Know It. *Black Hat USA*, 2008.

[13] D. Leonard and D. Loguinov. Demystifying Service Discovery: Implementing an Internet-wide Scanner. In *10th ACM IMC*, pages 109–122, 2010.

[14] R. Liston, S. Srinivasan, and E. Zegura. Diversity in DNS Performance Measures. In *2nd ACM SIGCOMM Workshop on Internet Measurment*, pages 19–31. ACM, 2002.

[15] Z. M. Mao, C. D. Cranor, F. Douglis, M. Rabinovich, O. Spatscheck, and J. Wang. A Precise and Efficient Evaluation of the Proximity Between Web Clients and Their Local DNS Servers. In *USENIX ATC*, pages 229–242, 2002.

[16] Geoip. maxmind llc, 2012.

[17] J. Pang, A. Akella, A. Shaikh, B. Krishnamurthy, and S. Seshan. On the Responsiveness of DNS-based Network Control. In *4th ACM SIGCOMM IMC*, pages 21–26, 2004.

[18] M. Rajab, F. Monrose, A. Terzis, and N. Provos. Peeking Through the Cloud: DNS-based Estimation and its Applications. In *Applied Cryptography and Network Security*, pages 21–38. Springer, 2008.

[19] K. Schomp, T. Callahan, M. Rabinovich, and M. Allman. Client-Side DNS Infrastructure Dataset, Oct. 2013. `http://dns-scans.eecs.cwru.edu/`.

[20] A. Shaikh, R. Tewari, and M. Agrawal. On the Effectiveness of DNS-based Server Selection. In *INFOCOM*, pages 1801–1810, 2001.

[21] C. Shue, A. Kalafut, M. Allman, and C. Taylor. On Building Inexpensive Network Capabilities. *ACM SIGCOMM CCR*, 42(2), Apr. 2012.

[22] G. Sisson. DNS Survey: October 2010. `http://dns.measurement-factory.com/surveys/201010/`, 2010.

[23] C. E. Wills, M. Mikhailov, and H. Shang. Inferring Relative Popularity of Internet Applications by Actively Querying DNS Caches. In *3rd ACM SIGCOMM IMC*, pages 78–90, 2003.