# COMPLEXITY AND SECURITY OF THE DOMAIN NAME SYSTEM

by

KYLE GRAHAM SCHOMP

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

Department of Electrical Engineering and Computer Science

CASE WESTERN RESERVE UNIVERSITY

May 2016

**CASE WESTERN RESERVE UNIVERSITY**

**SCHOOL OF GRADUATE STUDIES**

We hereby approve the thesis of

**KYLE GRAHAM SCHOMP**

candidate for the **DOCTOR OF PHILOSOPHY** degree*

Committee Chair: Michael Rabinovich

Committee Member: Mark Allman

Committee Member: Vincenzo Liberatore

Committee Member: Frank Merat

Committee Member: Soumya Ray

Date: 11/20/2015

*We also certify that written approval has been obtained for any propriety material contained therein.

# Contents

# List of Tables

# List of Figures

# List of Acronyms

**ADNS** Authoritative DNS Server

**CDN** Content Delivery Network

**DHCP** Dynamic Host Configuration Protocol

**DNS** Domain Name System

**DNSSEC** DNS Security Extensions

**FDNS** Forwarding DNS Server

**HDNS** Hidden DNS Server

**HTML** Hypertext Markup Language

**HTTP** Hypertext Transfer Protocol

**ICMP** Internet Control Message Protocol

**ISP** Internet Service Provider

**NAT** Network Address Translation

**NTP** Network Time Protocol

**ODNS** Open DNS Server

**RDNS** Recursive DNS Server

**RDNS**$_d$ Recursive Direct DNS Server

**RDNS**$_{di}$ Recursive Direct and Indirect DNS Server

**RDNS**$_i$ Recursive Indirect DNS Server

**RTT** Round-Trip Time

**SLD** Second-Level Domain

**TCP** Transmission Control Protocol

**TLD** Top-Level Domain

**TTL** DNS Record Time-To-Live

**UDP** User Datagram Protocol

**WPAD** Web Proxy Autodiscovery Protocol

# Acknowledgments

Thank you to everyone who helped me make it this far. In particular, thank you to Michael Rabinovich and Mark Allman for their guidance and advice. Thank you to my family for supporting me for many, many years. Thank you to my dear wife, Fangfei, for encouraging me to keep working.

KYLE GRAHAM SCHOMP

*Case Western Reserve University*

*May 2016*

# Complexity and Security of the Domain Name System

Abstract

by

KYLE GRAHAM SCHOMP

The Domain Name System (DNS) provides mapping of meaningful names to arbitrary data for applications and services on the Internet. Since its original design, the system has grown in complexity and our understanding of the system has lagged behind. In this dissertation, we perform measurement studies of the DNS infrastructure demonstrating the complexity of the system and showing that different parts of the infrastructure exhibit varying behaviors, some being violations of the DNS specification. The DNS also has known weaknesses to attack and we reinforce this by uncovering a new vulnerability against one component of the system. As a result, understanding and maintaining the DNS is increasingly hard. In response to these issues, we propose a modification to the DNS that simplifies the resolution path and reduces the attack surface. We observe that the potential costs of this modification can be managed and discuss ways that the cost may be mitigated.

# Chapter 1

# Introduction

The Domain Name System (DNS) [Moc87a, Moc87b] provides mapping of meaningful names to arbitrary data for applications and services on the Internet. The most common use of DNS is the resolution of hostnames to IP addresses which in turn are used to route packets across the network. The DNS was originally specified in 1983 [Moc83a, Moc83b] and, since its original design, the system has grown in complexity. As we will demonstrate, the DNS ecosystem now consists of multiple layers of servers and, at some layers, multiple servers cooperating to produce many resolution paths through the ecosystem. The policies controlling DNS servers may also differ by operator. Further, security vulnerabilities in the DNS allow "record injection" attacks, which can subvert replace the correct name to data mappings with potentially malicious mappings and consequently breach the security of applications that rely upon the DNS. Some actors within the DNS ecosystem have well known vulnerabilities and the security community continues to expose new weaknesses, leaving users vulnerable to attacks ranging from snooping on sensitive information to phishing.

In this dissertation, we show that the DNS ecosystem is complex and parts of the ecosystem are hidden from external observers. Different parts of the infrastructure exhibit varying behaviors, including violations of the DNS specification and security vulnerabilities. As a result, reasoning about and maintaining the system is difficult. Thus, we propose

a modification to the DNS that simplifies the resolution path and reduces the attack surface of the DNS.

**Topology of the DNS Infrastructure:** The complexity of the DNS infrastructure has increased dramatically to accommodate new performance and scale demands that have been placed upon the DNS as the Internet has expanded. The increase in complexity is especially true of the servers in the DNS infrastructure that are responsible for resolving names on behalf of clients. The original structure of the DNS system had clients issue queries to a local DNS resolver which in turn performs the task of resolution, then returns the answer to the client. Now, facilitated by the simplicity of the stateless and connectionless DNS protocol, the DNS has developed into a complex ecosystem that we demonstrate often involves several layers of shared resolvers. The shared resolvers, in turn, may coordinate with other resolvers producing intricate resolution paths. Resolvers operate based upon their own, potentially different, policies and some components are hidden from external observers. This complexity makes it difficult to understand the behavior of the resolving infrastructure and to attribute responsibility for distinct behaviors to the individual actors if or when those behaviors cause problems.

Our first step in this dissertation is to examine the resolver infrastructure. To study the wide range of deployed structures, we perform scans of the Internet with the intent of discovering large samples of DNS resolvers in Chapter 3. The resolvers that we discover give us entry points into the DNS ecosystem from many vantage points that are globally distributed. Using this data, we map the complex DNS resolver infrastructure, identifying actors serving various roles within it.

Second, we develop techniques to efficiently discover samples of the infrastructure for further measurement. As the DNS continues to evolve, techniques for efficient discovery and rediscovery of the DNS resolver infrastructure allow us to update and refine our understanding quickly. To improve our discovery techniques, we leverage observations

we make about the density of actors within IP address space and querying techniques to maximize the number of actors discovered per probe sent by our measurement apparatus, reducing the cost of discovering a sample of the infrastructure. We reuse these techniques in subsequent measurements because many of the individual actors in the DNS ecosystem are short-lived and therefore must be discovered again for each new experiment.

**Measuring Behavior in the DNS Infrastructure:** From discovering the resolver infrastructure, we turn to measurements of its behavior. In Chapter 4, we develop measurement techniques for isolating the behavior of individual actors within the system, some of whom cannot be accessed directly. Then, using the efficient discovery techniques detailed in Chapter 3, we apply our methodologies to assess the caching behavior of the various actors. Caching in the DNS has a large impact on performance—by reducing the latency of the iterative DNS resolution process—and scalability—by reducing the load in queries that components of the system must handle. Further, the duration that DNS answers remain within caches determines how long systems must maintain stale mappings to avoid service disruptions. We measure caching behavior both in aggregate and separately for the different actors.

The first component of our assessment of caching behavior is how various actors treat the time-to-live (TTL) setting, which controls how long resolvers should store answers in their cache. Despite being a simple notion, we find that different actors handle the TTL differently and exhibit a variety of behaviors. Previous studies demonstrate that in many cases the TTL is distorted before reaching the original requesting client and we confirm this observation. Next, we expand upon previous studies by demonstrating which actors cause modification and how they modify the TTL setting, thus misleading other actors into using an incorrect TTL.

We also assess the time an unused answer remains in the cache of the various actors within the resolving infrastructure. Two possible reasons for an eviction of an answer are

the TTL expiring or cache capacity is reached and space must be freed. We find scant evidence of a general capacity limitation problem in resolvers' caches.

**Characterization of DNS Client Behavior:** Individual clients on the Internet issue DNS queries for a variety of different purposes leading to a complex DNS querying behavior. Yet, our understanding of DNS query streams is largely based on aggregate populations of clients leaving our knowledge of individual client behavior limited. Focus on the aggregate traffic obstructs our knowledge of how DNS traffic is distributed among clients and how DNS traffic varies with population size. Further, the traffic patterns in aggregate populations obscure the processes generating DNS traffic. Knowledge of how individual clients use the DNS may inform better choices in ($i$) dimensioning of the DNS resolver infrastructure, ($ii$) protecting the system against malicious individuals via anomaly detection, and ($iii$) modifying the current DNS system or designing future naming systems.

In Chapter 5, we present what is to the best of our knowledge the first characterization of DNS client behavior. It paves the way for eventual construction of models and workload generators. We monitor DNS transactions between a population of thousands of clients and their local resolver such that we are able to directly tie lookups to individual clients. The study presented here uncovers a variety of behaviors and characteristics that significantly increase our comprehension of how DNS operates. Our ultimate goal for this work is an analytical model of DNS client behavior and this characterization moves us towards that analytical model.

Our measurements provide insight into how the DNS is utilized by real clients today. Broadly, we study two topics: the number and spacing of queries in time and which names individual clients lookup. We identify a wide variety of devices producing DNS traffic with differing behaviors and find that general purpose user-facing devices can be detected by markers of Web browsing behavior.

**A New Security Vulnerability in the DNS:** Because many applications depend upon DNS, the security of DNS has a large impact on the security of the overall network. Replacing an authoritative mapping from hostname to IP address with a fraudulent mapping will divert users to malicious hosts. Once diverted, users may be subject to a variety of follow-up attacks from phishing to malware installation. In Chapter 6, we uncover a new vulnerability to an attack that enables replacing authoritative mappings with malicious ones. Attacks of this type are known as "record injection" attacks. Using the discovery techniques introduced in Chapter 3, we issue specifically tailored queries to the uncovered resolver infrastructure and observe evidence of vulnerability to the record injection attack.

Given that record injection attacks may impact users, we built a Web-based tool that helps users learn about how their DNS resolutions are being handled directly from their Web browser [Sca]. The tool alerts users to any specific vulnerabilities uncovered in the resolver infrastructure they use and suggests alternatives. We also report on the adoption of a variety of mitigation strategies and the timing performance of DNS lookups.

**DNS Shared Resolvers Considered Harmful:** In this dissertation, we demonstrate the complexity of the DNS client-side infrastructure and show that it is constituted of many distinct components, potentially making the system difficult to manage and troubleshoot. Additionally, resolvers have previously been shown to be vulnerable to multiple forms of attack and we detect a new vulnerability in a specific component of the infrastructure. In Chapter 7, we analyze and motivate a change to the resolution path by removing the DNS resolver infrastructure entirely and pushing the functionality of iterative resolution back upon the clients. The clients themselves become more complex, but in a way that is easier to manage and more transparent than our current nebulous situation where a resolution proceeds along a path of hidden actors with diverse behaviors. Direct client resolution also ($i$) eliminates the threat of record injection attacks against shared resolvers for clients conducting their own resolutions and ($ii$) reduces the overall attack surface of the DNS ecosystem

since fewer components are needed in client resolution than traditional resolution.

There are two primary costs that arise from direct client resolution: increased delay in responses for clients and increased load in terms of queries for DNS servers. In simulations of the new resolution path, we show that the cost in terms of delay is low and the cost in terms of load may be manageable. We suggest potential mitigation techniques. Further, we note that direct client resolution is individually deployable as technical changes are only necessary on the client devices. However, for large scale transition to direct client resolution, we lay out several barriers to deployment and suggest future directions to make a large scale transition to client resolution practical.

# Chapter 2

# DNS Background

This chapter provides a minimum background on DNS needed to understand this dissertation. The concepts discussed here will be used repeatedly in the subsequent chapters. DNS is a query and response system using a simple datagram protocol to collect answers from a distributed database. Clients ask questions by sending queries for specifically structured names from within a hierarchical namespace. The responses to queries contain answers known as *resource records*. The process of obtaining an answer to a query is called *resolution* and there are three pieces of the system that participate. First, *authoritative DNS servers* maintain resource records and are queried by other components to obtain the authoritative resource records that match the given query. Second, clients use *stub DNS resolvers* to issue the query. Third, the stub resolvers commonly issue the query to *recursive DNS resolvers* that perform resolution by iteratively querying authoritative DNS servers. In the following sections, we describe each component in turn.

**Authoritative DNS Servers:** Authoritative DNS (ADNS) servers maintain the resource records for names within sections of the namespace. Partitioning the namespace into *zones* splits responsibility for maintaining the resource records across many ADNS servers and enables DNS to scale as the number of records grows. The zones are organized into a tree and often operated by different organizations. Names used in queries indicate their

position within the tree via labels separated by periods. For illustration, we use the name "www.google.com." throughout this chapter. The root is represented by an empty label on the right of the name and each label moving to the left is a step further down the tree.

At the root of the tree, the root ADNS servers only hold records for names that are within the root zone. This includes records that provide information on how to reach ADNS servers for zones below the root, e.g., "com.". Similarly, the ADNS servers for "com." also hold records with the information needed to reach ADNS servers for zones below "com.", e.g., "google.com.". This process of zone *delegation* may continue at the operator's preference. The ADNS servers for "google.com." contain records for names within the zone, e.g., "www.google.com.".

Among the contents of each resource record are the following important fields:

**NAME** is the name of the record, e.g., "www.google.com.".

**TYPE** indicates what information the record holds, e.g., "A" means that the record holds an IPv4 address. While IP addresses are the most commonly queried record types, many other types are also supported.

**TTL** (time-to-live) is the duration that a record remains valid in seconds. The contents of the record may continue to be used by whoever obtained the record until the TTL elapses. Use of the contents after the TTL expires is a violation.

**RDATA** holds the data to which the name maps. The structure of RDATA is determined by the TYPE, e.g., with TYPE "A", RDATA is an IPv4 address.

The ADNS servers accept queries from external agents and return responses containing associated resource records if any exist using the DNS protocol. Both the query and response are typically a single UDP datagram. For cases where the payload will not fit within a single datagram, TCP is used as a backup transport method. The port number—in both UDP and TCP—that ADNS servers listen upon for queries is 53.

**Stub DNS Resolvers:** Clients resolve names via the use of a stub resolver that is often a component of the operating system. The stub resolver then issues queries to a recursive DNS resolver and stub resolvers are commonly configured to use the IP addresses of one or more recursive DNS resolvers through DHCP, but the sufficiently technical users may manually configure another choice. In particular, Google Public DNS [Goob] and OpenDNS [Opeb] both offer recursive DNS resolvers for public use. Stub resolvers interact with the recursive DNS resolvers using the same DNS protocol used by the ADNS servers.

The stub resolver may *cache* the resource records returned in responses according to the TTL of the record. The stub resolver will use the same record to answer multiple queries for the same name from the client. Once the TTL has elapsed since receiving the record, the stub resolver discards the record.

**Recursive DNS Resolvers:** Recursive DNS (RDNS) resolvers are charged with resolving names for clients. After receiving the query from the stub resolver, the RDNS servers perform all of the work necessary to obtain an answer to the query and then return the answer to the stub resolver. To this end, the RDNS servers navigate the DNS tree by issuing queries to authoritative DNS servers and following delegations to other authoritative DNS servers. To anchor the resolution process, the RDNS servers always hold well-known IP addresses of the root servers. Like ADNS servers, RDNS servers listen on port 53 for queries. After receiving a query for a name, e.g., "www.google.com.", the RDNS server follows the delegations provided by ADNS servers starting at the root until the RDNS server arrives at an ADNS server that maintains the resource record for "www.google.com.". Then, the RDNS server returns the records to whomever issued the query.

As with stub resolvers, RDNS servers may cache records locally—constrained by TTL—and use the same record to answer multiple queries without issuing another query to the ADNS servers. The delegation records between ADNS servers may also be cached and reused. When an RDNS server returns a record to a stub resolver from cache, the

TTL value of the record is decreased to account for the time the record spent in the RDNS server's cache.

This concludes the minimum background on the DNS. For a detailed specification, the reader is encourage to consult the DNS specification [Moc83a, Moc83b, Moc87a, Moc87b].

# Chapter 3

# Topology of the DNS Infrastructure

In this chapter, we present the results of wide scale measurements of the client-side DNS infrastructure[1]. With the crucial role DNS plays, the complexity of the DNS infrastructure—especially the client-side query-resolving aspect—has increased dramatically. In § 2, we indicate a client using a stub resolver communicates with a single recursive DNS resolver, which in turn queries authoritative nameservers on the clients' behalf. However, this is a simplification of what often actually occurs. DNS has become a complex ecosystem often involving multiple layers of shared resolvers, which can, in turn, coordinate with additional resolvers. This chapter targets the challenge of improving our understanding the client-side DNS infrastructure as it is presently deployed. We make the following key contributions:

- 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 [DPLL08, LL10], probing strategies that improve the rate of discovery can facilitate subsequent measurements. The methodologies developed in this chapter will be used in subsequent chapters to aid in our experiments.

- We double, from 15 to 32 million, previous estimates of the how many open resolvers there are on the Internet.

[1] This work originally appeared in [SCRA13].

- 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 in terms of network delay. We find 20% of open resolvers experiencing at least 100 msec of delay before their queries leave the resolution infrastructure.

## 3.1 Related Work

Our over-arching methodology for studying 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 [DPLL08], 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. Our work extends the probing methodology presented in [DPLL08] and again used in [Cal12] to effectively discover resolver pools and examines resolver characteristics not discussed in those papers.

Efficiently scanning the IPv4 address space for service discovery (including DNS) while avoiding complaints is discussed in [LL10]. While [LL10] explores reducing the burden of probing on the targets of probes, 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, and ($ii$) identify pools of recursive resolvers.

We also consider this work related to [AMSU10], which performs DNS lookups from several vantage points in order to compare performance among various local resolvers and public DNS services. Ager, et al. [AMSU10] find that ISP-provided resolvers often outperform public DNS services in query latency. Huang, et al. [HMLG11] confirms this

result. Another performance-centric study is [LSZ02], which shows that DNS performance can vary widely between resolver infrastructure used. In our work, we offer an explanation for the variation in performance by demonstrating large latencies that queries must sometimes suffer within the resolving infrastructure.

Several studies [WMS03, RMTP08, GSG02] 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. Wills, et al. [WMS03] study how long DNS records for Websites remain in cache to infer how popular the Websites are. Rajab, et al. [RMTP08] extend this technique by using it to estimate the popularity of several services that rely on DNS and not just Web site popularity. Also, Su, et al. [SCKB06] propose a method of using redirections for the Akamai content delivery network to find quality Internet paths without performing path probing and monitoring themselves.

Several prior studies consider the number of open resolvers on the Internet [LL10, Sis10]. Leonard and Loguinov [LL10] perform a full Internet scan and report the number of open resolvers responding in 2010. Sisson [Sis10] uses a sample of the IP address space to estimate the number of open resolvers. In our work, we again estimate how many open resolvers there are on the Internet to see how it has progressed with time.

The distance between clients and their resolvers is studied in [STA01, MCD+02, ARS13, Cal12]. Shaikh, et al. [STA01] find that clients and their resolvers are often topologically quite far apart. Mao, et al. [MCD+02] find that clients and their resolvers are often within the same autonomous system. Alzoubi, et al. [ARS13] show that the physical distance between clients and their resolvers can be surprisingly large. Callahan [Cal12] demonstrates large distances between actors in the resolver infrastructure. We confirm the findings of [Cal12] and expand upon them by devising a method to measure latency between actors in the resolver infrastructure.

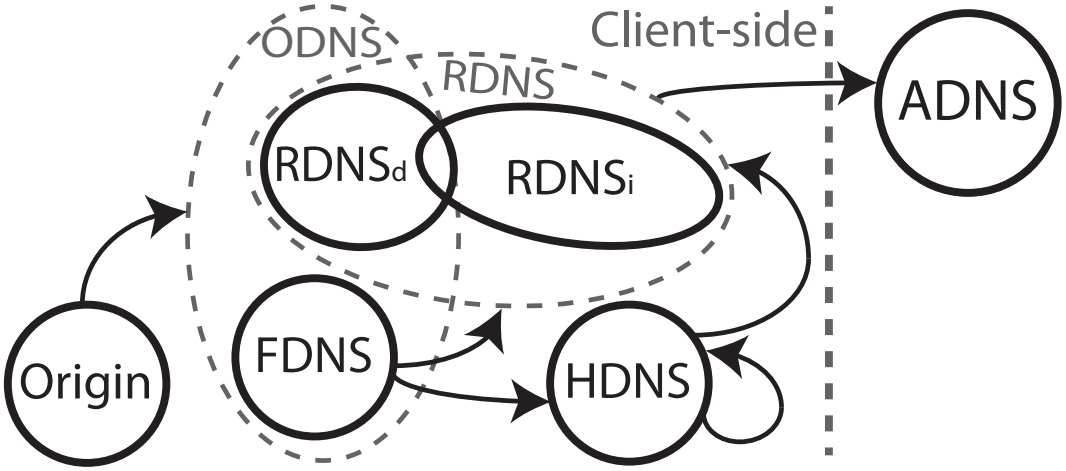


Figure 3.1: Structure of the client-side DNS infrastructure.

## 3.2 Client-side DNS Infrastructure

In this section, we provide descriptions of the various actors within the client-side DNS infrastructure that, as a whole, resolve names on behalf of clients. These descriptions have previously appeared in [SCRA13, Cal12]. The components of the client-side DNS infrastructure are depicted in Figure 3.1 and we define our terminology below.

**Origin** devices are end-user devices (i.e., stub resolvers). Origin devices also represent the measure points that we use to send DNS requests for discovering componets of the infrastructure.

**ODNS** ("open DNS") servers accept and process DNS requests from arbitrary sources on the Internet. Best practices recommend that DNS resolvers be limited to "internal" use.

**RDNS** ("recursive DNS resolvers") servers communicate directly with ADNS servers.

**FDNS** ("forwarding ODNS") servers are ODNS servers that do not perform recursive resolution. Instead, an FDNS server forwards the request to an RDNS server. More formally, the FDNS servers are $\{x | x \in \text{ODNS and } x \notin \text{RDNS}\}$.

**RDNS**$_d$ ("direct RDNS") servers are both RDNS servers and ODNS servers, i.e., $\{x|x \in$ ODNS and $x \in$ RDNS$\}$.

**RDNS**$_i$ ("indirect RDNS") servers are RDNS servers that issue queries to ADNS servers on behalf of one or more FDNS servers.

**RDNS**$_{di}$ ("indirect and direct RDNS") servers are RDNS servers that are both RDNS$_d$ servers and RDNS$_i$ servers, i.e., RDNS$_{di}$ servers respond to DNS queries from abitrary sources and support a population of FDNS servers. We highlight this set as particularly important for validating our measurement techniques because RDNS$_{di}$ servers may be probed both directly and indirectly.

**HDNS** ("hidden DNS") are servers which we cannot observe with our measurements. HDNS servers lie between FDNS servers and RDNS$_i$ servers. Although not detectable by our measurements, these servers are known to exist [Goob] and we account for them when designing our experiments.

The path of a typical resolution in the infrastructure (Figure 3.2) follows these steps:

1. A stub resolver (origin) within the client sends the DNS request to a home router (FDNS server).

2. The home router sends the request to a DNS provider, either to an HDNS server or directly to an RDNS$_i$ server.

3. Ultimately, the request arrives at an RDNS$_i$ server, which sends the request to ADNS servers as needed to complete the resolution.

4. The answer travels through the reverse path to the origin.

Some resolution paths may coordinate with each other and, as a result, multiple requests from the same FDNS server may actually arrive at the ADNS server through different RDNS$_i$ servers. These structures are called "RDNS pools" [Cal12]. We discuss RDNS pools in more detail in § 3.5.

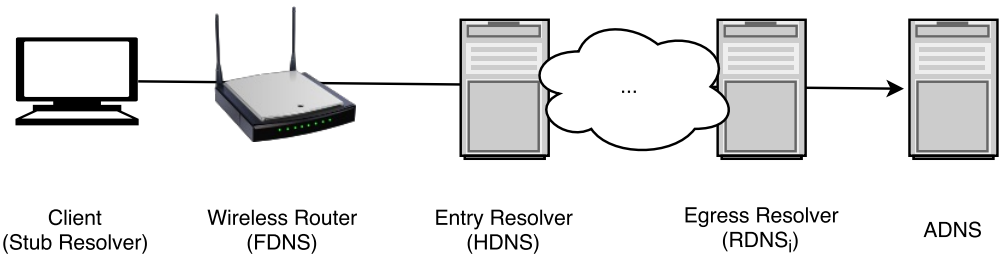


Figure 3.2: Typical path a resolution will take through the client-side DNS infrastructure.

## 3.3 Methodology Overview

In this section we sketch our general methodology and datasets. The specific methodology will vary with each experiment and is described with the details of the experiment in the subsequent sections and chapter. 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 § 4.4—after we have further developed experiments that can be brought to bear on the question.

### 3.3.1 Non-Interference With Normal Operation

While we are investigating the components of the DNS ecosystem, we use our own registered domain. Our probing rates are limited to ensure we do not interfere with normal operation of any of the components of the system that probe, and all DNS requests are for subdomains of our own domain. Thus, we do not interfere with any actual name-to-address bindings in use.

### 3.3.2 Discovering DNS Infrastructure

To examine the client-side DNS infrastructure, we must have an efficient method for finding both ODNS servers and RDNS servers. Discovering DNS infrastructure is a challenge

| Scan | Format | Start | Duration (Days) | ODNS | RDNS |
|---|---|---|---|---|---|
| $S_1$ | Random IP | 7/3/12 | 32 | 1.98M | 72.6K |
| $S_2$ | Random /24 | 8/5/12 | 17 | 841K | 43.9K |
| $S_3$ | Scan on First Hit | 10/4/12 | 25 | 17.6M | 72.1K |
| $S_4$ | Rescan of $S_2$ | 11/16/12 | 9 | 892K | 29.9K |
| $S_5$ | Scan on First Hit | 2/26/13 | 31 | 11M | 65.8K |

Table 3.1: Mapping & Measurement Datasets

because many of the components have policy restrictions preventing the acceptance of DNS requests from arbitrary hosts. We extend the measurement methodology from [DPLL08]. We register a domain name—"dnsresearch.us"—and deploy a customized ADNS server for this domain. Using approximately 100 PlanetLab [CCR+03] nodes as the origins of our measurements, we randomly probe the IP address space with DNS requests for hostnames within our domain. By embedding the probed IP address in the hostname request and observing the queries arrive at our ADNS server, we collect the IP addresses that are willing to handle our probes—thus discovering ODNS servers. The IP addresses from which the queries arrive at our ADNS server illuminate the set of RDNS servers. Finally, since the ADNS server has the addresses of both the RDNS and ODNS, we can identify $\text{RDNS}_d$ when the IP addresses are the same. When the IP addresses are different, we can associate FDNS servers with the RDNS servers they use for DNS resolution. Thus, we can elicit a response from an RDNS server that will not respond to direct probes by indirectly probing via the FDNS server.

### 3.3.3 ODNS Lifetimes

We note that during our measurements we find that ODNS servers are often short-lived—with around 40% becoming unreachable within one day (see § 3.4.1). Hence, techniques for quick rediscovery are important for our subsequent measurement studies and future study of the DNS infrastructure. We describe these techniques below and use them to collect different datasets for different experiments, as summarized in Table 3.1. Our datasets are publicly available [SCRA].

## 3.4 Methodology Details

We turn our attention to discovering components of the client-side DNS infrastructure. To facilitate our exploration of discovery methodologies, we use two datasets. The first dataset is from the $S_1$ scan in Table 3.1 and represents the probing of 255M unique random IP addresses using 267M DNS requests from 7/3/2012 to 8/3/2012. Our $S_1$ scan discovered 1.9M ODNS servers and 73K RDNS servers. The second dataset is from the $S_2$ scan in Table 3.1 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 IP address blocks—11.9M IP addresses—via 121M DNS requests. The $S_2$ dataset includes 841K ODNS servers and 44K RDNS servers. The number of probes exceeds the number of IP addresses because some ODNS servers use RDNS pools, which we attempt to discover through repeated probes to ODNS servers (see § 3.4.2 for details).

Previous work finds evidence that ODNS servers are mostly home network devices [Cal12]. These residential devices forward DNS queries from client devices to the recursive resolvers of ISPs and other major DNS providers.

### 3.4.1 ODNS Server Discovery

The fundamental aspect of discovery is finding ODNS servers 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 [LL10, Opea] to understand the prevalence of open resolvers. However, we are interested not only with discovering the existence of ODNS servers, but also with understanding their characteristics and behavior, which entails sending far more requests than discovery alone would dictate (as detailed in subsequent sections). Additionally, we find—as previously developed in the literature [DPLL08, LL10]—the window of accessibility for ODNS servers to be in general fairly short (for details, see § 3.4.1). Therefore, we must do in-depth probing in conjunction with ODNS discovery as returning

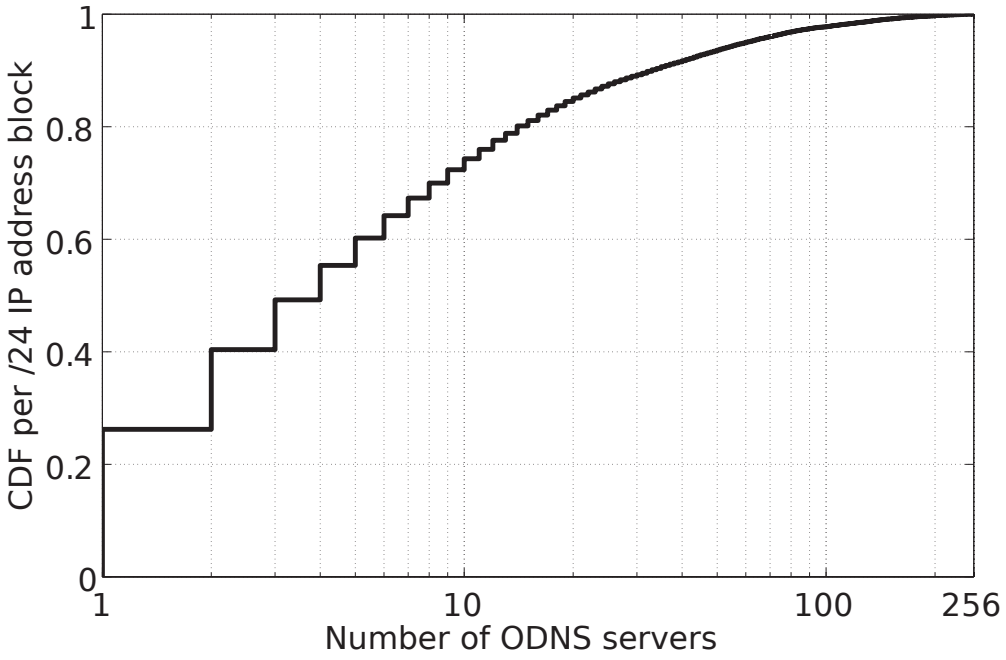


Figure 3.3: Distribution of the number of ODNS servers per /24 IP address block, excluding empty blocks.

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 server, 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 ODNS servers, and $(ii)$ whether there are probing strategies to improve the efficiency of this process.

Our first observation is that ODNS servers are unevenly distributed throughout IP space. As sketched above, in $S_2$ we choose and probe random /24 address blocks. We find that only 14% of these blocks contain ODNS servers. Further, as Figure 3.3 shows, the distribution of ODNS servers among the blocks that have ODNS servers is uneven. We find a small number of "dense" blocks with many ODNS servers. For instance, the top 10% of the address blocks each contain over 30 ODNS servers while 40% of the blocks have no

more than two ODNS servers. The average across all blocks with at least one ODNS server is approximately 13 ODNS servers per /24 block.

Discovery within such a sparse address space requires extensive scanning. One approach is a complete scan of the address space. While comprehensive, this strategy is onerous in terms of time and load on the scanning infrastructure. A plausible alternative is to collect a sample of the ODNS population and use that sample to gain general insights about the population. For collecting a sample of ODNS servers 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 server 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 $S_2$ 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 3.3. "Random IP" is then simulated by selecting randomly without replacement from the full $2^{32}$ address range and counting the rate of discovering ODNS servers. For "Scan on First Hit", we again select an address randomly without replacement. If the selected address is an ODNS server, we count not only this address, but also all addresses marked as ODNS servers in the encompassing /24 block and remove the block from the address pool for further selection.

Figure 3.4 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 ODNS servers with a complete scan. The discovery rate of "Scan on First Hit" decreases over time. The

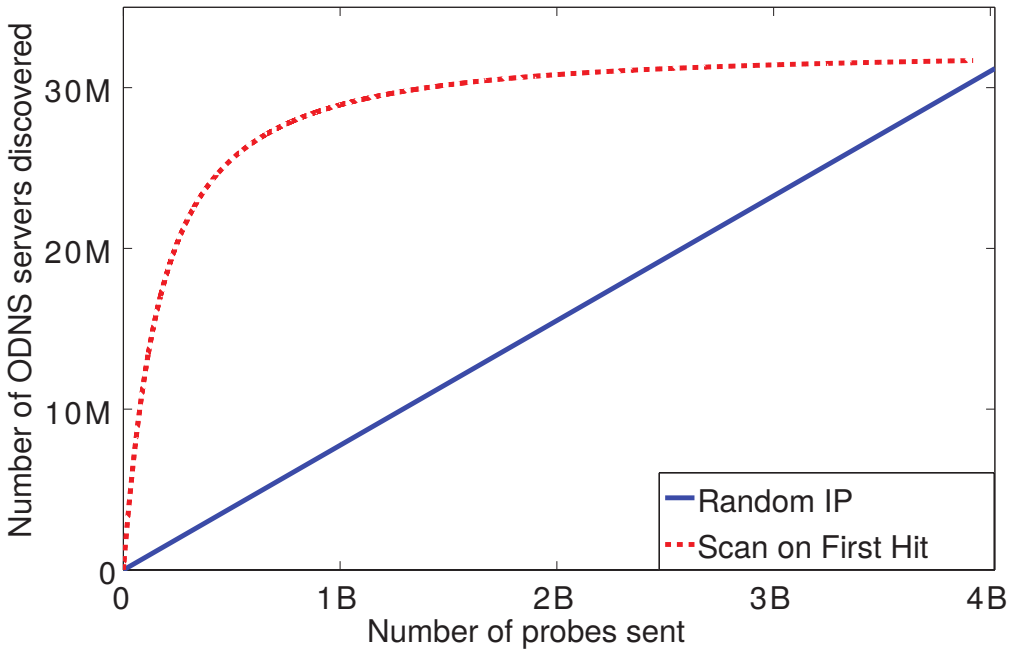


Figure 3.4: ODNS discovered versus DNS requests sent for the two discovery methods (extrapolation from $S_2$ scan to Internet scale).

reason is that the more dense a /24 address block is the higher the probability of finding an ODNS server. 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 ODNS servers. So, the random scan misses opportunities to learn about the "neighborhood" when finding an ODNS server and chances are that neighborhood is populated with additional ODNS servers.

While the "Scan on First Hit" strategy discovers more ODNS servers with fewer probes, it has a downside in that it introduces a bias in the set of discovered ODNS servers. Blocks with higher concentrations of ODNS servers have a greater chance of being discovered, thus biasing the resulting dataset towards ODNS servers 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 measurement results in § 4.4 after we detail the measurements we perform.

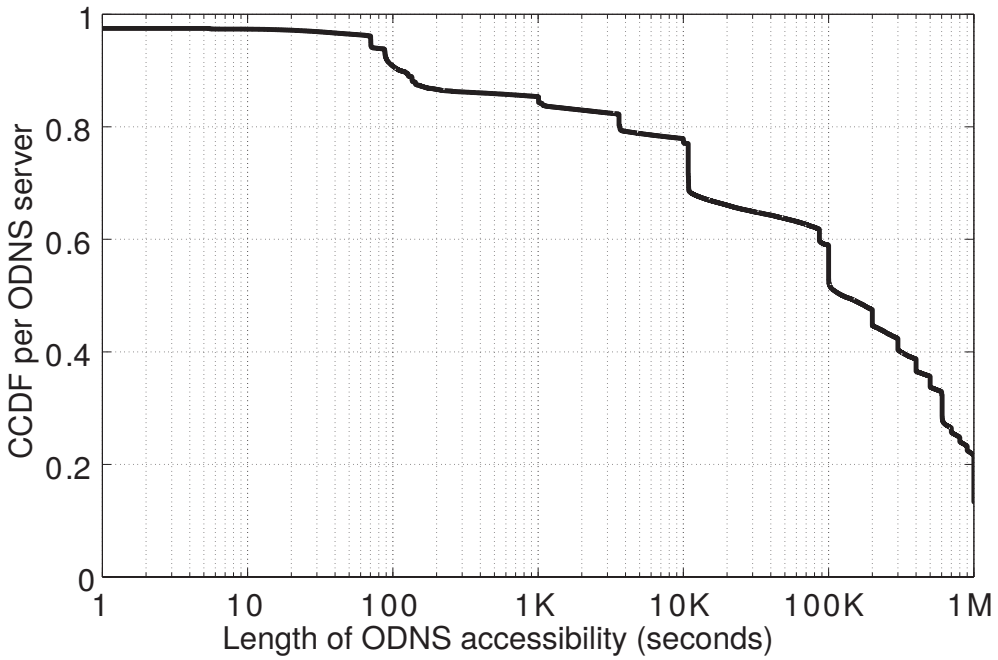


Figure 3.5: Distribution of the duration of ODNS accessibility.

**Rediscovery and Whitelisting**

ODNS servers have previously been found to be short lived [DPLL08, LL10, Cal12] and we confirm these results. In our $S_5$ scan conducted from 2/26/2013 through 3/28/2013 we repeatedly probe discovered ODNS servers for a period of 1M seconds after discovery. Details of the $S_5$ scan and the intervals at which the ODNS servers were probed are discussed in § 4. As shown in Figure 3.5 shows the period of the last probe to which the ODNS servers respond. Roughly 40% of the discovered ODNS servers did not answer any queries after one day and nearly 80% of the ODNS servers stop responding within one week. As noted above, there is evidence that the ODNS servers 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 ODNS servers become stale and biased quickly and for this reason we discover ODNS servers anew for each of our studies. Also, we note that 3% of ODNS servers never responded to probes again after the initial probe. Thus, their lifetime appears as 0 seconds in Figure 3.5.

Rescanning can be an expensive and time consuming process. Fortunately, we find that ODNS servers demonstrate a tight IP spatial cohesion: while an individual ODNS can be short lived, productive /24 blocks that contain ODNS servers tend to remain productive. We rescanned the same /24 address blocks from the $S_2$ scan between 11/16/2012 and 11/24/2012, nearly three months after the $S_2$ scan ended; this scan is labeled $S_4$ in Table 3.1. We also find that 76% of the 67K productive /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 3.4) we study re-scanning previously productive /24 blocks. Figure 3.6 shows the discovery rate for rescanning the 67K productive /24 address blocks from the $S_4$ 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 $S_1$ 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 3.6 suggests.

### 3.4.2 RDNS Server 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 ODNS servers 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 [Cal12], which may optionally utilize another layer of resolvers before the queries egress the infrastructure and are visible at our ADNS server. For example, Google's public DNS resolvers utilize a two-level topology that hashes the requested hostnames to particular

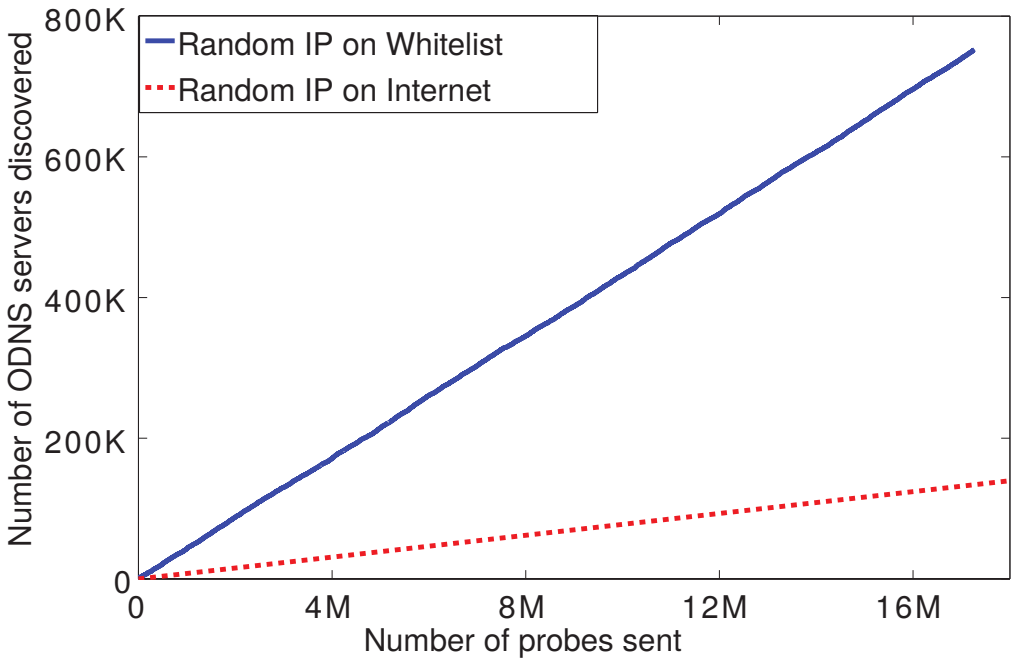


Figure 3.6: ODNS discovered using whitelisting in the $S_4$ scan compared to the discovery rate of the $S_1$ scan.

egress resolvers to improve their cache effectiveness [Goob]. Unfortunately, we can only discover the egress RDNS servers and do not have a technique for discovering HDNS servers in the middle of the infrastructure.

We use a two-pronged approach for RDNS discovery. First, for a given ODNS server 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 server rather than be answered from a cache. Second, our ADNS server returns a variable-length chain of CNAME records to queries from RDNS servers. The record type CNAME 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. Thus, by providing canonical names that are within our domain, we can elicit repeat queries from the RDNS servers. We observe that the RDNS server that sends a DNS request is often *not* the same RDNS that resolves the CNAME redirections; see an example in Figure 3.7. We

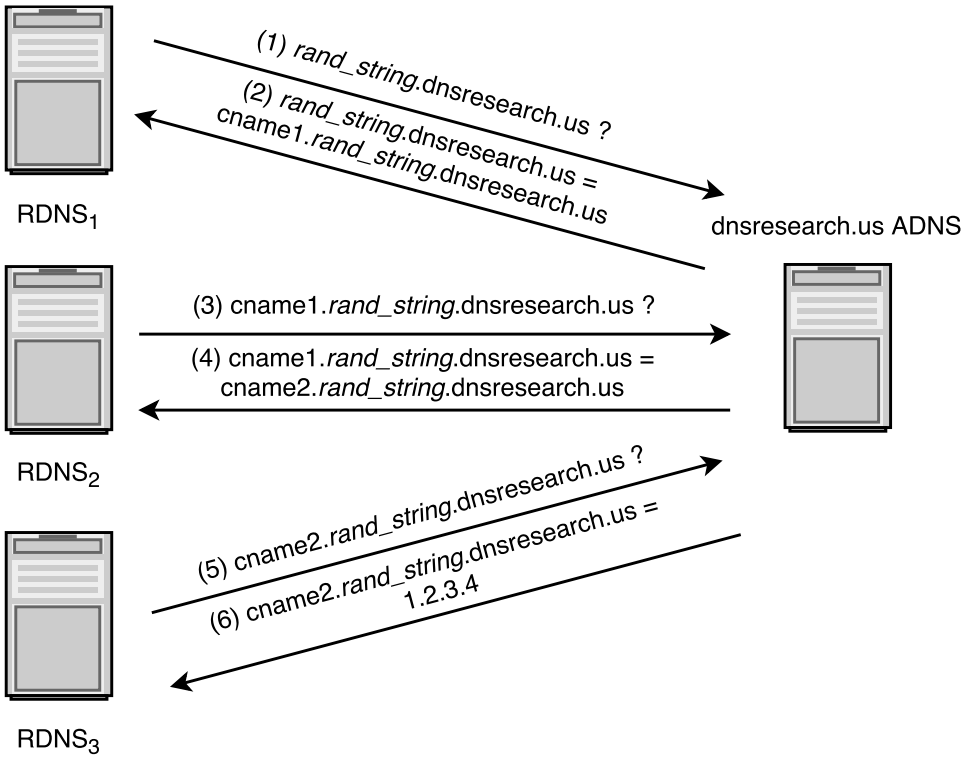


Figure 3.7: Illustration of RDNS discovery using a CNAME chain.

use both these mechanisms until we stop discovering new members of the observed RDNS pool. Specifically, our strategy is as follows.

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

- However, when this first query arrives from a previously unknown RDNS server, the ADNS server responds to a query with a CNAME type record for a new hostname within our domain. After receiving the subsequent query for this new hostname we repeat the process four additional times. When this batch of five CNAME queries leads to the discovery of at least one new RDNS server then the entire process is repeated with five additional CNAMEs. This process continues until no new RDNS

server is found, at which point a special A record is returned to the client indicating that new RDNS servers were discovered.

- When the A record returned—through the ODNS server—indicates new RDNS servers were discovered, the client sends five more probes for distinct hostnames to the same ODNS server. 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.

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

Our $S_1$ scan uses the above procedure. Furthermore, to enable exploration of alternate scanning strategies—as well as to discover RDNS pools in § 3.5—our $S_2$ scan uses a modified version of the above procedure that triggers a new CNAME batch as long as new RDNS servers are discovered *for the current ODNS server* rather than consulting the full set of RDNS servers 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 server in that block is found. The idea behind the last strategy is that the ODNS servers in a /24 block will all share the same RDNS infrastructure and so the first ODNS server will trigger the discovery of the lion's share of the RDNS servers. The results for all four strategies reflect simulations driven by the dataset collected by the $S_2$ scan.

Figure 3.8 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 ODNS servers within the same /24 address block do not all use the same RDNS server or

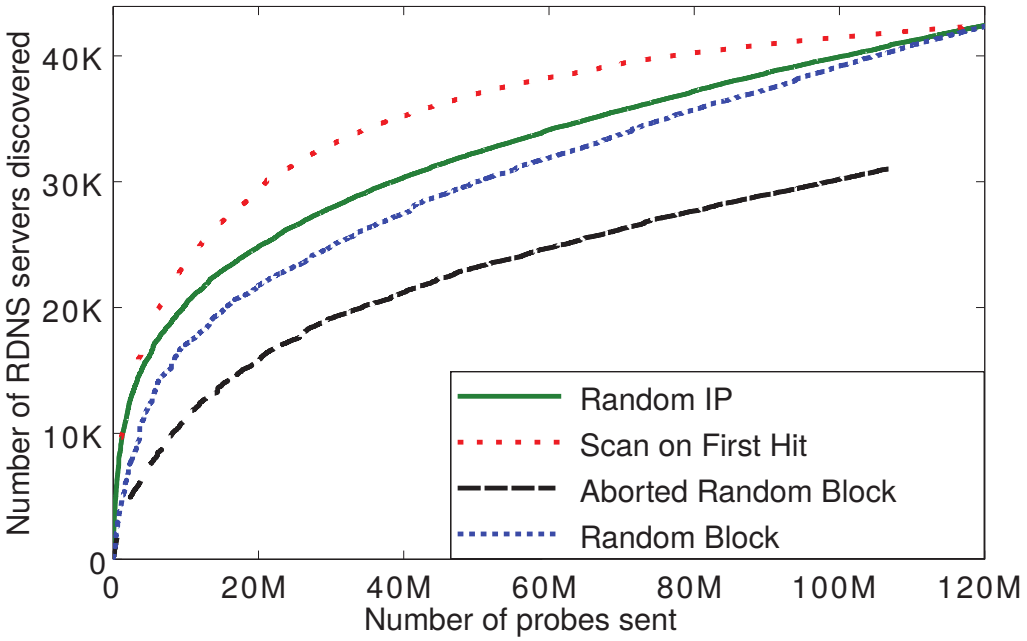


Figure 3.8: RDNS discovery rate versus DNS requests sent, simulated from the $S_2$ scan.

RDNS pool—contrary to our expectation. Therefore, learning about the "neighborhood" is beneficial not only for ODNS discovery but also for RDNS discovery. This accounts for "Scan on First Hit" achieving a higher RDNS discovery rate than Random IP. Second, because the vast majority of /24 blocks do not contain any ODNS servers, 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 RDNS servers within the $S_2$ dataset. The undiscovered RDNS servers were not reachable through the first ODNS server 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.

### RDNS$_d$ Evaluation

The RDNS$_d$ servers deserve special attention here. These servers have been counted during both ODNS and RDNS discovery. Yet, unlike RDNS$_i$ servers, it is unclear if all RDNS$_d$ servers have 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

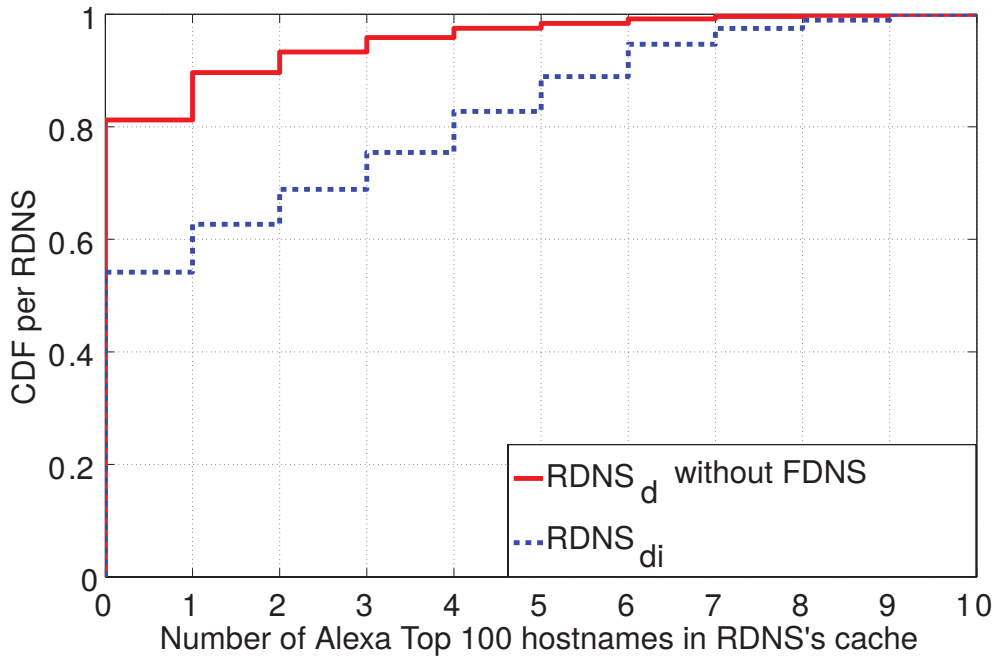


Figure 3.9: Number of the Alexa top 100 Web sites in the caches of RDNS servers.

[GSG02]. We find that 51% of the RDNS$_d$ servers in the $S_1$ scan are used by at least one FDNS, i.e., are in fact RDNS$_{di}$ servers. The remaining 49% could be resolvers which might be accessed by origins directly, or whose client FDNS servers are hidden from our scans' view or have been missed by our scans.

To determine if the 49% (17K) of RDNS$_d$ servers which are *not* used by any FDNS servers in our dataset from the $S_1$ scan are actually acting as resolvers for some client population we query them for the top 100 Web sites as listed by Alexa [Ale]. If the RDNS$_d$ servers 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$_d$ servers' 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 server—which we establish 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$_d$ server's cache, suggesting that some real client previously

requested the record. Figure 3.9 shows the distribution of the number of popular Web site records that appear to be in the caches of RDNS$_d$ servers *without* FDNS servers and in the caches of RDNS$_{di}$ servers. Although we will later show in § 4.3.3 that some RDNS servers are prone to inaccurate reporting of TTLs, the difference between the two curves indicates a difference in the behavior of the two sets of RDNS servers. We opt to remove RDNS$_d$ servers without FDNS servers from our analysis since their purpose is unclear. Instead, we focus the remainder of our study upon RDNS$_i$ servers which have a clear purpose within the client-side DNS infrastructure.

## 3.5 Topology

In this section, we present our findings on the size and structure of the client-side DNS infrastructure.

### 3.5.1 Estimating Global ODNS Population

Extrapolating from our limited scans of IP space, we estimate that there are approximately 32M ODNS servers 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 IP addresses in the "Random IP" scan $S_1$ where addresses are chosen randomly from the complete $2^{32}$ address space. Therefore, we estimate the population size as $2M/254.7M \times 2^{32} = 33M$ ODNS servers across the Internet. Second, from the "Random /24" scan $S_2$, the fraction of productive /24 address blocks (those with at least one ODNS server) is $0.141$ and a productive block contains on average 13 ODNS servers. Therefore, the ODNS population across the entire Internet is $0.141 \times 13 \times 2^{24} = 31M$.

These estimates significantly exceed previous results of complete Internet scans [LL10] and estimates [Sis10], which show around 15M responding DNS resolvers. One of our *partial* scans ($S_3$) using the "Scan on First Hit" strategy directly identifies 17.6M

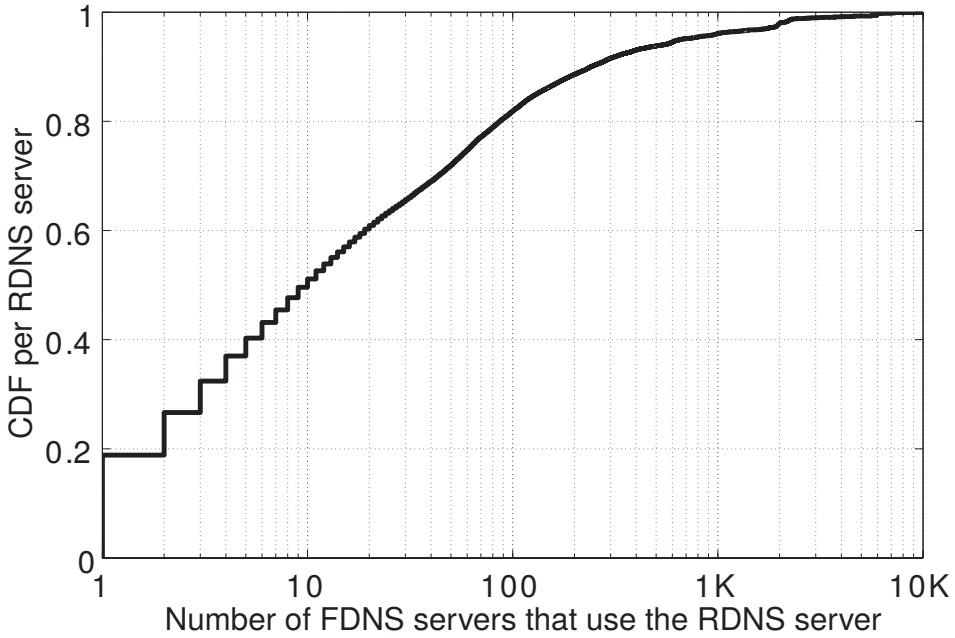


Figure 3.10: Number of FDNS servers per RDNS$_i$ in the $S_2$ scan.

ODNS servers—more than found in previous full scans. Additionally, [Opea], a complete Internet scan, reports 33M open resolvers as of May 2013 which agrees with our estimate. The results show the population of ODNS servers on the Internet has increased dramatically since previous studies. We do not have an explanation for this increase, and note that since our scans, [Opea] has observed a drop in the number of open resolvers to approximately 19M at the time of writing.

### 3.5.2 FDNS Population Size Per-RDNS

We find that many FDNS servers use the same RDNS servers. Figure 3.10 shows the number of FDNS servers per RDNS$_i$ server in the $S_2$ dataset. Over 80% of the RDNS$_i$ servers are used by more than one FDNS server and 50% of RDNS$_i$ servers appear with at least 10 FDNS servers in the dataset. This result indicates that many FDNS servers use the same resolution infrastructure upstream. While expected, it is an important observation to keep in mind when performing measurements.

### 3.5.3 RDNS Pool Sizes

The ODNS servers we find in both our $S_1$ scan and $S_2$ scan utilize RDNS servers in roughly 99% of the cases—i.e., they are in fact FDNS servers. Moreover, approximately 70% of FDNS servers use an RDNS pool in both scans. Per § 3.4.2, we use repeated DNS requests and CNAME chaining triggered by per-ODNS discovery of a new RDNS server to identify RDNS pools used by an FDNS server. Figure 3.11 shows the size distribution of the discovered RDNS pools in the $S_2$ scan. The plot shows that 10% of FDNS servers 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 server 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 an alternate DNS infrastructure for some queries. The pools we discover are smaller than those discovered by a previous study [Cal12] where roughly 20% of RDNS pools are larger than 10 servers. The difference may be due to the timescale of the experiment. We perform our discovery of RDNS servers immediately after discovering the FDNS server, whereas the previous work continues to probe over an extended period of time (up to 1 billion seconds). If FDNS server to RDNS server assignment changes with time, [Cal12] will discover all RDNS servers used and interpret them as belonging to a single pool, while our measurements will only discover the RDNS servers that are used by the FDNS server at the time that the FDNS server is discovered.

### 3.5.4 Distance between FDNS servers and RDNS servers

As discussed in § 3.4.1, ODNS servers—and consequently FDNS servers—are predominantly in residential settings and thus close to the client. Several previous studies look at the distance between clients and the RDNS servers that they use because ($i$) large distances can lead to high DNS latency and ($ii$) content delivery networks assign clients to replicas that are near the RDNS server based on the assumption that clients are also near to their

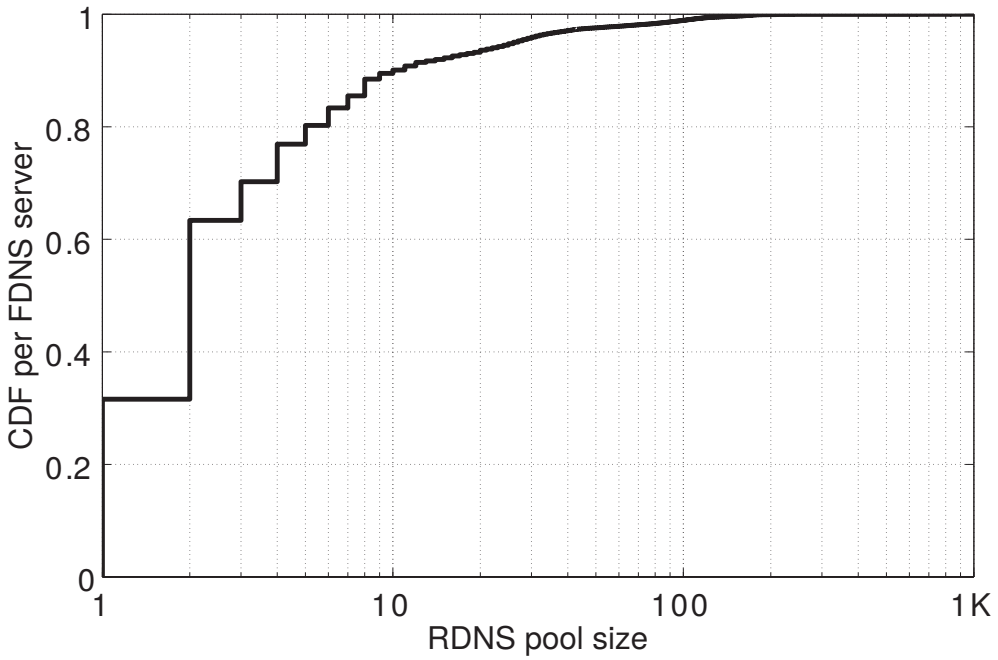


Figure 3.11: Distribution of the RDNS pool size for each FDNS in the $S_2$ scan.

RDNS server.

Prior studies [HMLG11, ARS13, Cal12] consider the physical distance between the clients and their DNS resolvers, finding that some clients are far from the resolvers that they use. We measure the distance between FDNS servers and RDNS servers in terms of latency. For this, we measure (1) the rough-trip-time (RTT) from our measurement point to the FDNS server and (2) the RTT from our measurement point to the RDNS server through the FDNS server. The difference between (2) and (1) is the RTT between the FDNS server and the RDNS server. We cannot measure RTT to the FDNS server by querying for a record previously placed in the FDNS server's cache because there is no way to determine whether the record is returned via the FDNS server's cache or an RDNS server's cache. Fortunately, we found that some FDNS servers respond to ICMP Echo requests. In the $S_5$ scan, we obtained (1) the RTT from our measurement point to the 22% of the FDNS servers that were responsive to a ping.

To measure (2), the RTT between our measurement point and the respective RDNS

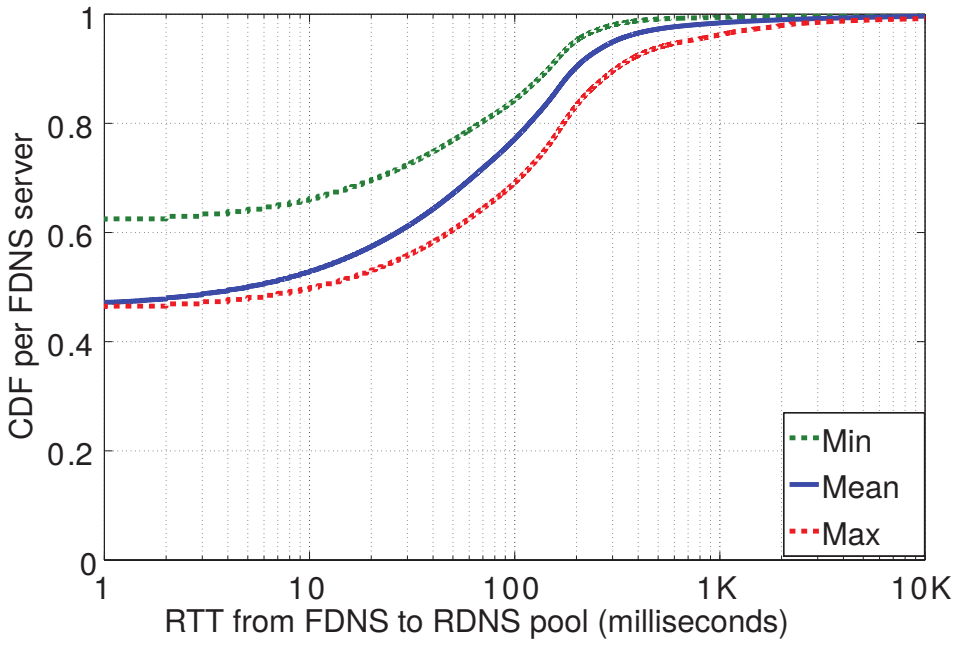


Figure 3.12: Distribution of round trip time between FDNS and RDNS.

server through the FDNS server, we leverage our observation that many FDNS servers use the same RDNS server (§ 3.5.2). Using 2 FDNS servers that both query the same RDNS server upstream, we add a record to the RDNS server's cache via a probe from the first FDNS server. We then obtain the distance from our measurement point to the RDNS server through the second FDNS server by querying for the same record through the second FDNS server. 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 server to RDNS server for 5.6M FDNS/RDNS pairs across 1.3M unique FDNS servers.

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 servers. We plot the results in Figure 3.12. The median round trip time is about 10 ms, however nearly 20% of the FDNS servers experience delays in excess of 200 ms to at least one of their RDNS servers. While the delay may be more closely related to net-

work conditions and server load than transmission time and distance, these FDNS servers nevertheless suffer a high cache miss cost.

## 3.6 Summary

In this chapter, we present a set of methodologies for efficiently discovering the client-side DNS infrastructure. Using these methodologies, we assess various aspects of the client-side DNS 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 round-trip-times that DNS messages travel within the infrastructure.

# Chapter 4

# Measuring Behavior in the DNS Infrastructure

In the previous chapter, we demonstrated the complex topology of the client-side DNS infrastructure[1]. This complexity makes it difficult to understand the behavior of the resolving infrastructure and to attribute responsibility for distinct behaviors to the individual actors. In this chapter, we develop measurement techniques for teasing apart the behavior of the actors within the system, some of whom cannot be accessed directly. Then, 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:

- 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

[1]This work originally appeared in [SCRA13].

[PAS+04] 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 [CAR13]. 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 the cause of eviction. We find scant evidence of a general capacity limitation problem .

How the client-side DNS infrastructure handles TTLs and caching are of particular importance because of its effect on the widely used DNS-based approach to scalability and reliability of Internet platforms. DNS is used to distribute load across servers—by providing directing different clients to one of several content replicas—and to mitigate downtime when individual servers fail—by changing the assignment of client to replica with time. How long records remain in cache directly impacts how quickly content providers may react to failures and balance load between replicas. Several studies consider DNS caching behavior [PAS+04, SKAT12, CAR13] and we show the different ways in which distinct actors frequently misrepresent how long name-to-address bindings should be valid and, in some cases, discover actors who retain the name-to-address bindings beyond validity.

## 4.1 Related Work

In this work, we explore behaviors involving the TTL. Liston, et al. [LSZ02] report on the behavior of the DNS in how TTL values are set at ADNS servers. The authors show low variation in the distribution of TTL values assigned per record suggesting that different operators assignment similar TTL values.

Several other works find that the authoritative TTL value is often violated [PAS+04,

SKAT12, Cal12, CAR13]. Pang, et al. [PAS+04] find that, while most resolution paths do not violate TTL, the resolution paths that do violate TTL do so by a large amount. Shue, et al. [SKAT12] find that many TTL violations are caused by Web browsers pinning name-to-address bindings to mitigate cross-site scripting attacks. Callahan [Cal12] shows that TTL violations occur in the resolution paths of ODNS servers. Finally, Callahan, et al. [CAR13] provide evidence in 2013 that the number of violations as well as their size is decreasing. We expand upon previous works by showing which actors within the infrastructure are responsible for modifications of the TTL.

## 4.2 Techniques for Untangling Behavior

We now discuss techniques that we developed to tease apart the behavior of FDNS from RDNS. We maintain our non-interference with normal operation requirement.

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 work is measurement techniques to isolate FDNS behavior from RDNS and HDNS behavior. Through cache injection[2] on FDNS servers, to which we found a sizable fraction of FDNS servers are susceptible, we short-circuit HDNS and RDNS from processing a measurement probe. Therefore, any artifacts are the sole result of the FDNS server. Similarly, we develop a technique of coordinated probing through two or more FDNS servers to determine the behavior of a shared RDNS server in near isolation from FDNS behavior. We validate the latter technique using the $\text{RDNS}_{di}$ servers —which we can probe both through an FDNS server and directly—as ground truth of RDNS behavior. Estimating from our experiments, over 77% of $\text{RDNS}_i$ servers 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.

[2] A technique for inserting records into a DNS cache against the spirit of the protocol. See § 6 for details.

### 4.2.1 Measuring FDNS Servers

The vast majority of ODNS servers we discovered in our mapping are in fact FDNS servers (over 95% across all of the datasets). Gaining an understanding of FDNS servers in isolation from the remainder of the client-side DNS infrastructure is a challenge. Fortunately, we find that a fraction of FDNS servers allow a primitive form of cache injection which we leverage to gain insight into the behavior of this FDNS server subset. Specifically, these FDNS servers 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 server with an acceptable response which only involves the FDNS server and not the rest of the infrastructure. We discuss the cache injection attacks in general and the attack we use for measurement in specific within Chapter 6.

Hence, we can study these FDNS servers in isolation from HDNS servers and RDNS servers using the following procedure. We begin by sending a DNS request for a hostname within our domain to the FDNS server under study and then immediately issue a DNS response for the same hostname to the FDNS server that binds the requested name to IP address $X$. On the other hand, when the request arrives to our ADNS server 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 server cache and the FDNS server is effectively isolated from the rest of the client-side infrastructure, which never touched record $X$. We stress that the FDNS servers we study may be a biased set since the set only includes FDNS servers that exhibit the cache injection vulnerability.

### 4.2.2 Measuring RDNS Servers

Since typically we cannot query RDNS$_i$ servers directly, we utilize FDNS servers as our window into RDNS$_i$ behavior. This, however, poses a problem as FDNS servers may alter DNS requests, responses and caching phenomena, hence obscuring RDNS behavior. A second response for a domain name through a single FDNS server may be returned from either the FDNS server cache or the RDNS$_i$ server cache, ambiguously. Fortunately, it is common to find multiple FDNS servers which use the same RDNS$_i$ server. We leverage a general experimental strategy which requires at least two FDNS servers per RDNS$_i$ server 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 server responds to this query with a randomly generated record, which should be cached at the RDNS$_i$ server 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$_i$ server 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$_i$ server cache and not from $F_2$ because the request was previously unseen by $F_2$. If the record is no longer in the RDNS$_i$ server cache, the request will arrive at our ADNS server, 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 FDNS servers which use the same RDNS$_i$ server at roughly the same time. In § 3.5.2, we find that over 80% of RDNS$_i$ servers are used by more than one FDNS server and the coordinated probing technique has a chance of succeeding. Further, 50% of RDNS$_i$ servers appear with at least 10 FDNS servers in the dataset, vastly increasing the chances of successful measurement.

FDNS server behavior may still distort the measurement by altering records. We discuss ways to mitigate this problem—such as using all available FDNS servers— in § 4.3.

## 4.3 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 [PAS+04, SKAT12, CAR13]. With regard to security, caching determines the lingering effect of a successful record injection (e.g., using the Kaminsky attack [Kam08]). TTLs and the subsequent pinning of clients is of particular importance to content delivery networks (CDNs) which attempt to redirect clients among content replicas via dynamically adjusting hostname-to-address bindings. Thus, DNS caching determines how rapidly CDNs may adjust assignments of clients to replicas.

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 $S_5$ scan (see Table 3.1). The scan covers 79M IP addresses with 1.3B DNS requests from 2/26/2013 through 3/28/2013. The $S_5$ scan encompasses 11M ODNS servers and 66K RDNS servers—46K of which are RDNS$_i$ servers. The $S_5$ dataset uses the "Scan on First Hit" methodology to increase the ODNS

discovery rate and thus the probability of finding multiple FDNS servers which use the same RDNS$_i$ server at roughly the same time. This assists with the coordinated probing strategy sketched in § 4.2.2. In the next section, we describe our observations of the aggregate behavior of the resolution path and then tease apart the behaviors and attribute them to various actors in the following sections.

### 4.3.1 Aggregate Behavior

To investigate aggregate behavior of the DNS resolution infrastructure, we perform in-depth probing of 2.4M FDNS servers during the $S_5$ scan. We did not test all of the FDNS servers because of limitations in the amount of traffic our ADNS server can handle. Therefore, we limit the number of FDNS servers which can be concurrently assessed to 25K per measurement origin (PlanetLab node). When a measurement origin finds a new FDNS server we skip in-depth measurement if 25K FDNS servers are presently being assessed.

We examine how FDNS servers and RDNS$_i$ servers 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 intervals taken from the same series of values as the TTLs to determine whether the records are still available in cache. 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 server 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 resolution infrastructure, either at the FDNS, HDNS, or RDNS$_i$ servers.

First, we observe that some of the responses to our queries arrive at the measurement origins with TTL values differing from those set by our ADNS server. Furthermore, we find cases where the correct TTL is returned in the *initial* response for a hostname and

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

Table 4.1: TTL behaviors for the whole resolution path.

an incorrect TTL is returned in *subsequent* responses for the same hostname. This behavior was also observed in [Cal12] and is interpreted as the DNS actor distorting TTL not when conveying the record back to the requester but when storing the record in its cache. Table 4.1 summarizes our results for aggregate behavior. In total, 19% of the FDNS servers and their underlying infrastructure always report the correct TTL value. Meanwhile, 38% of the FDNS servers respond with an incorrect TTL value to an initial request. The remaining 62% of FDNS servers honestly report the TTL on at least the first DNS request. This will be important in studying $\text{RDNS}_i$ servers below.

Beyond changing the TTL of DNS records, we find some FDNS servers do not correctly decrement the TTL that they return for records in cache[3]. We find that 7% of the FDNS servers return a constant TTL value, without ever decrementing, roughly agreeing with a previous study that found 5% of FDNS servers return constant TTL values [Cal12]. An additional 1% of the FDNS servers actually begin to *increment* the TTL after counting down to zero! Both these behaviors will eventually result in a TTL which disagrees with the one initially set by the ADNS server. Downstream devices—regardless of how they treat the TTL themselves—may unknowingly use such records in violation of the TTL set by the ADNS server.

Table 4.2 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 modification (Mode Lie) and the percentage of the lies the

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

| Expected (sec) | % < | % > | Mode Lie | |
|---|---|---|---|---|
| | | | Value | % of All Lies |
| 1 | 0% | 11% | 10000 | 35% |
| 10 | 0% | 8% | 10000 | 37% |
| 30 | 0% | 6% | 10000 | 43% |
| 60 | 0% | 4% | 10000 | 48% |
| 100 | 1% | 4% | 10000 | 48% |
| 120 | 1% | 3% | 10000 | 55% |
| 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 4.2: TTL deviations for the whole resolution path.

mode represents. For example, we find 11% of FDNS servers 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). These results confirm previous observations [Cal12] that authoritative TTLs of 0 or 10 seconds are frequently increased to 10,000 seconds while authoritative TTLs of 1M seconds are frequently capped at 604,800 seconds (1 week) somewhere in the resolution path. In a study of DNS clients (see Chapter 7), we observe that 64% of the queried hostnames return records with authoritative TTLs of less than 1,000 seconds. According to our findings, all of these records will have their TTLs increased by 3-11% of resolution paths, potentially leading to caching violations.

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 4.1 shows the length of time records remain in some cache within the

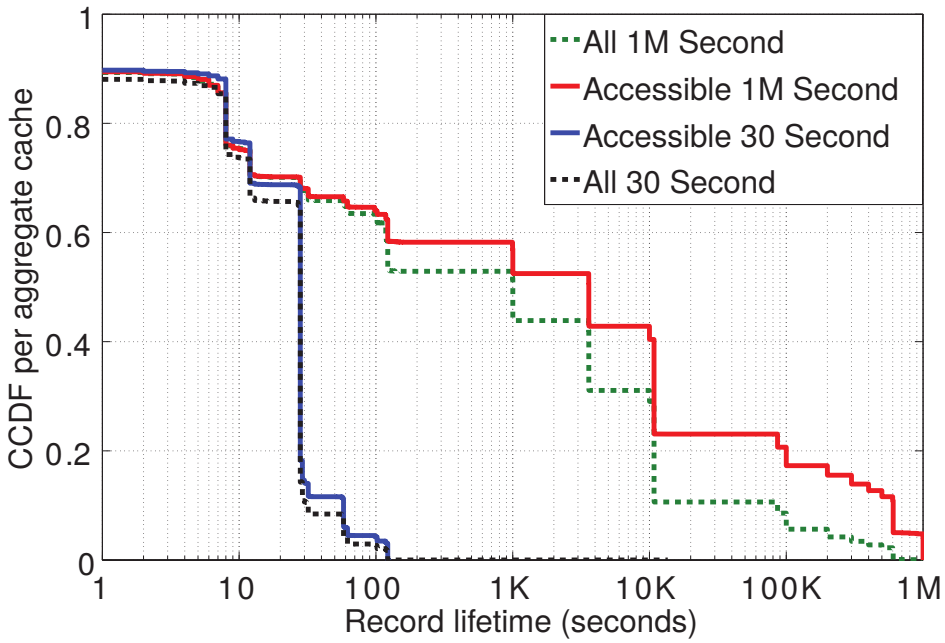


Figure 4.1: Distribution of record availability in the whole resolution path for records that have TTLs of 30 and 1 million seconds.

resolution infrastructure. We present results for 30 second TTL records, a short TTL value similar to those used by content delivery networks (e.g., we observe that Akamai returns records to us with a 20 seconds TTL and Limelight returns records with a 350 second TTL). Additionally, we use 1 million second TTL records, 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 server. However, a number of FDNS servers become unreachable in the course of the experiment. When this happens due to IP address reassignments as discussed in § 3.4.1, 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 FDNS servers that remain accessible throughout the experiment (the "Accessible" line). The true cache duration lies somewhere between these two lines.

Our results show that 90% of FDNS servers and their supporting infrastructure retain 30 second TTL records for no longer than the TTL—with 60% of the FDNS servers

retaining the record for the full 30 seconds. We find that 10% of FDNS servers 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 [PAS+04] suggesting that DNS behavior has changed since 2004.

The records assigned a TTL of 1 million seconds show much longer retention, with over 40% active for more than 1 hour. This indicates that short cache retention of the 30 second TTL records is due to the TTL setting rather than cache capacity constraints. Another study of cache retention in resolution paths [Cal12] finds that a record also with a 1 million second TTL remains in cache over 1 hour for roughly 70% of resolution paths. The difference is likely due to reprobing policy. Before the 1 hour mark, our experimental setup issues 7 queries to examine whether the record is still cached as compared to 11 queries in the [Cal12]. Further, the most recent probe before the 1 hour mark occurs at 1,000 seconds in our study and 2,048 seconds in [Cal12]. Thus, the records in [Cal12] are queried more frequently and more recently than the records in our study. Assuming a caching policy of evicting the least recently used or least frequently used record, the records in [Cal12] are more likely to be "saved" from eviction.

### 4.3.2 FDNS Server Behavior

We now study the behavior of FDNS servers in isolation by using the record injection technique described in § 4.2.1. We find 683K FDNS servers in the $S_5$ dataset (6%) accept our injected response records and were thus amenable to our measurement.

We utilize the same experimental technique as in § 4.3.1 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 server, so whenever $X$ appears in a DNS response we know the request was satisfied from the FDNS'

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

Table 4.3: TTL behaviors for the FDNS servers in isolation.

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 § 4.3.1.

Table 4.3 summarizes our general findings on FDNS' TTL behavior. First, 60% of the FDNS servers never lie with respect to the TTL. However, we find that 12% of FDNS servers lie in response to the initial request and 30% lie in response to the subsequent requests. As we 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 FDNS servers report a constant TTL value without ever decrementing it. Finally, 10% of the FDNS servers began to increment the TTL value upon decrementing it to zero.

Table 4.4 shows the TTL deviations we observe for FDNS servers that allow cache injection, 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. We conclude that FDNS servers that allow cache injection are less likely to cap the TTL value than other actors. 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 server 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 FDNS servers. Using repeated probes at the intervals mentioned at the beginning of § 4.3.1, we

| Expected (sec) | % < | % > | Mode Lie | |
|---|---|---|---|---|
| | | | Value | % of All Lies |
| 1 | 0% | 31% | 10000 | 88% |
| 10 | 0% | 19% | 10000 | 95% |
| 30 | 0% | 19% | 10000 | 96% |
| 60 | 0% | 19% | 10000 | 99% |
| 100 | 1% | 19% | 10000 | 97% |
| 120 | 1% | 19% | 10000 | 97% |
| 1000 | 1% | 19% | 10000 | 97% |
| 3600 | 1% | 19% | 10000 | 97% |
| 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 4.4: TTL deviations for the FDNS servers in isolation.

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 server with the TTLs set to 30 seconds and 1 million seconds. Figure 4.2 shows how long the records are retained in the caches. See § 4.3.1 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 FDNS servers.

The results show that roughly 40% of the FDNS servers retain the record with a TTL of 30 seconds for longer than the TTL. Additionally, 28% of FDNS servers hold the record for over 100 seconds, more than twice the TTL. This deviates from the aggregate results in Figure 4.1 indicating that FDNS servers which allow cache injection are more likely to retain records past the TTL value than the general FDNS population. We find that 1 million second TTL records are retained for at least 10,000 seconds in 50% of the FDNS servers.

As an aside, we observed that 47K FDNS servers (6.9%) returned the injected record in response to the first DNS request but did not return the injected record for any subsequent requests. Additionally, 43K of these FDNS servers always reported the correct TTL. This likely indicates that these 43K FDNS servers do not have caches and are pure forwarders

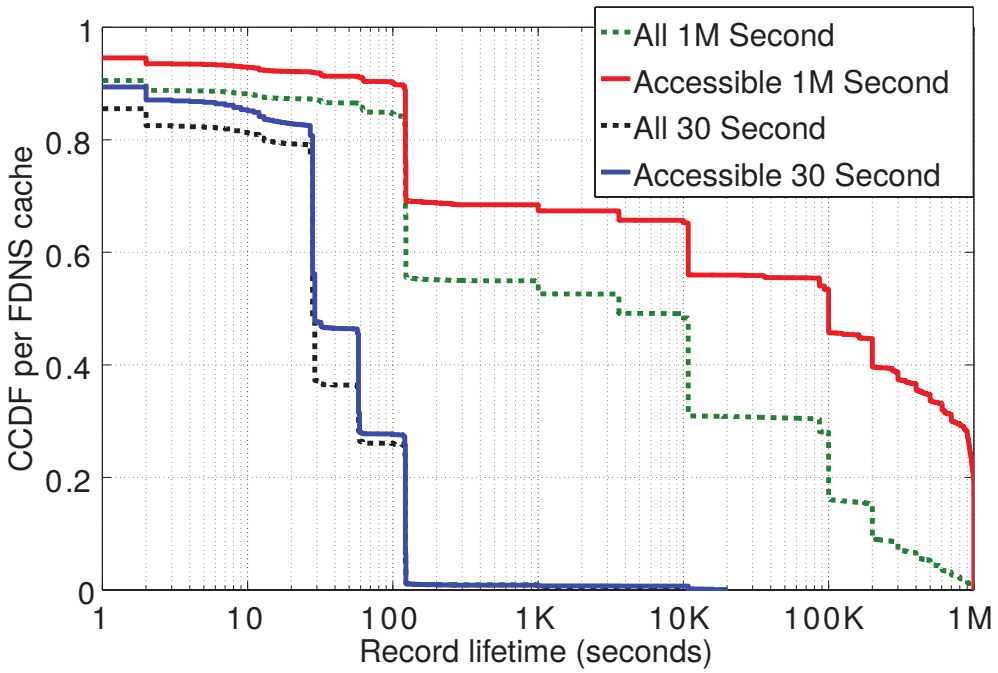


Figure 4.2: Distribution of record availability in FDNS caches for records that have TTLs of 30 and 1 million seconds.

that do not perform any application layer logic on the DNS datagrams they receive.

### 4.3.3 RDNS Server Behavior

We now turn to the behavior of RDNS$_i$ servers in near isolation from FDNS servers. RDNS$_{di}$ servers 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$_{di}$ servers. We use the same experimental technique as in § 4.3.1 with the same TTL values and re-probe intervals. Table 4.5 shows our general results while Table 4.6 shows the TTL deviations we observe. RDNS$_{di}$ servers do not exhibit either the constant TTL nor the pathological TTL incrementing behavior we observe in FDNS servers and the aggregate data. However, there is more TTL distortion with virtually no consistently honest RDNS$_{di}$ servers. The most common TTL distortion is to lower very large TTL values (e.g., 88% of RDNS$_{di}$ lower TTLs of 1M seconds).

For RDNS$_i$ servers that do not respond to direct probes, we leverage the coordinated

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

Table 4.5: TTL behaviors for the $\text{RDNS}_{di}$ servers in isolation.

| Expected (sec) | % < | % > | Mode Lie | |
|---|---|---|---|---|
| | | | Value | % of All Lies |
| 1 | 0% | 22% | 3600 | 53% |
| 10 | 0% | 22% | 3600 | 52% |
| 30 | 0% | 22% | 3600 | 52% |
| 60 | 0% | 22% | 3600 | 54% |
| 100 | 0% | 22% | 3600 | 53% |
| 120 | 0% | 22% | 3600 | 53% |
| 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 4.6: TTL deviations for the $\text{RDNS}_{di}$ servers in isolation.

probing strategy to assess their violations. As described in § 4.2.2, we begin by requesting a unique hostname via FDNS $F_1$. Our ADNS server responds to this query with two IP addresses, one random and the second uniquely bound to the RDNS$_i$ server from which the DNS request arrives. We assign the same set of TTL values as in § 4.3.1. Both records pass through RDNS$_i$ server 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$_i$ server 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$_i$ server, the FDNS servers or even some HDNS in the path. To gain confidence in our attribution of TTL deviations to RDNS$_i$ server, we leverage two pieces of information. First, as noted in the previous two sections, a significant number of FDNS servers are honest on the initial DNS response. We find 62% of FDNS servers are honest on the initial response in § 4.3.1 and 88% of FDNS servers are honest on the initial response in § 4.3.2. Second, we can utilize more than two FDNS servers in coordinated probing to mitigate the effect of FDNS lies. Instead of $F_1$ and $F_2$ representing single FDNS servers in the above description of the experiment, we utilize up to 10 FDNS servers 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$_i$ server 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$_i$ server is responsible for the lie. In this situation, there are three scenarios:

- First, some of the FDNS servers are in the set of FDNS servers that were honest in initial responses *and* the TTL values from these FDNS servers all agree. In this ideal case, their responses collectively identify the actual TTL the RDNS$_i$ server provides.

- In the second scenario, the TTL values arriving at honest FDNS servers do not agree. One potential cause of this case is HDNS servers interposing between some of the

FDNS servers and the RDNS$_i$ server. In this case, we assume that the RDNS$_i$ server returns the most common TTL value among the honest FDNS servers.

- In the final scenario, none of the FDNS servers are honest. In this case, we assume that the RDNS$_i$ server returns the most common TTL value among all the FDNS servers.

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

We validate our technique for determining RDNS$_i$ TTL behavior using RDNS$_{di}$ servers, which allow us to obtain ground truth by direct probing. Specifically, if through direct probing of the RDNS, we discover that it does not deviate from the ADNS server assigned TTL value and, through coordinated probing, we discover the same, that is a case of agreement. On the other hand, if our coordinated probing concluded that the RDNS$_{di}$ deviated, then that is a case of disagreement. Similarly, if direct probing indicates a TTL deviation and coordinated probing indicates the same TTL deviation, that is also a case of agreement. 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$_i$ server is honest but also in determining the quantitative TTL violations (as we will show, most lies are from a small fixed set of TTLs). Therefore, we find that our dataset contains a sufficient number of FDNS servers that respond to an initial request with the correct TTL, regardless of whether we were able to identify them as such in the previous two sections, per RDNS$_i$ server to attribute TTL behavior to RDNS$_i$ servers.

In our dataset, there are 46K RDNS$_i$ servers and we conduct in-depth probing for 22K of them (due to logistical issues, as sketched above). Table 4.7 shows our findings. We determine that 36% of RDNS$_i$ servers are honest. Further, 55% of RDNS$_i$ servers lie on the

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

Table 4.7: TTL behaviors for the RDNS$_i$ servers.

| Expected (sec) | % < | % > | Mode Lie | |
|---|---|---|---|---|
| | | | Value | % of All Lies |
| 1 | 0% | 1% | 300 | 34% |
| 10 | 0% | 1% | 300 | 34% |
| 30 | 0% | 1% | 300 | 37% |
| 60 | 0% | 1% | 300 | 44% |
| 100 | 1% | 1% | 300 | 41% |
| 120 | 1% | 1% | 300 | 48% |
| 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 4.8: TTL deviations for the RDNS$_i$ servers.

initial response to $F_1$, but only 5% of RDNS$_i$ servers 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$_i$ servers than FDNS servers. This supports our conjecture that FDNS servers, being mostly home-based devices, represent more primitive implementations of DNS. In addition to incorrect TTLs, we find 8% of RDNS$_i$ servers return constant TTL values without decrementing. The TTL deviations from RDNS$_i$ servers, merged with the results for RDNS$_{di}$ servers are shown in Table 4.8. When RDNS$_i$ servers do lie, virtually all lies occur at high TTL values. For authoritative TTLs set less than 1 hour, there is little deviation attributable to RDNS$_i$ servers.

We now consider how long a record remains cached and accessible at RDNS$_i$

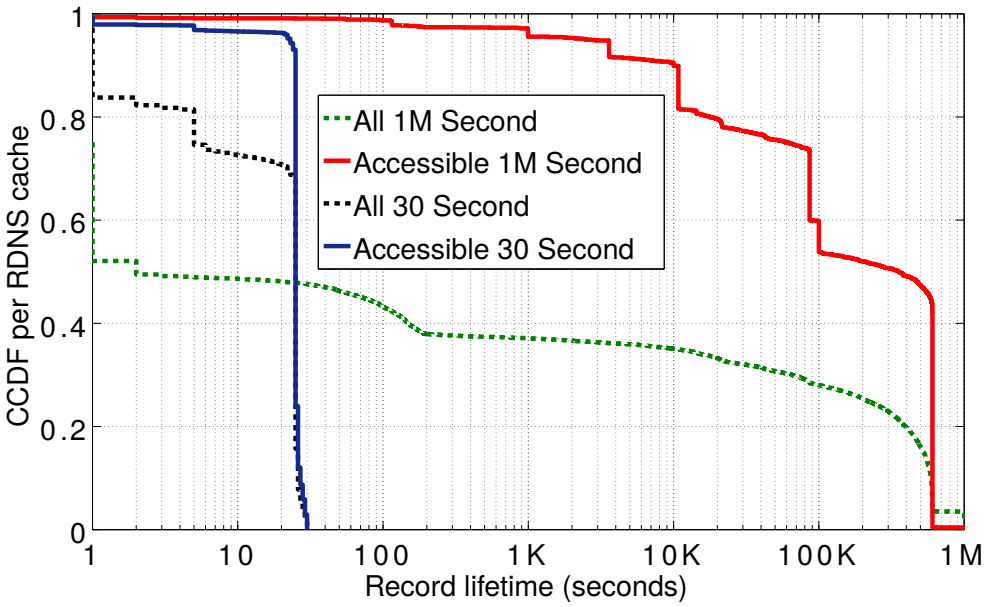


Figure 4.3: Distribution of record availability in RDNS$_i$ caches for records that have TTLs of 30 and 1 million seconds.

servers. 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 § 4.3.1 up to the TTL value[4]). For each RDNS$_i$ server, we track the latest time at which the record is available.

Again, when FDNS servers become unavailable in the course of the experiment we cannot detect how much longer a record stays in the RDNS server's cache. Thus, Figure 4.3 shows the duration of record retention separately for all measured RDNS servers (the "All" lines, representing 22K tested RDNS$_i$ servers and underestimating the result) and for those RDNS servers that remained accessible throughout the experiment (the "Accessible" lines, representing 8.8K and 2.4K RDNS$_i$ servers for the 30 second TTL and the 1 million second TTL, respectively).

Concentrating on the "Accessible" lines as more reliable, we show that 1 million second TTL records 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

[4]Unfortunately, we neglected to measure intervals beyond TTL and hence do not check if RDNS servers cache records beyond the authoritative TTL as we did for FDNS servers.

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 4.1) is likely due to FDNS affects. The 30 second TTL record remains in the cache for the full TTL in 94% of the tested RDNS$_i$ servers.

### 4.3.4 HDNS Server Behavior

We found in the previous two sections, that some FDNS servers and RDNS$_i$ servers report the TTL assigned by the ADNS server honestly. Thus, an honest FDNS server that forwards queries to an honest RDNS server should always report the correct TTL. However, we find occasions of TTL deviation when both the FDNS server and RDNS server are honest, indicating the presence of an intermediary (i.e., an HDNS server). Thus, we are able to observe HDNS behavior without ever directly communicating with the HDNS server from either a measurement origin or our ADNS server. For 2.3K honest FDNS servers using 179 honest RDNS servers, we observe TTL deviations for at least 1 TTL value. Note that we cannot identify distinct HDNS servers and do not know the number of HDNS servers that we observe. This result suggests cases where the HDNS server and RDNS server enact differing policies on the handling TTLs.

## 4.4 Dataset Representativeness

In this section, we turn to the issue of bias in our datasets first mentioned in § 3.3. Since our scans do not encompass the entire Internet and utilize the "Scan on First Hit" technique, it is possible that our results are not demonstrative of the entire population of FDNS servers and RDNS$_i$ servers due to biases in our scanning methodology. In particular, our results on FDNS behavior encompass a subset of FDNS servers, i.e., those which allow cache injection. Similarly, our results for RDNS$_i$ servers include only those RDNS$_i$ servers for which we discover at least two FDNS servers. 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 FDNS servers while for the RDNS$_i$ behavior, the 10 slices each include an identical number of measured RDNS$_i$ servers. 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 4.4 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$_i$ servers decreases over time but appears to converge to a constant value by the $7^{th}$ snapshot. This shows that (1) honest RDNS$_i$ servers are discovered at a higher rate towards the beginning of the scan and (2) we discover a sufficient number of RDNS$_i$ servers to be representative of the general population. Finally, the fraction of honest FDNS servers 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 FDNS servers that allow cache injection. The fraction of honest FDNS servers in the true population is likely higher than what we report here.

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 ODNS servers 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

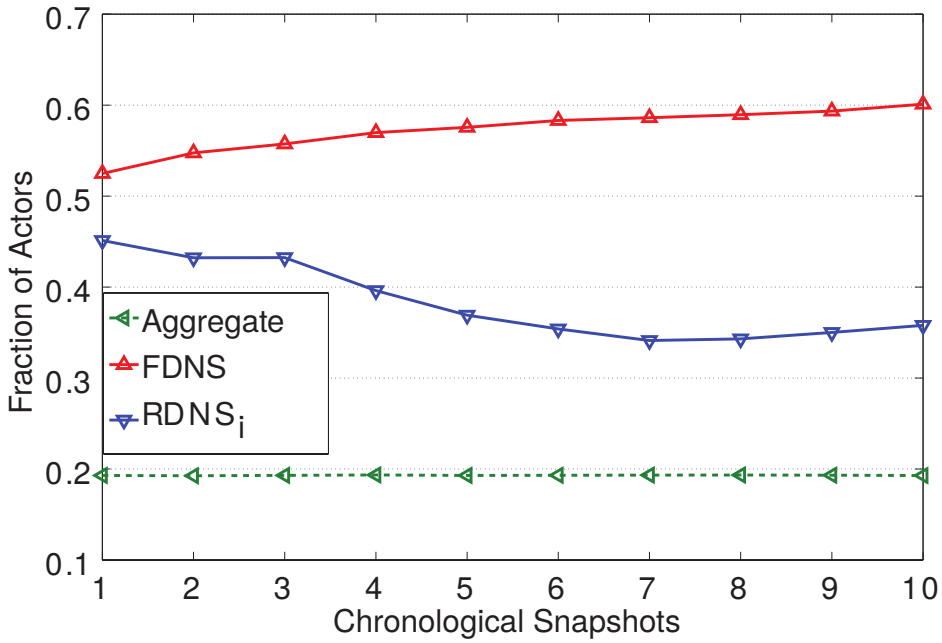


Figure 4.4: Fraction of honest actors over the discovery process.

behavior for first hop resolvers that are not arbitrarily accessible.

## 4.5 Summary

In this chapter, we present a set of methodologies for teasing apart the behavior of the actors within the system including components that cannot be directly probed. Using these methodologies, we assess the behavior of the client-side DNS infrastructure with respect to caching, both in aggregate and separately for different actors. 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$_i$ servers and the latter behavior predominantly to FDNS servers. Additionally, we find that cache evictions due to capacity limits occur infrequently in RDNS$_i$ servers 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 that have a relatively short TTL of 30 seconds.

# Chapter 5

# Characterization of DNS Client Behavior

Individual clients on the Internet issue DNS queries for a variety of different purposes leading to a complex DNS querying behavior. Previous studies show that DNS lookups precede over 60% of TCP connections [SAR14]. As a result, individual clients issue large numbers of DNS queries. Yet, our understanding of DNS query streams is largely based upon aggregate populations of clients—e.g., at an organizational [GYC+13] or at best network address translation (NAT) device level [CAR13]—leaving our knowledge of individual client behavior limited.

This chapter represents an initial step towards understanding individual client DNS behavior. We monitor DNS transactions between a population of thousands of clients and their local resolver such that we are able to directly tie lookups to individual clients. Our ultimate goal is an analytical model of DNS client behavior that can be used for everything from workload generation to resource provisioning to anomaly detection. In this chapter we provide a characterization of DNS behavior along the dimensions our model will ultimately cover and also anecdotally show promising modeling approaches.

Broadly, the remainder of this chapter follows the contours of what a model would

capture. We first focus on understanding the nature of the clients themselves in § 5.3, finding that while most are traditional user-facing devices, there are others that interact with the DNS in distinct ways. Next we observe in § 5.4 that DNS queries often occur closely-spaced in time—e.g., driven by loading objects for a single web page from disparate servers—and therefore we develop a method to gather together queries into clusters. We then turn to the number and spacing of queries in § 5.5. Finally, we tackle what hostnames individual clients lookup in § 5.6 and find that each client has a fairly distinct "working set" of names, and also that unsurprisingly hostname popularity has power law properties.

## 5.1 Related Work

Models of various protocols have been constructed for understanding, simulating and predicting traffic (e.g., [Pax94] for a variety of traditional protocols and [BC98, Mah97, CCG+04] as examples of HTTP modeling). Additionally, there is previous work on characterizing DNS traffic (e.g., [CAR13, FA13, JBB03, GYC+13]), which focuses on the aggregate traffic of a population of clients, in contrast to our focus on individual clients.

Several studies model various caches including those in DNS resolvers [JB01, JB99, BCF+99, JBB03]. The authors of [JB99] note the presence of temporal locality in request streams and we explore this notion for DNS traffic for each client and among groups of clients.

Finally, we note—as we discuss in § 5—that several recent studies involving DNS make assumptions about the behavior of individual clients or need to analyze data for specific information before proceeding. For instance, the authors of [FA13] model DNS hierarchical cache performance using a general assumption of the arrival process, while in [SAR14], the authors use simulation to explore changes to the resolution path. Both studies would benefit from a greater understanding of DNS client behavior.

## 5.2 Dataset

The dataset utilized in this chapter comes from two packet taps at Case Western Reserve University at the routers connecting the two data centers that house all five University DNS resolvers. Thus, our vantage point lies between client devices and their recursive DNS resolvers. We collect full payload packet traces of all UDP traffic involving port 53 (the default DNS port). The campus wireless network connects mobile devices via NATs and therefore we cannot isolate DNS traffic to individual clients. Hence, we do not consider this traffic in our study (although, future work remains to better understand DNS usage on mobile devices). The University Acceptable Use Policy prohibits the use of NAT on the wired networks. Meanwhile, the University offers wireless access throughout the campus. Therefore, we believe the traffic we capture from the wired network does represent individual clients. Our dataset includes all DNS traffic from two separate weeks and is further partitioned by whether the end hosts are located in the residential or office portions of the University network. Details of the datasets are given in Table 5.1 including the number of queries, the number of clients that issue those queries, and the number of distinct hostnames queried.

### 5.2.1 Calibration

During the February data collection, we collect query logs from the five campus DNS resolvers to validate our datasets[1]. Comparing the packet traces and logs we find a 0.6% and 1.8% measurement-based loss rates in the Feb:Residential and Feb:Office packet traces, respectively. We believe these losses are an artifact of our measurement apparatus given that the packets appear in the logs at the DNS resolvers. The loss rate is correlated with traffic volume: the Feb:Office dataset shows higher volume than the Feb:Residential dataset and hence the discrepancy in loss rates between the two datasets.

[1] We prefer traces over logs due to the better timestamp resolution (msec vs. sec).

| Dataset | Dates | Queries | Clients | Hostnames |
|---|---|---|---|---|
| Feb:Residential | Feb. 26 - Mar. 4 | 32.5M | 1359 (IPs) | 652K |
| Feb:Residential (filter) | Feb. 26-27, Mar. 2-4 | 16.4M | 1262 (MACs) | 505K |
| Feb:Residential:Users | | 15.3M | 1033 | 499K |
| Feb:Residential:Others | | 1.11M | 229 | 7.94K |
| Feb:Office | Feb. 26 - Mar. 4 | 232M | 8770 (IPs) | 1.98M |
| Feb:Office (filter) | Feb. 26-27, Mar. 2-4 | 143M | 8690 (MACs) | 1.87M |
| Feb:Office:Users | | 118M | 5986 | 1.52M |
| Feb:Office:Others | | 25.0M | 2704 | 158K |
| Jun:Residential | Jun. 23 - Jun. 29 | 11.7M | 345 (IPs) | 140K |
| Jun:Residential (filter) | Jun. 23-26, 29 | 6.22M | 334 (MACs) | 120K |
| Jun:Residential:Users | | 5.81M | 204 | 116K |
| Jun:Residential:Others | | 408K | 130 | 4.13K |
| Jun:Office | Jun. 23 - Jun. 29 | 245M | 8335 (IPs) | 1.61M |
| Jun:Office (filter) | Jun. 23-26, 29 | 133M | 8286 (MACs) | 1.52M |
| Jun:Office:Users | | 108M | 5495 | 1.42M |
| Jun:Office:Others | | 25.0M | 2791 | 63.1K |

Table 5.1: Details of the datasets used in the DNS client characterization study.

Further, for the day of Feb. 26th, we match the queries in the datasets to the queries in the logs to ensure we are capturing correct data. We match on the following fields that are available in the logs: $(i)$ source IP address, $(ii)$ source port number, $(iii)$ question hostname/class/type, and $(iv)$ timestamp rounded to 10 second bins. For the Feb:Residential and Feb:Office datasets respectively, 99.6% and 99.5% of the queries match to a query in the log indicating that we are capturing correct data. Manual inspection of the queries that do not match indicates edge cases where the timestamp in the logs and in the capture were not well synchronized.

### 5.2.2 Tracking Clients

Next we aim to track individual clients in the face of dynamic address assignment. Simultaneously with the DNS packet trace, we gather logs from the University's three DHCP servers. Therefore, we can map dynamically assigned IP addresses to client MAC addresses and track the client's behavior across any changes in IP address. We find that less than 3% of MAC addresses change IP address during our study and less than 0.1% of IP addresses

are assigned to more than a single MAC address. Also, note, we could not map 1.3% of the queries across our datasets to a MAC address because the source IP address in the query never appears in the DHCP logs. These likely represent static IP address allocations. Further, without any DHCP assignments and rarity of cases where the same IP address is used by multiple MAC addresses we are confident that these IP addresses represent a single host.

### 5.2.3 Timeframe

Since we consider clients in both residential and office settings, we exclude Saturday and Sunday from our datasets for a better comparison across the two settings.

### 5.2.4 Filtering Datasets

While exploring the datasets we found two anomalies that skew the data in ways that are not indicative of user behavior. First, we noticed roughly 25% of the queries requesting the TXT record for *debug.opendns.com*, 130M queries in total. (The next most popular record represents less than 1% of the lookups!) Further investigation indicates this query is not in response to users' actions, but is automatically issued to determine whether the client is using the public OpenDNS resolver (which the answer to the query indicates) [Opeb]. We observe 298 clients querying this domain name, which we assume use OpenDNS on other networks or used OpenDNS in the past. We remove these queries from further analysis. The second anomaly involves 18 clients whose prominent behavior is to query for *debug.opendns.com* and other domains repeatedly without evidence of accomplishing much work. The campus information technology department informed us that these clients serve an operational purpose and are not user-facing devices. Therefore, we remove the 18 clients as they are likely unique to this network and do not represent users.

Table 5.1 shows the impact of filtering on each dataset. In addition, we find commonality across the various partitions of our data and therefore unless otherwise noted we focus on the Feb:Residential dataset for conciseness—and discuss when the other datasets

differ as appropriate.

## 5.3 Identifying Types of Clients

Since our focus is on characterizing general purpose user-facing devices, we aim to separate them from other types of end systems. We expect general-purpose systems are involved in tasks, such as ($i$) web browsing, ($ii$) accessing search engines, ($iii$) using email, and ($iv$) conducting institutional-specific tasks[2]. Therefore, we develop the following markers to identify general-purpose hosts:

**Browsing:** A large number of websites embed Google Analytics [Gooa] in their pages, thus there is a high likelihood that regular users will query for Google Analytics hostnames on occasion while browsing the Web. We look for resolution of *www.google-analytics.com* or *ssl.google-analytics.com* to indicate browsing activity.

**Searching:** We detect web search activity via DNS queries for the largest search engines: Google, Yahoo, Bing, AOL, Ask, DuckDuckGo, Altavista, Baidu, Lycos, Excite, Naver, and Yandex. We look for resolution of either the search engine domain name (e.g., *google.com*) or the hostname (e.g., *www.google.com*).

**Email:** Case uses Google to manage campus email and therefore we use queries for "mail.google.com" to indicate email use.

**Institutional-Specific Tasks:** Case uses a single sign-on system for authenticating users before they perform a variety of tasks and therefore we use queries for the corresponding hostname as indicative of user behavior.

Table 5.2 shows the breakdown of the clients in the Feb:Residential dataset. Of the 1,262 clients we identify 1,033 as user-facing based on at least one of the above markers.

[2]In our case, this is campus-life tasks, e.g., checking the course materials portal.

| Marker | Clients | % |
|---|---|---|
| All clients | 1262 | 100% |
| Google analytics | 983 | 78% |
| Search engine | 1010 | 80% |
| Google | 1006 | 80% |
| Any other | 602 | 48% |
| Gmail | 881 | 70% |
| LDAP Login | 840 | 66% |
| Any | 1033 | 82% |
| At least two | 991 | 79% |

Table 5.2: Feb:Residential clients that fit markers for general purpose devices.

Intuitively we expect that multiple markers likely apply to most general purpose systems and in fact we find at least two markers apply to 991 of the clients in our dataset. Results for our other datasets are similar.

We next turn to the 229 clients ($\approx 18\%$) that do not match any of our markers for user-facing clients. To better understand these clients we aggregate them based on the vendor portion of their MAC addresses. First, we find a set of vendors and query streams that indicate special-purpose devices:

**50 Microsoft devices:** Of the Microsoft devices, 48 query for names within the *xboxlive.com* domain. Only 1 other client in the dataset also queries for a name within *xboxlive.com*. We conclude that the 48 devices are Microsoft Xbox gaming consoles.

**33 Sony devices:** All 33 devices query for names within the *playstation.net* domain, while no other client in the dataset does. We conclude that the 33 devices are Sony Playstation gaming consoles.

**16 Apple devices:** These devices on average issue 11K queries—representing 96% of their lookups—for the *apple.com* domain, even though the average across all devices that lookup an *apple.com* name is 262 queries. Additionally, we observe these devices issue queries for Netflix domains as well. We hypothesize that they are Apple TVs

used for streaming video.

**7 Linksys devices:** All 7 devices predominantly issue queries for *esuds.usatech.com*, a laundry management system. We conclude that these devices are transactions systems attached to the laundry machines in the residence halls (!).

In addition to these, we find devices that we cannot pinpoint explicitly, but do not in fact seem to be general-purpose client systems. We find 41 Dell devices that differ from the larger population of hosts in that they query for more PTR records than A records. A potential explanation is that these devices are servers obtaining hostnames for clients that connect to them (e.g., as part of *sshd*'s verification steps or to log client connects). We also identify 12 Kyocera devices that issue queries for only the campus NTP and SMTP servers. We conclude that these are copy machines that also offer emailing of scanned documents.

For the IP addresses that do not appear in the DHCP logs, we cannot obtain a vendor ID. However, we note that 97% of the queries and 96% of the unique hostnames from these machines involve Case domains and therefore we conclude that they serve some administrative function and are not general purpose clients. The remaining 61 devices are distributed among 42 hardware vendors. In the remainder of the chapter we will consider the general purpose clients (Users) and the special purpose clients (Others) separately, as we detail in Table 5.1.

## 5.4 Query Clusters

Applications often call for multiple DNS queries in rapid succession—e.g., as part of loading all objects on a web page, or prefetching names for links users may click. In this section, we quantify this behavior using the DBSCAN algorithm [EKSX96] to construct clusters of DNS queries that likely share an application event. The DBSCAN algorithm uses two parameters to form clusters: a minimum cluster size $M$ and a distance $\varepsilon$ that controls the addition of samples to a cluster. We use the absolute difference in the query timestamps as

| Dataset | No. of Clusters | Cluster Size (mean) | Queries in Clusters (percentage) |
|---|---|---|---|
| Feb:Residential:Users | 1.0M | 12 | 80% |
| Feb:Office:Users | 7.4M | 13 | 83% |
| Jun:Residential:Users | 220K | 22 | 84% |
| Jun:Office:Users | 6.6M | 13 | 79% |
| Feb:Residential:Others | 170K | 4.3 | 65% |
| Feb:Office:Others | 2.5M | 7.8 | 90% |
| Jun:Residential:Others | 36K | 6.7 | 58% |
| Jun:Office:Others | 2.7M | 6.5 | 77% |

Table 5.3: Details of clustering per dataset.

the distance metric. Our first task is to choose suitable parameters. Our strategy is to start with a range of parameters and determine whether there is a point of convergence. Based on the recommendations in [EKSX96] we start with an $M$ range of 3–6 and an $\varepsilon$ range of 0.5–5 seconds. We find that 96% of the clusters we identify with $M = 6$ are exactly found when $M = 3$ and hence at $M = 3$ we have converged on a reasonably stable answer for clusters greater than or equal to size 6. However, $M = 6$ (and $M = 4$ or 5) excludes clusters as small as size three. Inspection of a sample of the 270K clusters of three lookups when $M = 3$ finds that the queries are frequently for similar but distinct hostnames—all three queries are for the same SLD in 48% of the clusters and 2 of the 3 queries are for the same SLD in 83% of the clusters. This result suggests that the clusters of size 3 are in fact valid clusters and not noise. Therefore, we use $M = 3$ in our analysis since smaller clusters are included and larger clusters are rarely affected by the choice of $M \in [3, 6]$. Additionally, we find that for $\varepsilon \in [2.5, 5]$, the total number of clusters, the distribution of cluster sizes, and the assignment of queries to clusters remain similar irrespective of $\varepsilon$ value and therefore use $\varepsilon = 2.5$ in our analysis. We define the first DNS query per cluster as the *root* and all subsequent queries in the cluster as *dependents*. Table 5.3 shows the breakdown of the number of clusters found by DBSCAN for each dataset and the percentage of the queries in the datasets that belong to clusters. In the Feb:Residential:Users dataset we find 1.0M clusters that encompass 80% of the roughly 15M queries in the dataset.

We examine the clustering algorithm applied to the Feb:Residential:Users dataset. First, we consider the 67K unique hostnames the algorithm labels as noise. The most frequent hostnames in the queries are: WPAD [GCD99] queries for discovering proxies, Google Mail and Google Docs, software update polling (e.g., McAfee and Symantec), heartbeat signals for gaming applications (e.g., Origin, Steam, Blizzard, Riot), video streaming (e.g., Netflix, Youtube, Twitch), and the Network Time Protocol (NTP). All of these names can intuitively come from applications that require only sporadic DNS queries, as they are either making quick checks every once in a while, or are using long-lived sessions that leverage DNS only when starting.

To validate the clusters themselves, we observe that there are frequently occurring roots. The 1M clusters we form have 72K unique roots. The 100 most frequently occurring roots account for 395K (40%) of the clusters. Further, the 100 most popular roots include popular websites (e.g., *www.facebook.com*, *www.google.com*). These are the type of names we would expect to be roots in the context of web browsing. Another common root is *safebrowsing.google.com* [Gooc], a blacklist directory used by some web browsers to determine if a given web site is safe to retrieve. This is a distinctly different type of root than a popular web site because the root is not directly related to the dependents by the page content, but rather via a process running on the clients. This in some sense means SafeBrowsing-based clusters have two roots. While use of SafeBrowsing is fairly common in our dataset, we do not find other prevalent cases of this "two roots" phenomenon. From a modeling standpoint we have not yet determined whether "two roots" clusters would need special treatment.

Figure 5.1 shows the distribution of queries per cluster. While the majority of clusters are small, there are relatively few large clusters. We find that 90% of clusters contain at most 26 queries for at most 22 hostnames. Additionally, we find 90% of the clusters encompass at most 10 SLDs. The largest cluster spans 95 seconds and consists of 9,366 queries for names that match to the $3^{rd}$ level label. The second largest cluster consists of 6,211 queries

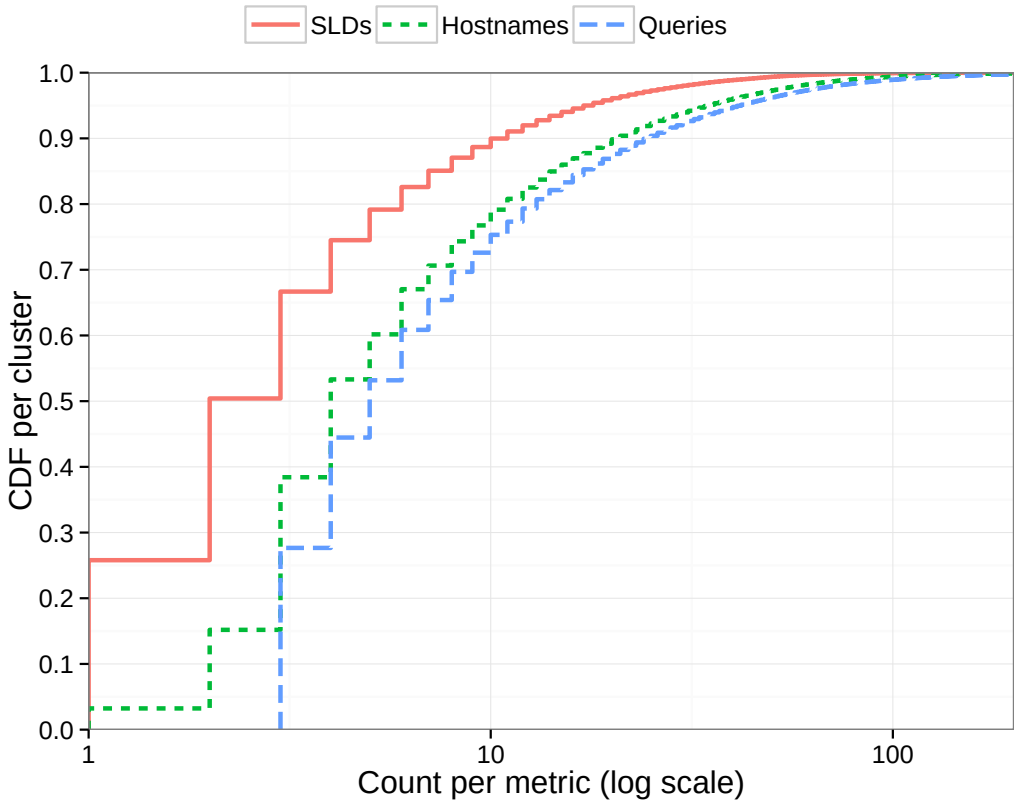


Figure 5.1: Number of queries, hostnames, and SLDs per cluster in the Feb:Residential:Users dataset.

for *myapps.developer.ubuntu.com*—which is likely a bug.

Finally, we examine that percentage of DNS queries occurring in clusters on a per-client basis (Figure 5.2) in the Feb:Residential datasets. For Users devices, we find that 70% or more of DNS queries occur in clusters for 74% of clients. Clustering of DNS queries is therefore common in most of the general purpose devices in our datasets. The same is not true for Others, however, where participation in clustering varies across clients from no participation to full participation. This result is to be expected, since the Others category combines many different types of devices, each with their own behavior. We note that this trend holds across all of our measurements with the Users datasets all demonstrating similar behavior while the Others datasets show variation across location (residential vs. office) and time (February vs. June). We focus on the Feb:Residential datasets for the

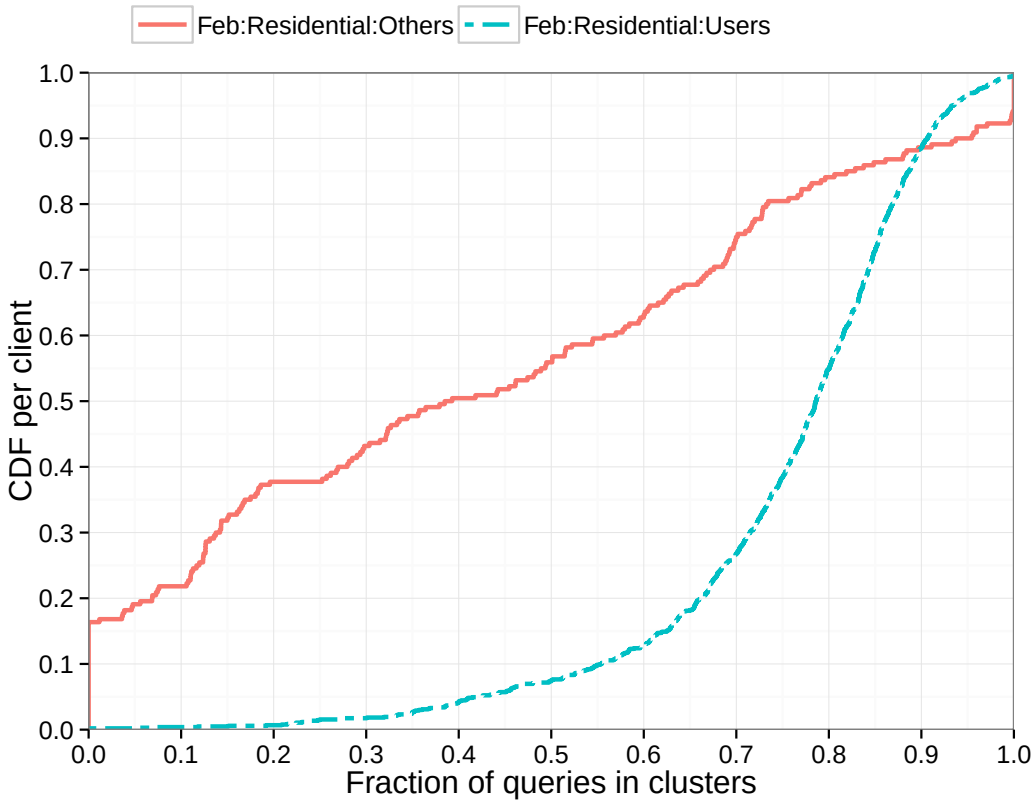


Figure 5.2: Fraction of queries in clusters for each client in the Feb:Residential:Users and Feb:Residential:Others datasets.

remainder of this work.

## 5.5 Query Timing

Next we tackle the question of when and how many queries clients issue. We begin with the distribution of the average number of queries that clients issue per day, as given in Figure 5.3 for the Feb:Residential datasets. We find that Users issue 2K lookups per day at the median and 90% of Users issue less than 6.7K queries per day. The Others show greater variability than Users. Relatively few clients generate more queries than the rest of the clients—i.e., the top 5% of clients produce roughly as many total DNS queries per day as the bottom 95%.

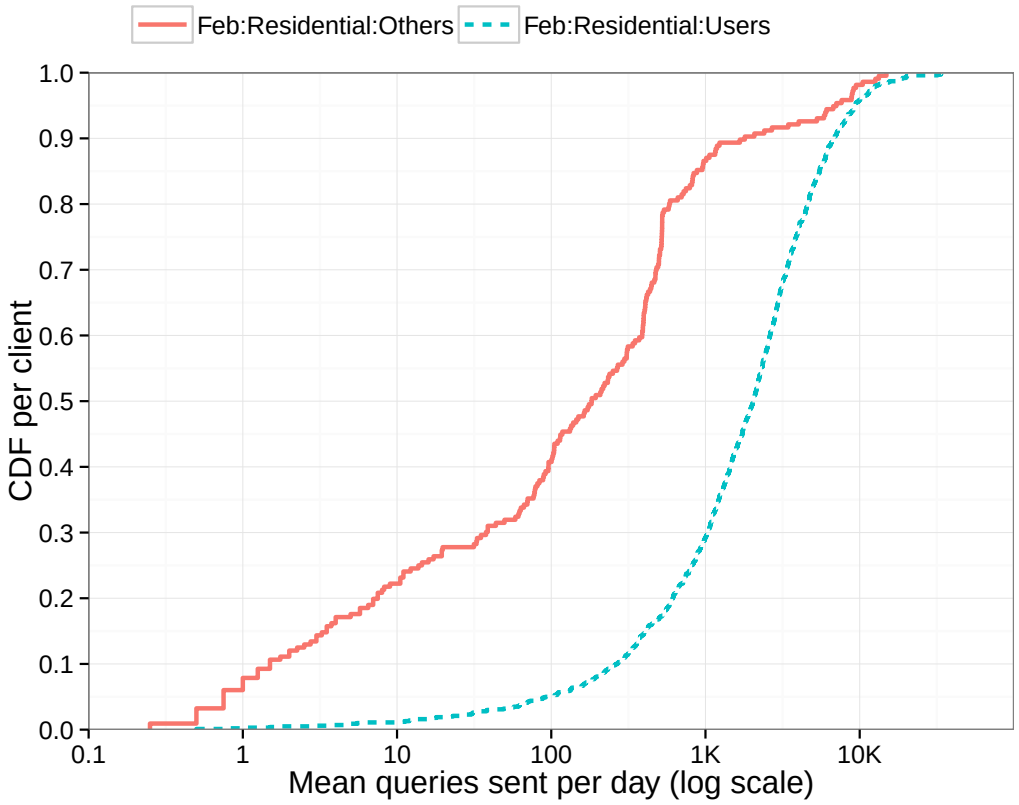


Figure 5.3: Queries issued by each client per day in the Feb:Residential:Others and Feb:Residential:Users datasets.

A related subject is the time between subsequent queries from the same client, or inter-query times. Figure 5.4 shows the distribution of the inter-query times in the Feb:Residential:Users dataset. The "Aggregate" line shows the distribution across all clients. The area "90%" shows the range within which 90% of the individual client inter-query time distributions fall. The majority of inter-query times are short, with 50% of lookups occurring within 34 milliseconds of the previous query. In the tail, 0.1% of inter-query times are over 25 minutes. We conclude that the distribution has a heavy tail because the tail fits well to the Pareto distribution (see below). Intuitively, long inter-query times represent off periods when the client's user is away from the keyboard (e.g., asleep or at class). The Others datasets show wide ranging behavior suggesting that, since they include a variety of device types, they are less amenable to succinct description in an

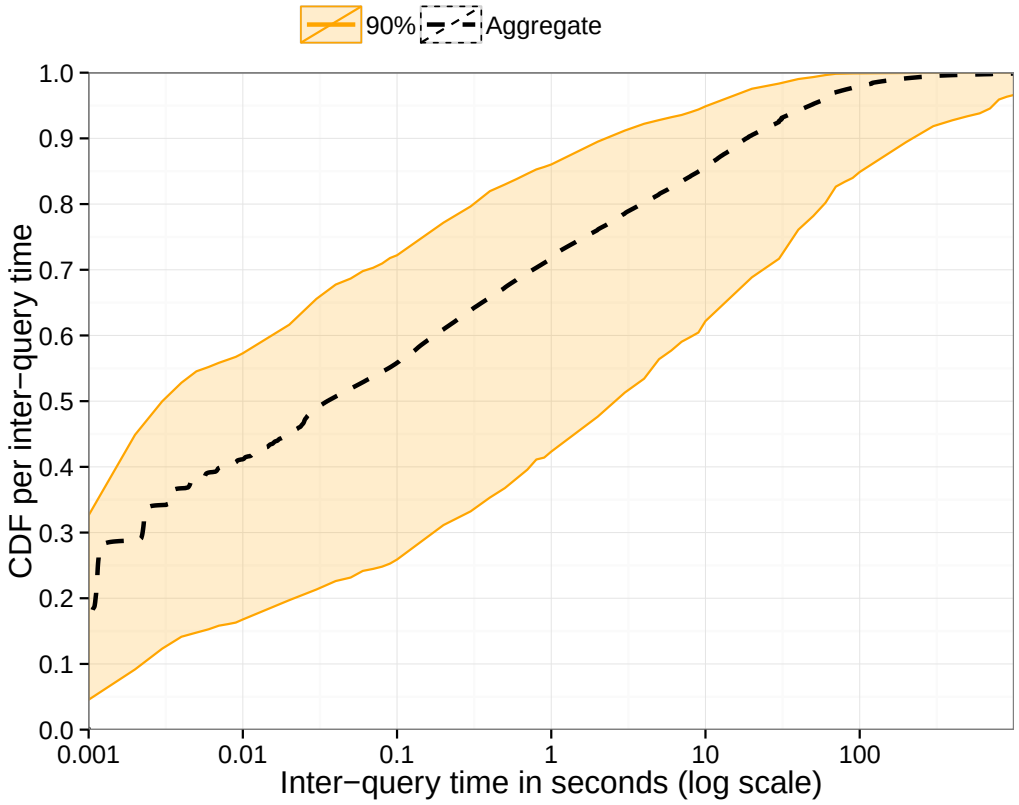


Figure 5.4: Time between queries from the same client in aggregate and per client.

aggregate model.

For Users, we are able to model the aggregate inter-query time distribution using the Weibull distribution for the body and the Pareto distribution for the heavy tail. We find that partitioning the data at an inter-query time of 22 seconds minimizes the mean squared error between the data and the two analytical distributions. Next, we fit the analytical distributions—split at 22 seconds—to each of the individual client inter-query time distributions. We find that while the parameters vary per client, the empirical data is well represented by the analytical models as the mean squared error for 90% of clients is less than $0.0014$. Therefore, DNS clients can likely be modeled well with a distribution with parameters varied per client.

Next, we move from focusing on individual lookups to focusing on timing related to the 1M clusters that encompass 12M (80%) of the queries in our dataset (see § 5.4).

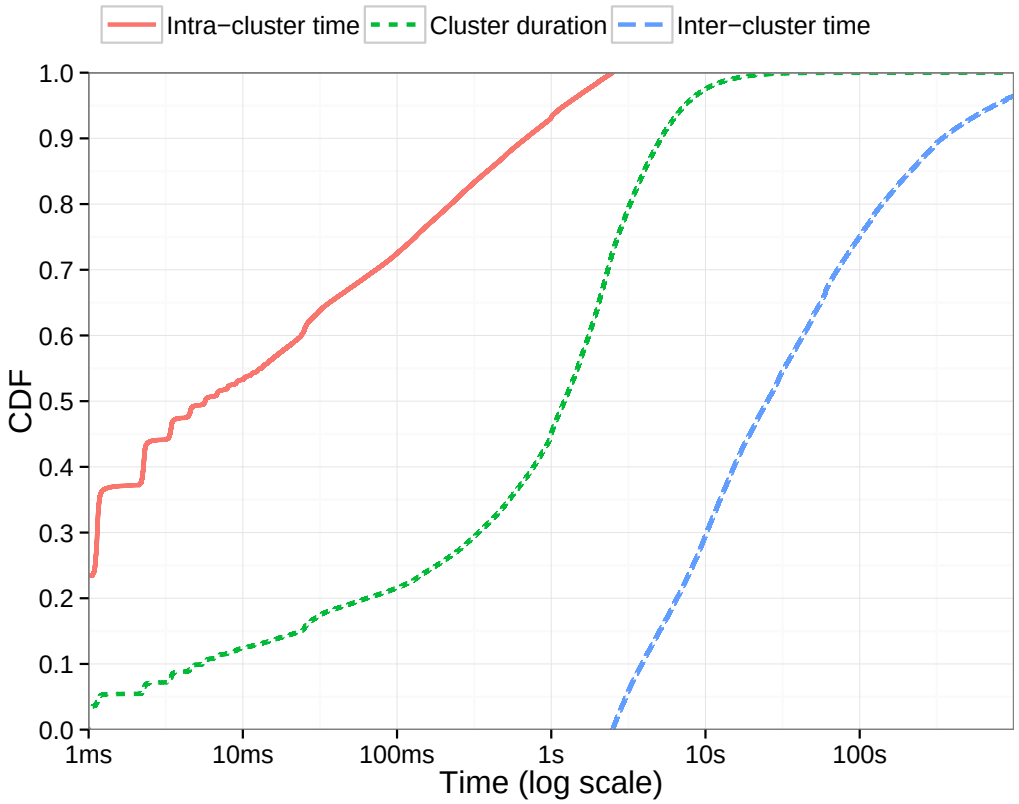


Figure 5.5: Duration of clusters, inter-cluster query time and intra-cluster query time.

Figure 5.5 shows our results. The "Intra-cluster time" line shows the distribution of the time between successive queries within the same cluster and shows that nearly all are far less than the $\varepsilon$ value of 2.5 seconds. On the other hand, the line "Inter-cluster time" shows the time between the last query of a cluster and the first query of the next cluster and grows quickly beyond $\varepsilon$. The line "Cluster duration" shows the time between the first and last query in each cluster. Most clusters are short, with 99% less than 18 seconds. Additionally, we find that most of client DNS traffic occurs in short clusters: 50% of clustered queries belong to clusters with duration less than 4.6 seconds and 90% are in clusters with duration less than 20 seconds. In the Others datasets, a smaller percentage—roughly 60%—of DNS queries occur in clusters.

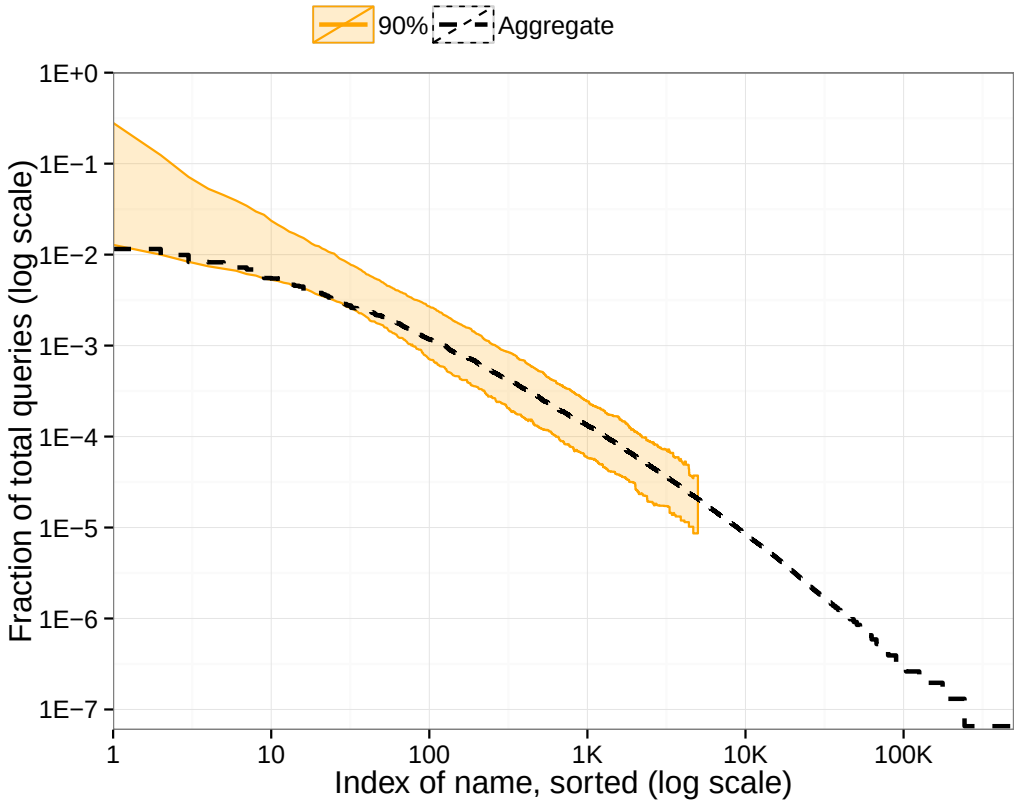


Figure 5.6: Fraction of queries issued for each hostname per client.

## 5.6 Query Targets

Finally, we tackle the queries themselves including relationships between queries.

### 5.6.1 Popularity of Names

We analyze the popularity of hostnames in our dataset using two methods. First, we determine the total number of queries per hostname across the dataset. Figure 5.6 shows the fraction of queries for each hostname (with the hostnames sorted in terms of decreasing popularity). Per § 5.5, we plot the aggregate distribution and a range that encompasses 90% of the individual client distributions. Of the 499K unique hostnames within our dataset, 256K (51%) are looked up only once. Meanwhile, the top 100 hostnames account for 28% of DNS queries. The second way we assess popularity is in terms of how many clients

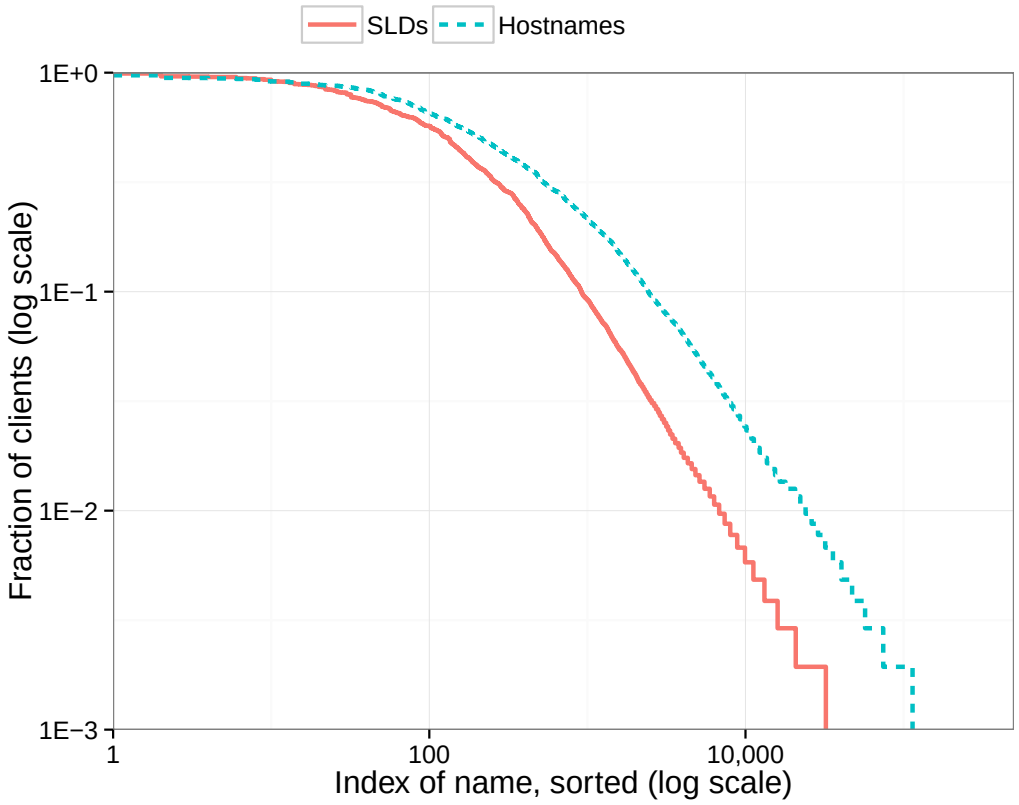


Figure 5.7: Fraction of clients issuing queries for each hostname and SLD.

look up a given name. Figure 5.7 shows the fraction of clients that query for each name. We find that 77% of hostnames are queried by only a single client. However, over 90% of the clients look up the 14 most popular hostnames. We find that 13 of these hostnames are Google services and the remaining one is *www.facebook.com*. The plot shows similar results for second-level domains (SLDs), where 66% of the SLDs are looked up by a single client.

Both the distribution of queries per name and clients per name demonstrate power law behavior in the tail. Interestingly, the Pearson correlation between these two metrics—popularity by queries and popularity by clients—is only $0.54$ indicating that a domain name with many queries is not necessarily queried by a large fraction of the client population and vice versa. As an example, *updatekeepalive.mcafee.com* is the $19^{th}$ most queried hostname but only 8.1% of clients are responsible for these queries, placing the hostname

$3,109^{th}$ in terms of the fraction of clients per name. Similarly, 55% of the clients query for *s2.symcb.com*, placing it $175^{th}$. However, in terms of total queries this hostname ranks as only the $1,215^{th}$ most queried. This phenomenon can be explained by differences in TTL. The record for *s2.symcb.com* has a one hour TTL—explaining why we do not observe more frequent queries. Meanwhile, *updatekeepalive.mcafee.com* has a 1 minute TTL. Given this short TTL and that the name implies polling activity, the large numbers of queries from a given client is unsurprising.

The heavy tails of the popularity distributions represent the majority of DNS transactions. Therefore, while tempting, we cannot disregard unpopular names—not even those queried just once—because their sheer number is large and together they are responsible for most of DNS activity. Therefore they have an impact on the entire DNS ecosystem (e.g., its utilization and cache behavior).

### 5.6.2 Co-occurrence Name Relationships

In addition to understanding popularity, we next aim to assess the relationships between names, as these have implications on how to model client behavior. The crucial relationship between two names that we seek to quantify is frequent querying for the pair together. We begin with the clusters we develop in § 5.4. Further, we leverage the intuition that the first query within a cluster triggers the subsequent queries in the cluster and is therefore the *root* lookup. This intuition follows from the structure of modern web pages, with a container page calling for additional objects from a variety of servers—e.g., [HTT] shows that the average web page leverages objects from 16 different hostnames.

Finding co-occurrence is complicated because of client caching of names. This means that we cannot expect to see the entire set of dependent lookups each time we observe some root lookup. Our methodology for detecting co-occurrence is as follows. First, we define $clusters(r)$ as the number of clusters with $r$ as the root across our dataset and $pairs(r, d)$ as the number of clusters with root $r$ that include dependent $d$. Second, we

limit our analysis to the case when $clusters(r) \geq 10$ to limit the potential for false positive relationships based on too few samples. Across our dataset we find 7.1K (9.9%) of the clusters meet this criteria. Within these clusters we find 7.5M dependent queries and 2.2M unique $(r, d)$ pairs. Third, for each pair $(r, d)$, we compute the fraction of co-occurrence as $pairs(r, d)/clusters(r)$—i.e., the fraction of the clusters with root $r$ that include $d$. Co-occurrence of most pairs is low with 2.0M (93%) pairs having a co-occurrence fraction much less than 0.1. We focus on the 78K pairs that have high co-occurrence fractions—greater than 0.2—and aim to understand them. These pairs include 98% of the roots we identify. Additionally, nearly all roots have at least one dependent with which they co-occur frequently. Also, these pairs comprise 28% of the 7.5M dependent queries we study.

We cannot test the majority of the clusters and pairs for co-occurrence because of limited samples. However, we hypothesize that our results apply to all clusters. We note that the distribution of the number of queries per cluster in Figure 5.1 is similar to the distribution of the number of dependents per root where the co-occurrence fraction is greater than 0.2. Combining our observations that 80% of queries occur in clusters, 28% of the dependent queries within clusters have high co-occurrence with the root, and the average cluster has 1 root and 11 dependents, we estimate that at a minimum $80 * 0.28 * 11/12 = 21\%$ of DNS queries are driven by co-occurrence relationships. We conclude that co-occurrence relationships are common, though the relationships do not always manifest as requests on the wire due to caching.

Finally, we note that intuitively dependent names could be expected to share labels with their roots—e.g., *www.facebook.com* and *star.c10r.facebook.com*—and this could be a further way to assess co-occurrence. However, we find that 27% of the pairs within clusters with co-occurrence of at least 0.2 share the same SLD and 11% share the 3rd level label as the root. This suggests that while not rare, counting on co-occurring names to be from the same zone to build clusters is dubious. As an extreme example, Google Analytics is a dependent of 1,049 unique roots, most of which are not Google names.

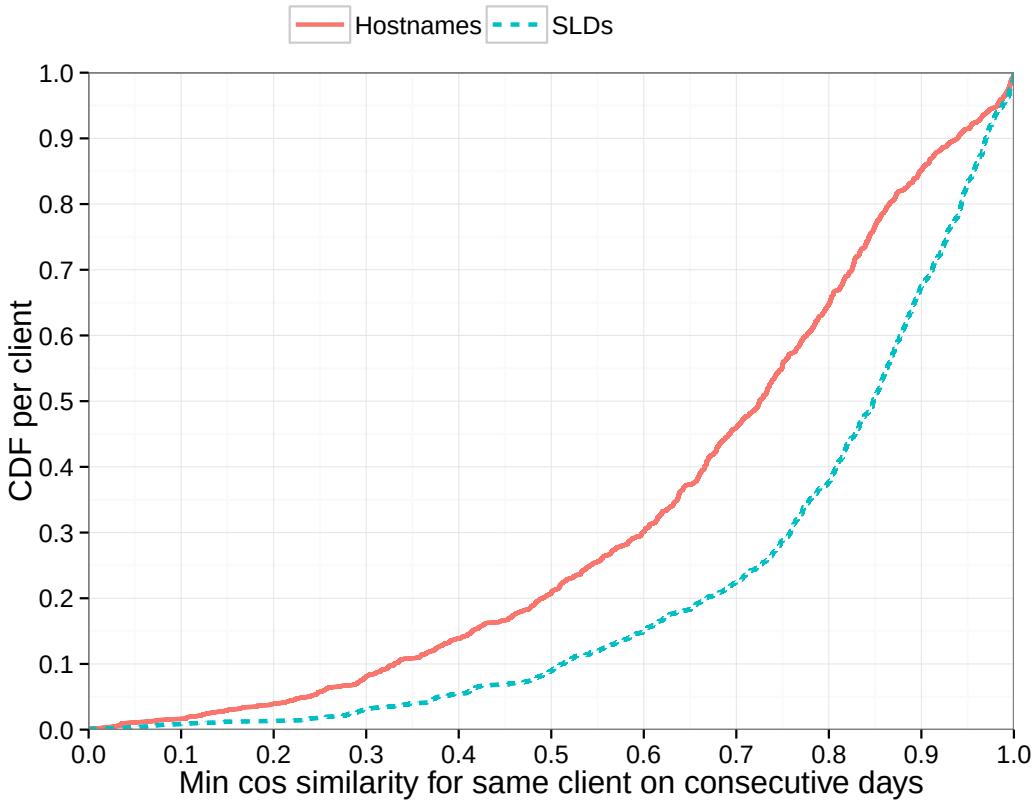


Figure 5.8: Cosine similarity between the query vectors for the same client.

### 5.6.3 Temporal Locality

We next explore how the set of names a client queries changes over time. As a foundation, we construct a vector $V_{c,d}$ for each client $c$ and each day $d$ in our dataset, which represents the fraction of lookups for each name we observe in our dataset. Specifically, we start from an alphabetically ordered list of all hostnames looked up across all clients in our dataset, $N$. We initially set each $V_{c,d}$ to a vector of $|N|$ zeros. We then iterate through $N$ and set the corresponding position in each $V_{c,d}$ as the total number of queries client $c$ issues for the $N_i^{th}$ name on day $d$ divided by the total number of queries $c$ issues on day $d$. Thus, an example $V_{c,d}$ would be $[0, 0.25, 0, 0.5, 0.25]$ in the case where there are five total names in the dataset and on day $d$ the client queries for the second name once, the fourth name twice and the fifth name once. We repeat this process using only the SLDs from each query, as

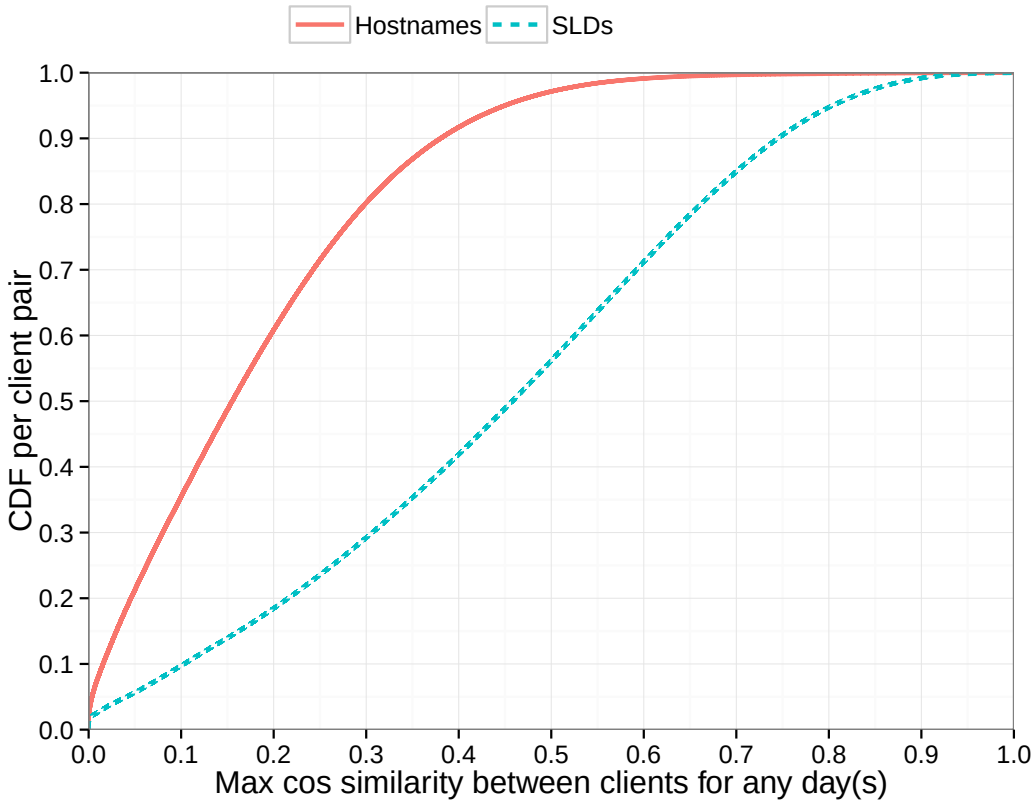


Figure 5.9: Cosine similarity between the query vectors for different clients.

well.

The first question we tackle with these vectors is whether clients' queries tend to remain stable across days in the dataset. For this, we compute the minimum cosine similarity of the query vectors for each client across all pairs of consecutive days. Figure 5.8 shows the distribution of minimum cosine similarity per client. In general, the cosine similarity values are high—greater than $0.5$ for 80% of clients for unique hostnames—indicating that clients query for a similar set of names in similar relative frequencies across days. Given this result, it is unsurprising that the figure also shows high similarity across SLDs, as well.

Next we assess whether different clients query for similar sets of names. We compute the cosine similarity across all pairs of clients and for all days of our dataset. Figure 5.9 shows the distribution of the maximum similarity per client pair from any day. When considering hostnames, we find drastically lower similarity values than when focusing on a

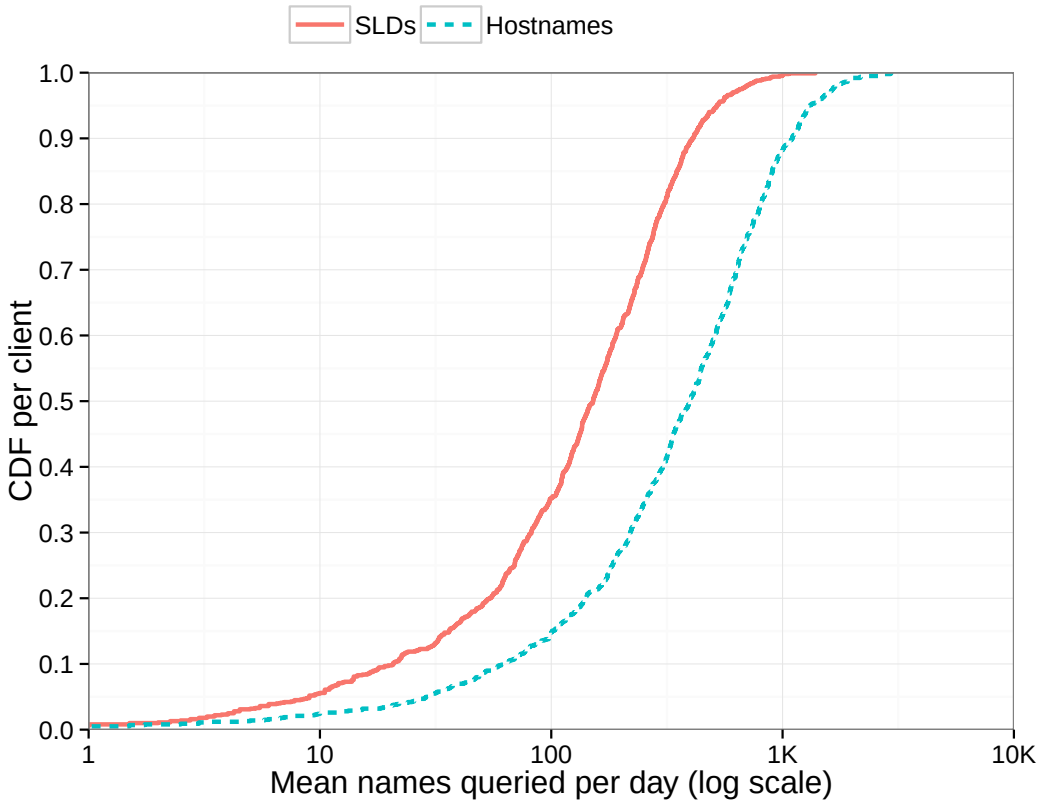


Figure 5.10: Mean hostnames and SLDs queried by each client per day.

single client—with only 3% showing similarity of at least 0.5—showing that each client queries for a fairly distinct set of hostnames. The similarity between clients is also low for sets of SLDs, with 55% of the pairs showing a maximum similarity less than $0.5$. Thus, clients query for different specific hostnames, but also for distinct sets of SLDs. These results show that a client DNS model must ensure that $(i)$ each client tends to stay similar across time and also that $(ii)$ clients must be distinct from one another.

A final aspect we explore is how quickly a client re-queries for a given name. As we show above in Figure 5.3, 50% of the clients send less than 2K queries per day on average. Figure 5.10 shows the distribution of the average number of unique hostnames that clients query per day. We find the number of names is less than the overall number of lookups, indicating the presence of repeat queries. For instance, at the median, a client queries for 400 unique hostnames and 150 SLDs each day. To assess the temporal locality of re-

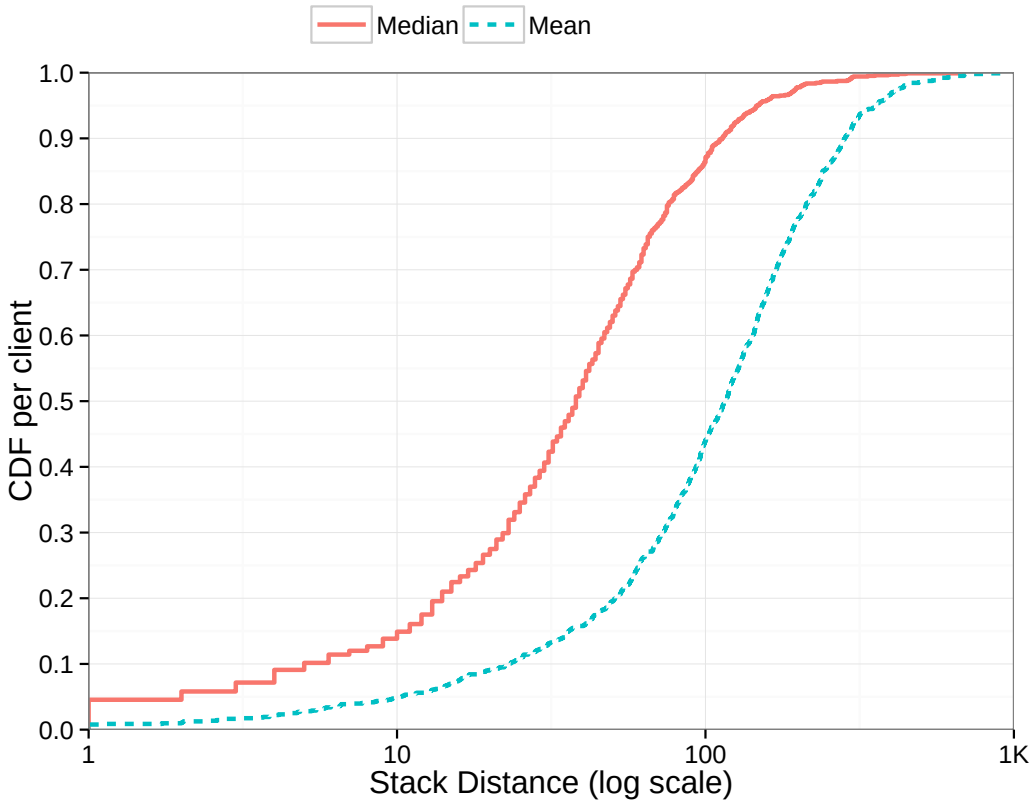


Figure 5.11: Mean and median stack distance for each client.

queries, we compute the stack distance [MGST70] for each query—the number of unique queries since the last query for the given name. Figure 5.11 shows the distributions of the mean and median stack distance per client. We find the stack distance to be relatively short in most cases—with over 85% of the medians being less than 100. However, the longer means show that the re-use rate is not always short. Further, the distributions also show that variation exists among clients, with some clients revisiting names frequently while some clients query a larger set of names with less frequency.

## 5.7 Summary

This work is an initial step towards richly understanding individual DNS client behavior. We characterize client behavior along dimensions that will ultimately inform an analytical

model. We find that types of clients interact with the DNS in distinct ways. Further, DNS queries often occur in clusters of related names, and short clusters account for most DNS traffic. As a first step towards an analytical model, we show that the client query arrival process is modeled well by a combination of the Weibull and Pareto distributions. Additionally, we observe that clients have a "working set" of names that is rather stable over time and fairly distinct from other clients. Finally, we note that we find these high-level insights hold across both time—currently 4 months, but future work will explore a longer timeframe—and qualitatively different user populations—student residential vs. University office. This is an initial indication that the broad properties we illuminate hold the promise to be invariants.

# Chapter 6

# A New Security Vulnerability in the DNS

Because many applications depend upon DNS, the security of DNS has a large impact on the security of the overall network. Replacing an authoritative mapping from hostname to IP address with a fraudulent mapping will divert users to malicious hosts. Once diverted, users may be subject to a variety of follow-up attacks from phishing to malware installation. In this chapter, we uncover a new DNS vulnerability to an attack designed to substitute the authoritative mapping[1] and measure the prevalence of this vulnerability.

Fraudulent hostname-to-IP address mappings originate in two places: ($i$) a component in the hostname resolution machinery (e.g., a local DNS resolver) or ($ii$) a man-in-the-middle that can monitor DNS transactions and either change or inject responses. A variant of the first is a record injection attack whereby an attacker populates the cache of a DNS resolver with an illegitimate record, which the resolver then uses to satisfy subsequent legitimate requests for the given hostname. The attack we uncover is a form of record injection attack. We note that the base DNS protocol offers no protection against the latter form of attack.

Unfortunately, assessing the extent of security threats within the DNS infrastructure is anything but straightforward. As demonstrated in Chapter 3, DNS transactions often take a complex path through a maze of intermediate resolvers. This work attributes the

[1]This work originally appeared as part of [SCRA14].

uncovered vulnerability to a specific actor in this infrastructure—home network routers—and we find that 7–9% of home routers are vulnerable.

## 6.1 Related Work

In 2005, Arends and Koch [AK05] notes the danger of record injection attacks on open DNS resolvers that accept DNS queries from arbitrary sources. The prevalence of such resolvers is increasing—from 15M in 2010 [LL10] to 30M in 2013 as estimated by our own work (§ 3.5.1). While not all open resolvers are vulnerable to known attacks, their increasing numbers provide a larger attack surface that is important to understand.

Many previous studies discuss specific DNS security vulnerabilities. Dagon, et al. [DAD+09] provide a model of DNS record injection and show how implementation considerations weaken the security of resolvers. Ariyapperuma and Mitchell [AM07] provide an overview of many known DNS security vulnerabilities. Schuba [Sch93] notes the lack of authentication in the DNS even though DNS is sometimes used to identify and trust remote machines. Callahan [Cal12] demonstrates that a large percentage of RDNS servers are still vulnerable to Kaminsky [Kam08] style attacks many years after the vulnerability was announced. Additionally, other forms of record injection (e.g., bailiwick rule violations, negative response rewriting) are also still present in the modern DNS.

The Internet engineering community has spent considerable energy fortifying DNS with DNSSEC [AAL+05] which cryptographically protects the integrity of the authoritative bindings set by the holder of a name. While DNSSEC is the long-term security strategy for the DNS, deployment is currently low—with only about 1% of the resolvers validating DNSSEC records [GC11, Fuj12]. Given the low DNSSEC deployment, understanding the security landscape of DNS without DNSSEC remains of critical importance.

Studies have also offered potential solutions to DNS record injection vulnerabilities. Bernstein [Ber08] proposes a method of adding encryption and authentication to DNS using

| Scan | Begin | Duration (Days) | # ODNS | # RDNS |
|---|---|---|---|---|
| $S_7$ | 3/1/13 | 11 | 40.5K | 5.3K |
| $S_8$ | 7/19/13 | 12 | 2.31M | 86.1K |

Table 6.1: Datasets collected for measuring vulnerability to record injection attacks.

elliptic curve cryptography. Antonakakis, et al. [ADL+10] introduce a heuristic method for detecting malicious records in cache. Leveraging the observation that DNS records are generally stable over time, the authors use a combination of whitelisting and statistical methods to determine if an IP address change is malicious or benign. Tzur-David, et al. [TDLDA12] use anomaly detection at the RDNS server in the timing of DNS messages. Responses that arrive too quickly—compared to an estimated round-trip-time between the RDNS server and the ADNS server—are delayed. If another response arrives, then it is highly probable that one of the two responses is malicious and both are dropped. Yuan, et al. [YKMC06] propose a peer-to-peer strategy that enables groups of RDNS servers to detect poisoned records within the group. In this chapter, we present a new vulnerability in a different component of the infrastructure.

## 6.2 Methodology

Our basic methodology for studying the vulnerability of the client-side DNS infrastructure is to probe the Internet in search of ODNS servers, similar to how we collected our previous datasets. See § 3.3 for details of our scanning apparatus. Table 6.1 provides information about the datasets we collect and utilize in the remainder of this chapter. We continue labeling the datasets where we left off in Chapter 3 to avoid confusion.

Note, we return to methodological issues in § 6.6. In particular, we use the techniques we develop in the following sections to address two specific issues. First, we aim to understand whether the ODNS servers we find are actually in operational use by real users. Second, since we do not probe the entire Internet address space, we seek to understand if our sample is representative of the broader Internet.

## 6.3 Record Injection Attacks

The potential for an off-path attacker to poison a DNS cache via record injection is not a new observation. Resolvers rely upon matching the contents of the response to the contents of the request in order to validate responses. Consider an attacker $A$ that wishes to poison the record for "www.foo.com" from the ADNS server $D$ in the cache of an RDNS resolver $R$. A legitimate response to a request from $R$ will have the following attributes. In the IP header, the source IP address is the IP address of $D$ and the destination IP address is the IP address of $R$. In the UDP header, the source port number is the DNS default port of 53 and the destination port number is the source port used by $R$ in the request. Finally, in the UDP datagram, the querystring is "$www.foo.com$" and the transaction ID value matches the value set in the request. Of these, only the destination UDP port number and DNS transaction ID are difficult for the attacker to learn.

To execute a record injection attack, $A$ crafts a malicious DNS response and times sending the response to $R$ so that the response arrives after a query for "$www.foo.com$". If $A$ guesses both the transaction ID and port number correctly *and* squeezes the malicious response in before the legitimate response from $D$ arrives, $A$ poisons $R$'s cache with the malicious response. But, if $A$ guesses incorrectly, the malicious response is discarded and $R$ will cache the valid response from $D$ when it arrives. $A$ must then wait until the cached record is evicted before attempting to poison with a malicious record again.

The attack is made simpler by three insights. First, $A$ does not need to coincide their attack with a query from a client. Instead, $A$ may issue a query to $R$ themselves and immediately follow it with a malicious response. The legitimate response from $D$ will take much longer to return to $R$ than for the malicious response to arrive, so timing becomes a non-issue. Second, $A$ can send multiple responses per attempt, each with different guesses for the transaction ID and port number, and improve $A$'s chances of guessing the correct values. Third, in 2008, Kaminsky [Kam08] demonstrated how $A$ need not wait for cached records to be evicted between poison attempts, thus dramatically reducing the time needed

for a record injection attack to succeed.

To protect resolvers against record injection attacks, resolvers should make it as hard as possible for the attacker to guess the correct values. The recommended way is by selecting the transaction ID and port number at random [Kam08]. Randomization of either value alone is insufficient to effectively protect against Kaminsky attacks [Kam08].

## 6.4 Preplay Attack

While Kaminsky attacks require an attacker to forge an acceptable DNS response, we determine that FDNS servers are vulnerable to a previously unknown injection attack. FDNS servers do not themselves recursively look up mappings, but they often do have caches of previous lookups. The FDNS servers populate these caches with the responses from upstream RDNS resolvers. In some cases we find that FDNS servers fail to validate the DNS responses. This leaves these FDNS servers vulnerable to the crudest form of cache injection: a "preplay" attack whereby an attacker sends a request to a victim FDNS server and then, before the legitimate response comes back, the attacker answers the request with a fraudulent response. The FDNS server will then forward the fake response to the originator and cache the result. An FDNS server that ($i$) forwards requests with both a new random ephemeral port number and a new random DNS transaction ID and ($ii$) verifies these and the upstream RDNS server's IP address on returning responses is protected against the preplay attack. Such protections would reduce an attacker to guessing a variety of values in the short amount of time before the legitimate response from the RDNS arrives. However, we find a non-trivial number of FDNS servers simply forward on the packets received and/or do not verify the values on DNS responses. This leaves the door open for a crude attack whereby an attacker does not have to guess these values, but can just use those from the original request.

To assess the extent of this vulnerability during our $S_7$ and $S_8$ experiments, we

send a request for a hostname from our domain to each ODNS and immediately issue a fraudulent response containing IP address $X$. On the other hand, our ADNS responds to these requests with a binding to IP address $Y$. The probing host issues a subsequent request and determines which IP address is in the ODNS' cache.

In its most primitive form, the preplay attack does not involve spoofing or guessing to make the fraudulent response appear legitimate—we use the ephemeral port number and DNS transaction ID from the original request. Additionally, we use the probing machine's genuine IP address which is clearly not the IP address of RDNS server that an FDNS server should expect. In addition to the primitive form, we also consider variants of the attack that involve spoofing the source IP address and manipulating the destination port number in the DNS response. To learn and use the correct source IP address in the fraudulent response, an attacker could perform the following steps:

1. The attacker first sends the victim FDNS server $V$ a DNS request for a hostname within a domain controlled by the attacker. The requested name embeds $V$'s IP address. When this request arrives at the attacker's ADNS, the attacker observes the IP address of an RDNS $R$ used by $V$.

2. The attacker sends a second DNS request to $V$, now for the victim domain, and immediately follows up with a fraudulent response that appears to originate from $R$. The method of constructing the fraudulent response will be discussed later.

We note that the above method of determining the IP address of the RDNS server used by an FDNS server may not work for RDNS pools and definitely does not work for hierarchical client-side DNS systems, e.g., Google or any other DNS resolution service where the ingress DNS server is not the same as the egress DNS server that the attacker observes. However, we find in § 3.5.3 that over 30% of FDNS servers likely do not use hierarchical client-side DNS systems as they only query a single RDNS server.

| | Variant | Tested | Vulnerable | Overlap with Default |
|---|---|---|---|---|
| $S_7$ | Default | 41K | 3.5K (8.6%) | 100% |
| | Ephemeral | 41K | 3.3K (8.2%) | 98% |
| | Random | 41K | 3.4K (8.3%) | 98% |
| | Spoof-Default | 41K | 2.8K (6.8%) | 98% |
| | Spoof-Ephemeral | 41K | 3.2K (7.8%) | 97% |
| | Spoof-Random | 41K | 3.3K (8.1%) | 98% |
| $S_8$ | Default | 2.3M | 170K (7.3%) | N/A |

Table 6.2: FDNS vulnerability to preplay cache injection in the $S_7$ and $S_8$ datasets.

Next, we examine if an attacker can increase the attack success rate by manipulating the destination port number of the DNS response. the FDNS servers will expect the response to come to the port number used in the request and reject other responses. We experimentally test two possibilities for setting the destination port number of the attack response to match the port selection of the FDNS server. First, we observe that the source port number on DNS responses from some FDNS servers received by our probing hosts is not always port 53 even though we send our probes to port 53. We hypothesize that the port upon which the FDNS communicates with our probing host is the same ephemeral port upon which the FDNS communicates with its RDNS server and therefore send the DNS response for the preplay attack to this port number. Second, it is possible that some FDNS servers ignore port numbers in deciding whether to accept a DNS response. Therefore, we attempt to perform the preplay attack using a randomly selected port number. In total, the above described variants make six possible combinations: non-spoofed source IP address and default port 53 (Default), non-spoofed source IP address and ephemeral port (Ephemeral), non-spoofed source IP address and random port (Random), spoofed RDNS IP address and default port 53 (Spoof-Default), spoofed RDNS IP address and ephemeral port (Spoof-Ephemeral), and spoofed RDNS IP address and random port (Spoof-Random).

We first test the preplay attack during the $S_7$ scan. Unlike our previous scans which were all performed from PlanetLab nodes, the $S_7$ scan leverages a single machine in a residential network.[2] For each ODNS server, we attempt each attack variant three times to

[2]Due to some of our tests using spoofed addresses which is against PlanetLab's acceptable use policy.

reduce any impact from packet loss. Of the roughly 41K ODNS servers we test, we find 2.8 to 3.5K (or 6.8 to 8.6%) to be vulnerable to the preplay attack variants. While all variants without spoofing have similar success rates, our success rate with spoofing is lower—likely indicating filtering of this traffic.

We conclude that ODNS servers are failing to take three simple measures to thwart this attack: ($i$) use a new and random DNS transaction ID, ($ii$) verify that the source IP address in DNS responses matches the IP address of the upstream RDNS, and ($iii$) verify the destination port number on responses. The latter is particularly intriguing as it suggests these devices are not running a traditional protocol stack in which packets arriving on an unbound port number are dropped. Given we find no increase in the success rate with our attempts at spoofing, we return to PlanetLab with the $S_8$ scan to assess the vulnerability at a larger scale. Of the 2.3M ODNS servers we test, we find 170K (or 7.3%) to be vulnerable to the preplay attack. Our previous measurements of the DNS infrastructure in § 3.5.1 indicate that there are between 31M—33M ODNS servers on the Internet. Thus, we estimate that there are 2.2M preplay vulnerable ODNS servers on the Internet.

## 6.5 Implications

### 6.5.1 Duration of Record Injection

The injection attack we discuss above can only be successful when part of the DNS infrastructure caches a fraudulent record and then returns that record in response to a normal user request. In our assessment of the caches of FDNS servers and RDNS resolvers (§ 4), we find ($i$) little evidence of cache evictions based on capacity limits and ($ii$) that records with long TTLs—which can be set in injected records—stay in the cache for at least one day in 60% of the RDNS$_i$ resolvers and 50% of the FDNS servers. This shows the impact of record injection can be long-lived.

### 6.5.2 Phantom DNS Records

A class of denial-of-service attacks relies on placing a large DNS record in a cache (at an RDNS, say) and then spoofing requests that will cause the record to be sent to some victim. This can both hide the actual origin of the attack, as well as amplify (in volume) an attacker's traffic by using records that are larger than requests. To date this requires attackers to register a domain and serve large records to insert them into the various caches or find an ADNS that is already serving large records. However, using record injection techniques, an attacker does not need to be bound to any centralized infrastructure. In fact, any hostname—real or not!—could be readily inserted in the cache and then used in a subsequent attack. This leaves less of a paper trail that can potentially trace back to an attacker. The preplay attack allows such record injection into millions of devices with trivial effort.

## 6.6 Context

We now return to contextual issues surrounding our measurements, as sketched in § 6.2.

### 6.6.1 Are Open Resolvers Used?

We first turn to the question of whether ODNS servers in fact serve users or are active, yet unused artifacts. This bears directly on whether the preplay attack represents a real problem. First, in companion work we use several criteria—including scraping any present HTTP content on the ODNS, consulting blacklists of residential hosts and observing DNS protocol behavior—to determine that "78% [of ODNS servers] are likely residential networking devices" [SCRA13]. Using the same criteria against the FDNS servers in the $S_8$ scan, we find that 91% of the FDNS servers that are vulnerable to the preplay attack are likely residential network devices. While this result does not speak directly to use, our experience is that these devices act as DNS forwarders for devices within homes and therefore

we believe this suggests actual use.

Additionally, we seek to test directly for evidence that the FDNS servers we probe are in use by some client population. We start by gathering round-trip time (RTT) samples for each FDNS and the corresponding RDNS. For the FDNS we use the preplay attack to measure the RTT by taking the time between sending a fraudulent response to the FDNS and receiving the response back from the FDNS at our client. Measuring the RTT to the RDNS is more complicated. The process starts with the client requesting some name $N$ from our ADNS. The ADNS responds with some CNAME $N'$, which the RDNS then resolves and our ADNS returns a random address $S$. The mapping between $N$ and $S$ then returns to the FDNS and ultimately our client. The client then issues a request for $N'$—which will presumably be in the RDNS' cache, but given the primitive nature of preplay-vulnerable FDNS devices not in the FDNS' cache since the RDNS resolves the CNAME on the FDNS's behalf. The response for $N'$ will be $S$ when the RDNS answers the request from the cache. It is possible that there are cases when the RDNS server returns the CNAME to the FDNS server, the FDNS server caches the CNAME, and the FDNS answers the request for $N'$ from cache. In this case, we will underestimate the RDNS RTT. We address this issue below.

After we obtain RTTs for both FDNS and RDNS, we seek to understand whether popular web site names are in the FDNS cache as a proxy for whether the FDNS is in use by a population of users. We therefore issue DNS requests for the Alexa[3] top 1,000 web sites and time the responses.[4] Given unreliable TTL reporting by FDNS servers as we previously observed in § 4.3.2, we determine that a given hostname is in the FDNS cache using the time required to resolve the name. Since we expect individual FDNS to have diurnal variation, we perform the lookups on each FDNS every 4 hours for one day. Our own queries will populate the FDNS cache and therefore we must exercise care with

[3] www.alexa.com

[4] Note, we augment the list of sites by prepending each web site name from Alexa with "www"—which is not included in the list—and we therefore probe for 2,000 names.

subsequent probes lest we wrongly conclude users employ the FDNS when it is our own probes we observe. We mitigate this issue in two ways. First, we probe all names with an authoritative TTL of 4 hours or more only once, accounting for 415 names. Second, we inject records into FDNS cache from our ADNS with the same TTLs as the remaining 1,585 records in our corpus (all of which are less than 4 hours). At each 4 hour interval we check our own records and if the FDNS incorrectly returns a record that had an initial TTL of $x$ we exclude all but the initial query for popular names with an initial TTL of at least $x$.

We determine that a given hostname is in the FDNS cache if the time required to resolve the name during our $S_8$ scan does not exceed the median FDNS RTT. Figure 6.1 shows the distribution of the fraction of FDNS servers that hold a given number of records in their cache. The "All" line shows the distribution for all preplay-vulnerable FDNS servers. We find 81% of the FDNS servers have at least one popular name in their cache at some point during the experiment. However, the distribution also shows that over 30% of the FDNS servers have at least 100 hostnames in the cache. It seems unlikely that an FDNS with few clients behind it would have a large number of records in cache. Instead, we believe these represent cases where our heuristic is not properly delineating between the FDNS and RDNS cache. Therefore, in an effort to better delineate the FDNS and RDNS caches we plot the subset of FDNS servers where the maximum FDNS RTT is at least 10 msec less than the minimum RDNS RTT, which we denote as "Far from RDNS". This subset encompasses 8.4K FDNS servers and we do see the tail of the distribution fall away. Within this subset, 53% of the FDNS servers are in use. Additionally, we examine the subset of FDNS servers that are accessible for our entire 24 hour experiment. In this subset, we find more in-use FDNS servers—90% of all FDNS servers and 68% of FDNS servers that are far from their corresponding RDNS resolver.

Note, our heuristic provides a lower bound on the number of in-use FDNS servers since we only measure a fraction of the 24 hour period. Indeed, the median TTL for the popular names is 10 minutes. Assuming the median TTL, an FDNS that enforces the TTL

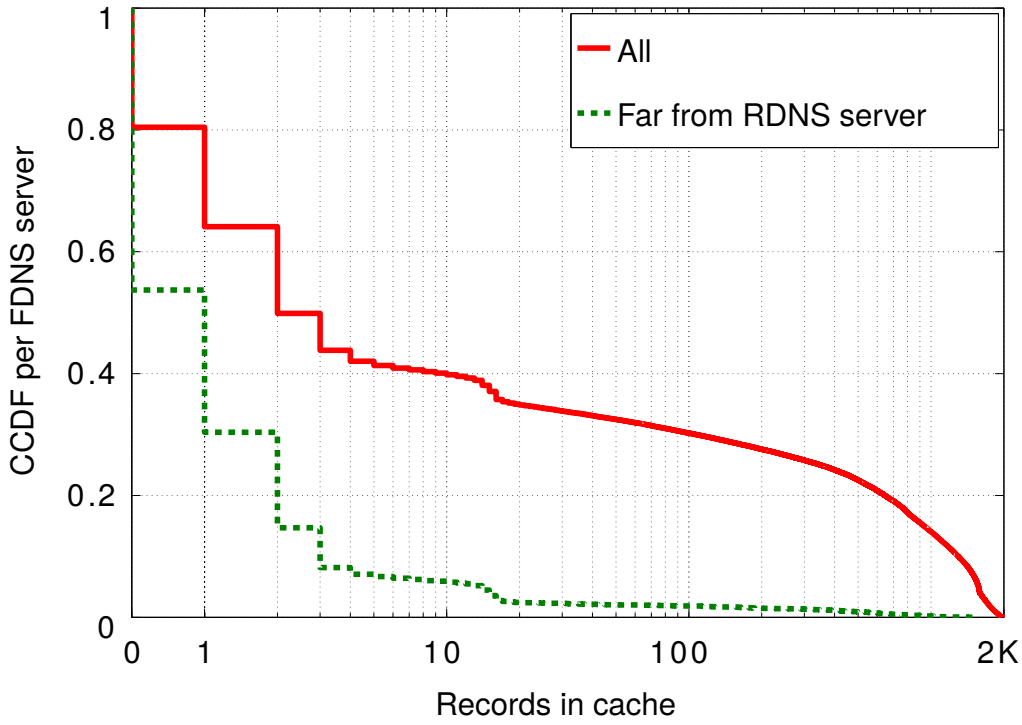


Figure 6.1: Distribution of popular websites in FDNS server caches.

and an FDNS available for 24 hours, our strategy provides a one-hour window into the FDNS' cache—or, just over 4% of the day. Further, our extensive probing of the FDNS' cache may actually overflow the cache thus pushing out records added via use. Therefore, we believe that many of the FDNS servers that do not show as in use are in fact in use, but that short TTLs and our coarse and extensive probing conspire to hide the use. Our general conclusion is that the FDNS servers we find are in fact in use by people during their normal browsing.

### 6.6.2 Industry Response

Many residential networking devices have Web interfaces for management. As part of the $S_8$ scan, we probe the vulnerable ODNS servers on TCP port 80 in search of identifying information. For 4.5K (2.6%) of the vulnerable FDNS, we are able to identify the manufacturer from either names or model numbers in the HTTP response headers or

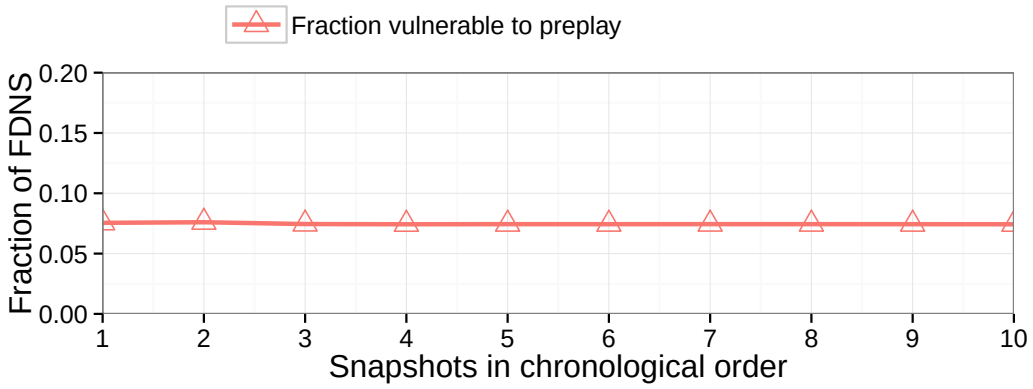


Figure 6.2: Vulnerability frequency at snapshots during discovery.

HTML body. The identified manufacturers include: Asus, Belkin, Buffalo, Cisco, D-Link, Huawei, Linksys, Netcomm, Netgear, TP-Link, Yamaha, and Zyxel. Many simplistic home networking devices rely upon common open source software—[Cal12] found that 24% of ODNS servers run the RomPager embedded web server—and we speculate that vulnerability across manufacturers indicates common software heritage. In an effort to disclose the vulnerability, we alerted the United States Computer Emergency Readiness Team (US-CERT) [UC] which contacted the manufacturers directly. To date, we are unaware of any steps being taken to mitigate the preplay vulnerability.

We speculate that the Preplay vulnerability is the result of a shortcut taken for simplicity in some consumer networking devices. Many of these simplistic devices are deployed without methods to remotely update software, leaving vendors powerless to fix vulnerabilities in the devices. Therefore, we note the importance of remote update functionality even in simplistic pieces of networking hardware.

### 6.6.3 Representativeness

Finally, we return to the issue of representativeness of our results as mentioned in § 6.2. Since our scans do not encompass the entire Internet our insights could be skewed by our scanning methodology. To check this we divide our scans into ten chronological slices and

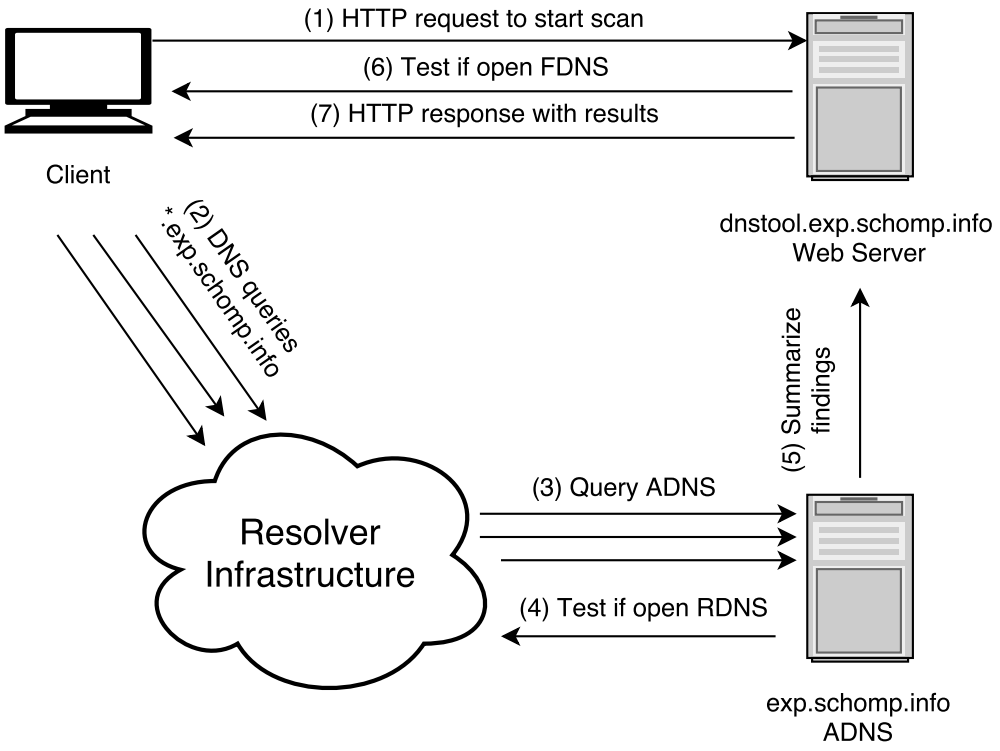


Figure 6.3: Schematic of the DNS Vulnerability Scanner Webtool.

derive the cumulative vulnerability rate at each slice for the preplay attack. The slices are equal in size in terms of the number of vulnerable FDNS servers. The cumulative vulnerability rate should plateau once the dataset is typical of the broader population. Figure 6.2 shows the cumulative vulnerability rate across the ten slices. The FDNS vulnerability rate reaches steady state immediately, illustrating that we are in fact capturing a representative sample of FDNS servers with random sampling of IP addresses—which is not surprising.

## 6.7 Vulnerability Scanner

The DNS vulnerability that we uncover in this work has the potential to impact users directly. We built a Web-based tool that helps users learn about how their DNS resolutions are being handled [Sca] directly from their Web browser. Specifically, the tool enables users to

learn about the servers that participate in resolving names on their behalf, whether any of those servers expose the user to record injection vulnerabilities, and how long resolutions take.

Figure 6.3 illustrates the functioning of the tool. It is Web-based, so the user first navigates to a landing page using their Web browser. There is no software to install.

1. After navigating to *http://dnstool.exp.schomp.info/*, the user begins a measurement by pressing a button on the Webpage, causing an HTTP request to our server.

2. Next, through Javascript, the Web browser is instructed to repeatedly fetch objects from the server in the form *http://random_string.dnstool.exp.schomp.info/small_object/*. Javascript does not expose DNS method calls, yet, because the domain names are unique, each object fetch will cause a DNS request. The object fetches repeat for the duration of the test.

3. The DNS requests will ultimately arrive at our authoritative DNS server, where we can check the queries for adoption of mitigation techniques against the Kaminsky vulnerability (See § 6.3). Our ADNS server responds to the queries with a negative answer to prevent subsequent HTTP requests since they serve no further purpose.

4. For each resolver IP address contacting our ADNS server, we issue DNS queries to test if the resolver is open.

5. A summary of the resolvers observed, analysis of their mitigation deployment, and status as an open resolver is passed from our ADNS server to our Web server.

6. Concurrently with this process, the Web server tests for open FDNS server in front of the user. In typical configurations, the IP address of the client is also the IP address of their home WiFi router. We issue DNS queries to the source of the HTTP request to test for an open FDNS server. If one is found, we next proceed to test for Preplay vulnerability (See § 6.4).

7. Finally, the results of the scan are passed back to the client for display to the user.

## 6.8 Summary

In this study, we assess the susceptibility of the client-side DNS infrastructure to a new record injection attack. Through active probing, we uncover and measure a new attack vector—the preplay attack. We find 7–9% of the open DNS resolvers are vulnerable to the preplay attack. The vulnerable resolvers are typically residential routers and inspection of their caches suggest that they are in use. We develop a tool to enable users to learn about how their DNS resolutions are being handled and discover whether the resolvers in use are vulnerable to known record injection attacks.

# Chapter 7

# DNS Shared Resolvers Considered Harmful

DNS recursive resolvers abstract the multi-step iterative DNS resolution procedure from clients and provide a shared cache across clients—thus offering the possibility of better performance to clients and scalability to DNS itself. While DNS resolvers follow the classic architectural approach of modularity, we question whether this factorization is still useful in the modern Internet, or whether we should eliminate resolvers and instead have the stub resolvers on clients perform their own recursive resolutions[1]. Eliminating DNS resolvers promises a number of benefits.

First, removing resolvers simplifies the overall system. As sketched in Chapter 3, modern DNS resolvers constitute a complex infrastructure with many distinct components, making the system difficult to manage and troubleshoot. Pushing the functionality to clients makes the clients themselves more complex, but in a way that is easier to manage and more transparent than our current nebulous situation where a resolution is handled by an unknown number of hidden actors within the complex client-side infrastructure.

Second, shared DNS resolvers handicap the operation of replicated services, notably

[1]This work originally appeared in [SAR14].

content delivery networks (CDNs)—which carry 39–55% of Internet traffic [GD11]. Since a DNS request generally precedes content requests, CDNs often use the origin of the DNS request as a hint about the location of the client. However, in § 3.5.4, we note that clients and their DNS resolvers may in fact be far apart and therefore the replica chosen based on the DNS resolver's IP address will offer suboptimal performance. This result is supported by several other studies [STA01, MCD+02, HMLG11, ARS13]. Further, public DNS resolvers such as Google DNS also complicate replica selection by serving arbitrary hosts from all over the Internet and hence obscuring any hint a CDN may be able to take from the source address of the resolver. Our proposal of simply removing shared DNS resolvers directly tackles this issue—without additional mechanisms—by exposing the client's IP address to the authoritative DNS servers that direct clients to specific replicas.

Third, DNS resolvers are vulnerable to multiple forms of attack [Kam08, SCRA14, HS13] such as fraudulent record injection, which expose end users to critical security threats. Further, attackers can launder requests through open resolvers that will answer arbitrary queries from arbitrary clients. We find in § 3.5.1 that the number of open resolvers has increased to over 30M. These open resolvers can be used by an attacker to hide their tracks (e.g., as part of a wider DoS campaign) or circumvent firewalls to expose closed portions of the resolution ecosystem—i.e., that only answer queries for "internal" hosts—to indirect attacks. Also, in § 6.4 we demonstrate a brand new cache poisoning attack against FDNS servers. By removing DNS resolvers, we ($i$) eliminate the threat of resolver cache poisoning attacks on clients conducting their own resolutions and ($ii$) we reduce the overall attack surface of the DNS ecosystem as shared resolvers gradually disappear.

In addition, we note that clients may independently choose to resolve hostnames themselves without changes anywhere else in the system—there are no barriers to transition to our approach. While this does not eliminate the security issues surrounding shared resolvers, it makes them moot for clients that have chosen to conduct their own lookups. We discuss transitioning to our approach in § 7.5.3.

In the remainder of this chapter, we empirically establish that in terms of performance and scalability the benefits of shared DNS resolvers are at best modest. Through trace-driven simulation, we show that direct client resolution provides similar performance to the end user when compared to using a shared resolver. Further, we show that the overall load increase on the rest of the system is modest. While these are not the only two issues to tackle when considering the removal of DNS resolvers—we briefly sketch others in § 7.5—we believe these are the two largest initial questions to consider. We believe our investigation shows eliding DNS resolvers to be a promising approach for strengthening the overall name resolution process.

## 7.1 Related Work

There are many approaches to strengthening the DNS against attack. Perhaps the most significant is DNSSEC [AAL+05], which is a general approach that strives to tackle security issues not by point solutions that aim to fix parts of the infrastructure, but by cryptographically securing the information in DNS transactions. Currently DNSSEC deployment is low—with only roughly 1% of resolvers validating DNSSEC records [GC11, Fuj12] despite DNSSEC's original specification being published in 2005. Our approach is orthogonal to DNSSEC.

In addition to security concerns, shared resolvers pose challenges to CDNs. In particular, CDNs frequently assume DNS resolvers and clients are close, which turns out to be wrong in some cases [ARS13, MCD+02, STA01, QMZ+07, Cal12, SCRA13]. Several proposals develop ways to convey clients' network location to CDNs within DNS requests [OSRB12, HBL12, CvdGLK15]. Contavalli, et al. [CvdGLK15] suggest the inclusion of a new DNS extension to allow RDNS servers to pass the subnet of the client to ADNS servers. CDNs may base redirections off of the client's subnet instead of the RDNS server's IP address. Otto, et al. [OSRB12] propose having clients repeat the last step of the resolution

process directly to obtain a CDN redirection based upon the client's location. Huang, et al. [HBL12] develop a solution wherein clients extend the hostnames they look up by prepending a network location identifier onto the hostname. CDNs that support this technique may then extract the network location from the hostname and provide a CDN redirection based upon the network location. Our proposal simply provides network location information to CDNs as the source address of the DNS request's that clients make.

## 7.2 Datasets and Methodology

We leverage three datasets in our study. Our first dataset is a 4 month long set of traffic logs from the Case Connection Zone (CCZ) [Cas]—a fiber-to-the-home network connecting roughly 100 residences to the Internet with 1 Gbps fiber. Our logs are collected using Bro [Pax99]. For each DNS transaction, we record the request and response and corresponding timestamps. For each TCP connection, we record a summary that includes the initiation time, duration, IP addresses, port numbers, bytes transferred and some ancillary information. We collect data between April 1 and July 31, 2012 from a vantage point between the houses and ISP's network—which also places the monitor between the houses and the ISP's two shared DNS resolvers—as illustrated in Figure 7.1. This vantage point has two implications: $(i)$ we cannot observe which device within a house is responsible for specific traffic as the residences are NAT'ed and previous work shows multiple devices per house exist in our network [SSDA12], and $(ii)$ we cannot observe the traffic between the ISP's shared resolvers and the authoritative DNS servers (ADNS). The delay between our vantage point and both the users' end hosts and the shared resolvers is typically less than 1 msec.

Our passive monitor records 58.8M DNS resolutions. Of these, we find 41M—475K unique domain names—to have valid DNS requests and responses in the trace. We exclude 17.8M transactions for one of three basic reasons: $(i)$ Bro glitches that cause bad

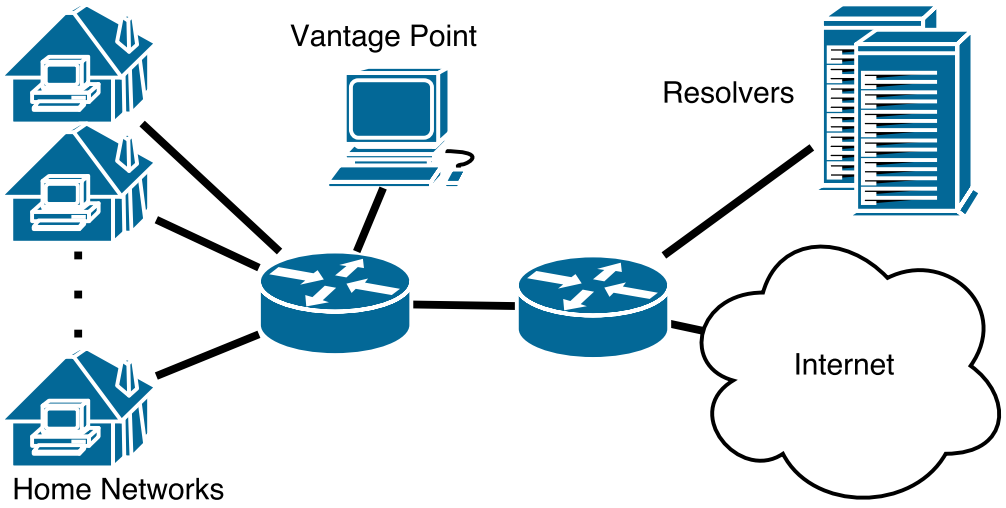


Figure 7.1: Vantage point for our data collection.

timestamping[2] (180K), $(ii)$ no response to DNS queries (8M), $(iii)$ requests with no valid question (90K) and $(iv)$ transactions with responses that have no resource records nor any indication of an error (9.5M). We additionally link DNS transactions with subsequent TCP connections[3]: we link a connection with the nearest preceding DNS query from the same IP address whose response includes the remote IP address used in the TCP connection. We find TCP traffic that leverages 20.4M (or 50%) of the DNS resolutions. Our dataset includes 242M TCP connections and we use the filtering techniques described in [SSDA12] to remove the invalid connections such as those never completing the handshake. Table 7.1 shows the breakdown of the remaining 108M valid TCP connections. We find that 39% do not use a remote IP address found in a previous DNS response (e.g., BitTorrent connections). This leaves 66.3M (61%) TCP connections that leverage the DNS. We further divide TCP connections into those that represent the first use of a DNS response and those that come after a previous use of the DNS response, largely coming from the local DNS cache without requiring additional lookups.

[2]We find this to be a general Bro issue not triggered by DNS traffic. The bad timestamps do not seem to disproportionately impact a certain kind of DNS transaction and therefore we believe there is no systematic measurement bias.

[3]We focus on TCP traffic because less than 0.1% of UDP traffic in the CCZ uses IP addresses from DNS responses.

| | Count | Percent |
|---|---|---|
| Total | 108M | 100% |
| Do not use DNS | 41.7M | 39% |
| Use DNS | 66.3M | 61% |
| First use of DNS response | 20.4M | 19% |
| Remainder | 45.8M | 42% |

Table 7.1: Breakdown of TCP connections in the trace by how they use DNS.

As a baseline, Figure 7.2 shows the distribution of lookup duration found in our logs on the "DNS resolution time" line. The step at less than 1 msec represents names in the shared recursive resolver's cache, while the step at 10 msec is due to responses from nearby ADNS servers. Significantly, we find that hosts do not always create TCP connections immediately after DNS responses. The "Delay before use" line in the figure shows the distribution of the time between a DNS response and the initiation of the first TCP connection based on that response. Note, 20.6M DNS resolutions do not trigger subsequent TCP activity and therefore show in the distribution as having infinite delay. We suspect these unused and delayed-use resolutions largely indicate DNS prefetching, which is common in modern browsers [STI$^+$, Ros]. We find a TCP connection using a DNS response within 50 msec in 36% of the cases. Often, the end host waits significantly longer to use the DNS response than it actually takes to obtain the response. Observing that hosts do not immediately use DNS responses is significant because this indicates there is slack in the process which may allow for longer DNS transactions without impacting the connections that depend on the results.

While we cannot observe the shared resolver's iterative resolution process from our vantage point, we need the timing information about the process to drive our simulations. Therefore, we collect a second dataset by using *dig* to iteratively resolve the names from our passive data collection and record the durations of all iterative steps of the lookup process from a machine within the Case Connection Zone. We perform each iterative step five times and use the average transaction time in our simulations.

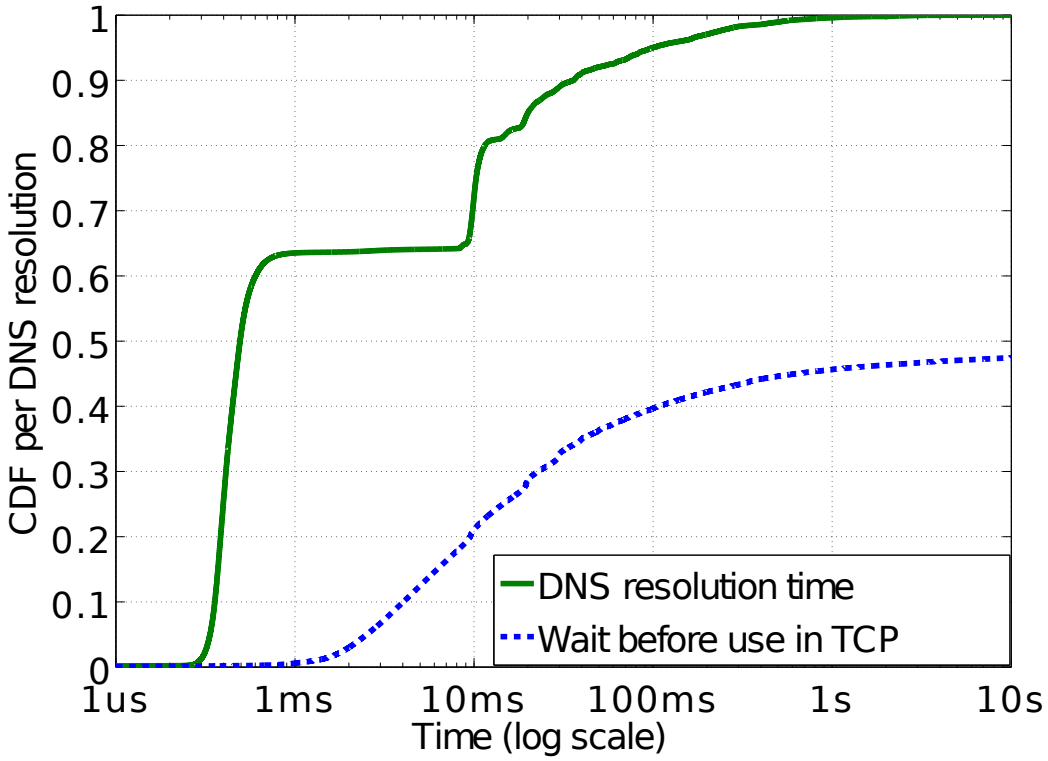


Figure 7.2: Distribution of DNS transaction time and time between DNS response and TCP connection.

We lookup the 475K unique domain names in our trace in two steps. Between November 26 and December 12, 2013 we resolved the 197K names that we find in subsequent TCP traffic. Of these, 5K could not be resolved either due to ADNS server misconfiguration or because the name no longer existed. The 192K unique names we successfully resolve account for 99% of the 20.4M used resolutions in our traces. Further, the successful lookups also cover 98% of the 66.3M TCP connections that utilize DNS in our traces. We conduct a second set of active probes for the unused names on April 10, 2014. This second round of probing was conducted at much higher rate than the first, which caused queuing delays and hence we consider the timing information to be inaccurate. However, we never use the timing information from these lookups as they represent names without subsequent TCP connections. Rather, our objective in this probing is to obtain the time-to-live (TTL) of each record as this will impact our assessment of load (§ 7.4). In total, we have probe data for 459K unique domain names, covering 97% of the resolutions in our trace.

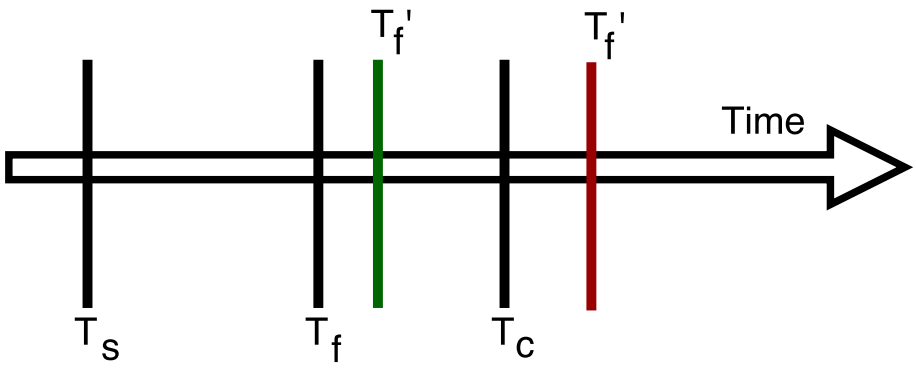


Figure 7.3: Timeline of a DNS resolution from the trace with two possible simulated resolutions superimposed.

Given the dataset and the iterative resolution data, we conduct trace-driven simulations of end-host resolutions that use the natural traffic load we observe in our dataset, as well as the timing and TTL information from the active probing to simulate the needed steps of the iterative process for each lookup. We derive the following variables from our traces (shown in Figure 7.3): $T_s$ is the time we observe a given DNS request that starts a resolution, $T_f$ is the time we observe the corresponding response that finishes a resolution and $T_c$ is the time we observe a TCP connection using the given DNS response to initiate a connection.[4] Further, when simulating DNS resolutions from the client, we use the DNS transaction start times from the traces, but the DNS responses will come back at $T'_f$—which depends on the state of the simulated client's cache and the timing of the required iterative DNS transactions. Thus, while in the traces $T_s < T_f < T_c$ holds, in our simulations, $T'_f$ can fall at any point after $T_s$. We assume that processing time between a DNS response and subsequent TCP connection is negligible. Therefore, when $T'_f <= T_c$, our simulated DNS transaction does not interfere with follow-on TCP activity. However, when $T'_f > T_c$ our simulated DNS transaction actively impedes follow-on TCP activity, which would otherwise be ready to proceed at $T_c$, but would be forced to wait until $T'_f$ to

[4]Given that multiple TCP connections can leverage a single DNS lookup, $T_c$ is actually a set of values and we perform our computations on each value. However, for ease of exposition we often discuss it as a single value.

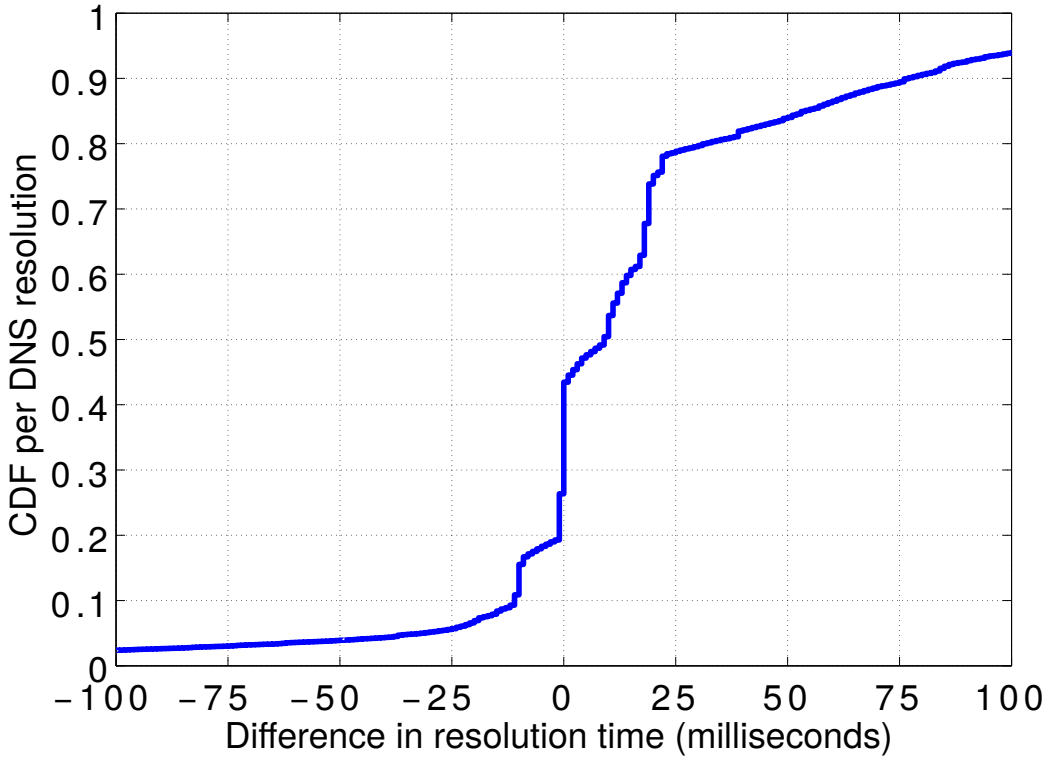


Figure 7.4: Distribution of difference between simulated client resolution and resolution in the trace with a shared resolver.

commence.

Finally, during one week of the passive monitoring of the network we sketch above, we collect TCP SYN/FIN/RST packet traces (June 11-17, 2012). We use *p0f* [Zal] to determine a signature for each TCP connection in our corpus which contains inferences about the hosts based on operating system fingerprint, MSS and IP TTL. Since each home in the network is NAT'ed these signatures allow us to gain some visibility into per-device activity within the house. While not perfect—as two like systems will have the same signature—we find 294 unique signatures across the 100 residences during the week of our dataset. Unfortunately we cannot fingerprint UDP to then correlate the DNS transactions with specific TCP connections. We therefore assign DNS transactions the signature of the closest TCP connection that leverages the binding in the DNS response. We consider this the worst case since it is the soonest the binding will be needed.

## 7.3 Impact on Performance

We now turn to examining the impact of removing shared resolvers on TCP connections. Largely the impact manifests as changes in the duration of the resolution process which is a prerequisite for TCP connections. As a baseline we plot the distributions of the difference between the actual (via a shared resolver) and simulated (directly by clients) resolution times for each DNS resolution in our 4 month trace in Figure 7.4. A positive value indicates that the simulated resolution took longer than the natural duration (i.e., $T'_f > T_f$). The simulated direct client resolutions take less time for 19% of the resolutions, roughly the same amount of time in 26% of resolutions, and more time in 55% of resolutions. The figure shows that direct client resolution adds no more than 50 msec to the resolution process in 84% of the cases.

Also, relying on client resolution can impose delay on a TCP connection only if the DNS response comes after TCP is otherwise ready to begin (i.e., $T'_f > T_c$). First, 39% of the 108M TCP connections in our trace do not require a DNS lookup. No matter what changes are made to the DNS resolution process, these connections cannot be delayed.

Next, we concentrate on the 61% of TCP connections that do utilize DNS responses. These connections pose a problem because they may come from multiple devices within a residence. Since we cannot distinguish between the devices in our full 4 month dataset we conflate multiple devices' caches together. To cope with our suboptimal vantage point we first derive bounds for the impact experienced by these connections. Recall that the impact from the removal of a shared DNS resolver is ameliorated by two factors: $(i)$ the delayed use of DNS resolutions and $(ii)$ device-level DNS caches. We first assume an optimistic case where all traffic within the residence involves a single device with a single cache. The distribution of the added delay imposed on the TCP connections is shown in the "Unified home caches" line in Figure 7.5. In this simulation, only 12% of TCP connections using DNS experience an added delay under direct client resolution, and 4% experience 50 ms or more of delay. Second, we assume no client DNS caching at all and present the distribution

of added time for each TCP connection requiring a DNS lookup on the "No cache" line. The line shows that nearly 60% of TCP connections using DNS are not impacted because their DNS resolutions complete before the use despite any added delay direct client lookup may impose. Taking Figure 7.5 together with our finding that 39% of TCP connections do not rely on DNS, indicates that 75–93% of all TCP connections will feel no impact from direct client DNS resolution.

While these bounds show the impact of direct client lookup is modest at most, we aim to more accurately determine where the performance may fall. We leverage the *p0f* signatures that annotate one week of our data (see § 7.2) to develop a refined—even if not fully accurate—view of device-level caches and re-run our simulation for the given week. The distribution of the amount of time direct client lookup adds to TCP connections is given by the "p0f caches" line on Figure 7.5. The results are similar to those under the assumption of one unified cache for the entire house, which shows that reality is likely closer to the lower-cost bound.

## 7.4 Impact on Scalability

In addition to performance issues, our proposal for eliminating shared resolvers also has potential scalability issues. By caching records, shared resolvers shield authoritative servers from the full workload imposed by clients. Further, by performing iterative lookups on clients' behalf, the resolvers relieve the end hosts from the need to perform these steps. However, under our proposal we force each client to individually consult the authoritative infrastructure and hence we increase the work for the network, the clients and the authoritative servers.

In terms of network load, we find DNS to be less than 0.1% of the total traffic volume of the CCZ network regardless of whether clients rely on a shared resolver or directly resolve names themselves. This indicates that network load is not a concern regardless of

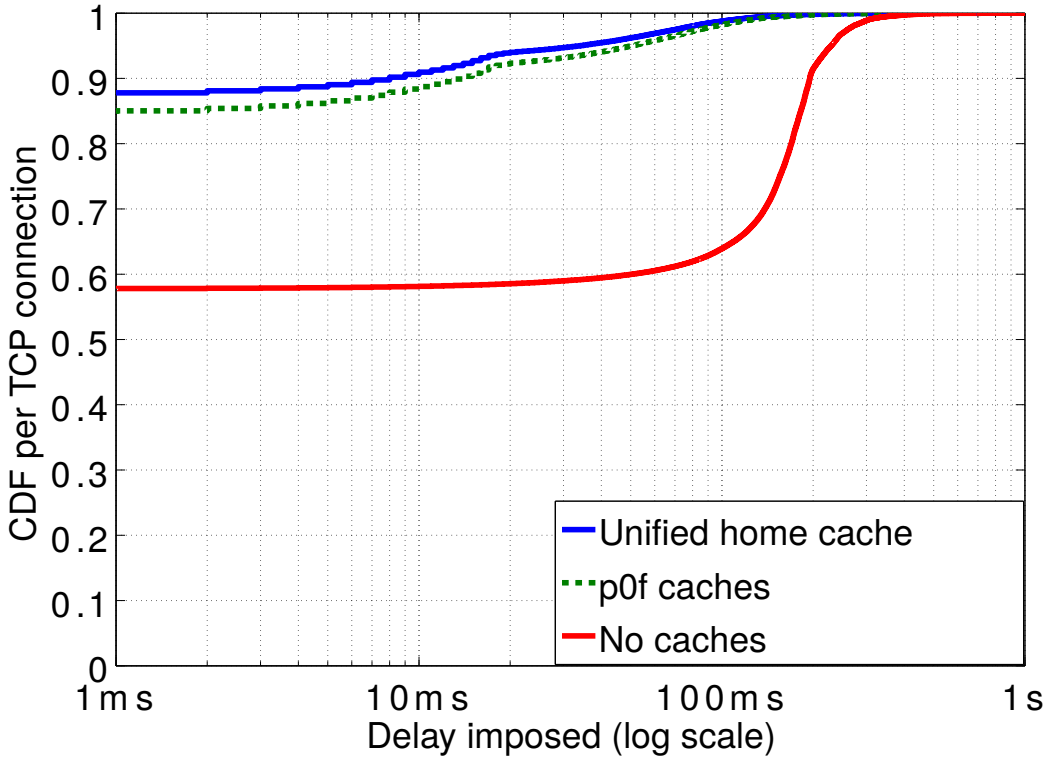


Figure 7.5: Delay added by direct client resolution to the TCP connections that require DNS.

approach taken since DNS remains a small component of network traffic.

Next, we divide our four month dataset into ten seconds bins for each host within the network and record the number of DNS lookups for each bin. With a shared resolver, we find the per bin average, $99.9^{th}$ percentile and peak to be 0.267, 80, and 438 transactions, respectively. With direct client resolutions, these numbers increase by 12%, 20% and 53%, respectively, but the absolute levels remain manageable for personal computers—e.g. the $99.9^{th}$ percentile remains under 100 transactions, or 10 transactions per second. We therefore conclude that neither network nor client load present a barrier to our approach.

We next assess the load increase on the authoritative DNS infrastructure. We simulate both shared resolver and direct client resolution behavior using the workload in our dataset. We use relative change in load as an approximation of the global load increase that authoritative domains may experience under direct client resolution. First, we find that roughly 93% of the authoritative domains do not experience an increase in the average load

(over our ten second windows) while over 99% of authoritative domains show no increase in the peak load. This is due to sparse use of these domains—across both device and time—and therefore low likelihood of benefiting from a shared cache. However, the domains that do experience an increase in load are already popular domains with high loads, which we may exacerbate. The "google.com" second-level domain (SLD) and the ".com" top-level domain (TLD) are the most popular SLD and TLD in our dataset and would experience average load increases of 2.6 and 3.4 times, respectively, with client resolution. Given the popularity of these zones we may have expected an increase on the order of the number of houses we monitor. The increases are two orders of magnitude less, indicating that clients' caches are crucial to dampening demand.

In addition, we note that whether or not to use a shared resolver is a decision made by clients and edge networks and not the authoritative domains. Clients could organically choose to directly contact ADNS servers without involving a shared resolver. Thus, ADNS servers cannot avoid dealing with the additional load such decisions would yield.[5] We are not interested in setting up a situation where clients and domains are at odds and therefore next investigate two ways to mitigate the additional load stemming from client resolution.

Domains like "google.com" can directly manage the load increase by dynamically tuning the TTL of their records to trade load for flexibility. Additionally, it is logical to assume that for each DNS resolution, an organization like Google expects to handle additional follow on traffic (e.g., an HTTP request) which in most cases will be significantly larger than the DNS resolution. For these reasons, we conclude the load increase on SLDs to be manageable. The TLDs, however, have less flexibility over the TTL of their records given that these records affect another party (their client SLDs). We thus focus on the ".com" TLD, which is the most popular TLD with 54% of all transactions across all TLDs in our dataset. The first two rows of Table 7.2 provide a baseline for the average, $99.9^{th}$

[5]While domains could take steps to incentivize use of shared resolvers through various load management techniques (e.g., preferentially dropping incoming requests not from a known shared resolver), these techniques would make accessing the domains more brittle and run contrary to the goal of most domain operators to make their domains as accessible as possible.

| Simulation | Load (DNS transactions / 10 secs) | | |
|---|---|---|---|
| | Average | $99.9^{th}$ | Peak |
| Shared resolver | 0.54 | 17 | 127 |
| Client resolution | 1.84 | 36 | 145 |
| 1 week TTL | 1.15 | 28 | 131 |
| 2 week TTL | 0.89 | 25 | 131 |
| 3 week TTL | 0.77 | 23 | 131 |
| 2 questions | 0.87 | 33 | 135 |
| 5 questions | 0.72 | 30 | 135 |
| 7 questions | 0.68 | 29 | 135 |
| 10 questions | 0.64 | 29 | 135 |
| 1 week TTL & 2 questions | 0.72 | 31 | 135 |
| 1 week TTL & 10 questions | 0.58 | 28 | 135 |
| 2 week TTL & 2 questions | 0.64 | 29 | 135 |
| 2 week TTL & 10 questions | 0.56 | 28 | 135 |
| 3 week TTL & 2 questions | 0.62 | 29 | 135 |
| 3 week TTL &10 questions | 0.55 | 28 | 135 |
| 5-day per-p0f signature caches | | | |
| Shared resolver | 0.32 | 4 | 30 |
| Client resolution | 1.60 | 14 | 62 |

Table 7.2: The load on the ".com" TLD for various mitigation methods.

percentile, and peak load that the ".com" TLD experiences per ten second bin when using a shared resolver and client resolution. With client resolution, the load increases by factors of 3.4, 2.1 and 1.1 at the average, $99.9^{th}$ percentile and peak, respectively. Below we consider two potential load mitigation techniques: $(i)$ a static increase in the records' TTL by ADNS servers and $(ii)$ opportunistic use of extra DNS questions by the clients.

### 7.4.1 Increase TTLs

The first way for domains to shed load is to increase TTLs such that clients cache their records longer. This has been previously proposed as a way to improve availability of SLD mappings [PO12]. The cost of increasing the TTL is reduced flexibility in changing the name-to-address bindings. To see the significance of this issue, we actively request all SLD delegation records from all TLDs we observe in our dataset and find 82% have TTLs of exactly two days. Furthermore, more than 99% of resolutions in our dataset are

for records under SLDs that have TTLs of two days. At the same time, we find that SLD delegations do not change often. We actively resolve these records every day for 67 days and find that an average of 1.1% change within a one week time interval and there is linear growth over longer time periods (e.g., 2.3% and 3.4% change in a two and three weeks time interval, respectively).

To understand how increasing TTLs would impact the load on the TLD servers we vary the TTL of the ".com" delegation records in our simulation. The second group of results in Table 7.2 shows the impact of TTLs from 1–3 weeks. As expected, the load drops as the TTL increases. Still, while the peak load falls to within 3% of the peak load when using a shared resolver, the average load is twice the current load for the one-week TTL and 43% more for the three-week TTL. While we find that over 96% of these records do not change within three weeks, we believe it is unlikely that the community will deem a TTL of three weeks practical due to the lack of flexibility when changes are in fact needed.

Our dataset contains resolutions for 226 distinct TLDs, but many of them handle only a few DNS transactions. In the TLDs that do experience high absolute loads, we find similar benefits to the ".com" TLD from increasing the SLD delegation record TTLs.

One consequence of increasing SLD delegation record TTLs is reduced flexibility. When migrating SLD ADNS servers, the SLD operators must continue to handle DNS requests at the old location until the TTL expires. Previously, the migration period was two days, but with our suggestion it increases to 1 or 3 weeks—increasing the expense of migration.

### 7.4.2 Multiple DNS Questions

A second method to reduce the load on authoritative servers is for clients to piggyback DNS prefetching questions on naturally occurring lookups. In current usage, all DNS transactions involve asking one question. However, the DNS protocol supports multiple questions per request. By opportunistically appending questions for SLD records that the client is

likely to use, the client can populate its cache and avoid a later specific query for the piggybacked record. For instance, if the client suffers a cache miss for the "google.com" delegation record and also notices that its cached copy of "amazon.com" will expire soon, the client could ask for both records in a single request (which is required anyway). This technique could potentially reduce the number of DNS transactions arriving at the TLDs, but not the total number of questions. In fact, the total number of questions could increase, as well, since clients may opportunistically request records that are never subsequently used.

To explore opportunistic prefetching of DNS records we simulate clients that track client accesses for each SLD. For each DNS resolution, we increment a counter for the corresponding SLD and at the end of each day all counters are halved to decay historical popularity.[6] When making a necessary DNS request to a TLD, the client adds questions to the request for the most popular SLD delegation records that are either not in the cache or are close to expiring. We explore requests with 2–10 questions in our simulations because answers for 10 questions generally fit within a 1500B packet.

The third group of results in Table 7.2 shows the load client resolution places on the ".com" TLD server with 2–10 questions per DNS transaction. Including a second question in each request decreases the average load to less than half that of as-needed client resolution. The savings at the $99.9^{th}$ percentile and peak are more modest at 9% and 7%, respectively. Increasing the number of questions to 10 cuts the load of direct resolutions by almost two-third and yields the average load in transactions within 20% of using a shared resolver. For 10 questions per request, the ".com" TLD server must process 12 times the number of questions as compared a shared resolver with a single question per request.

One issue with prefetching is that—unlike everything else we propose—leveraging multiple questions per DNS transaction will require changes to authoritative servers. While posing multiple questions is consistent with the DNS protocol and hence no specification

[6]This is a simple algorithm that could be refined in many ways but suffices to get an initial understanding of the efficacy of the general technique.

changes are needed, we find that authoritative servers generally ignore all but the first question in a request. Further, answering multiple questions per DNS request naturally will increase the processing cost of completely answering the request. An attacker could leverage this feature to coax a busy TLD server into becoming even busier—and ultimately overloading the server to the point of impacting normal requests. TLD servers can mitigate this in a number of ways, including prioritizing resources to clients making only a small number of requests [VS12] and declining to answer multiple questions per query when the load is high.

### 7.4.3 Combining Methods

Finally, the two mitigations we investigate above are not mutually exclusive and therefore we next study the efficacy of both extending the TTL and opportunistically prefetching delegation records, as we show in the fourth group of results in Table 7.2. The first combination involves setting the TTL to one week and using two questions per DNS request. In this case we increase the average load by one-third, the $99.9^{th}$ percentile load by 82% and the peak load by 6% compared with using a shared cache. On the other end of our parameter space, a three-week TTL with ten questions per transaction produces an average load that is nearly the same as the load with a shared resolver. However, the $99.9^{th}$ percentile and peak load show a 65% and 6% increase, respectively. Note that extra questions can actually increase peak rates due to changing the timing of the query flow and sending unnecessary requests.

While the two mitigations we explore help lower the costs for client resolution on the authoritative servers, they do not eliminate it for realistic parameter settings. Additional capacity will be needed by popular domains if clients decide to switch to direct DNS resolution. Further, no parameter setting appears to be a "sweet spot" that provides the optimal benefit to all points on the load distribution.

## 7.5 Additional Considerations

We now discuss additional considerations which we cannot directly quantify, but are part of the tradeoffs of eliding shared resolution infrastructure and moving to direct client resolution.

### 7.5.1 Privacy Concerns

Direct client resolution can reduce users' privacy. When using a shared resolver, clients gain a measure of anonymity as outside their edge network lookups cannot be directly attributed to a specific client. Therefore, a downside to removing the shared resolver is the loss of this measure of privacy. This may be viewed by some users as too revealing to eavesdroppers or simply ADNS servers that log individuals' activities. On the other hand, as we discuss in § 7.1, the ability to directly locate clients is useful for CDNs. As a comment on this tussle we note that many users are willing to use open shared resolvers (e.g., Google DNS [Goob]) and are therefore comfortable with directly attributable DNS requests arriving at a large third-party network. Further, we note that the DNS extension to specify a client's subnet in queries [CvdGLK15] also reduces the client's ability to hide behind a shared resolver. Google DNS already supports the extension [Wan] and, thus, client privacy may already be reduced.

### 7.5.2 Policy Issues

Additionally, shared resolvers also allow network operators to implement edge network policy (e.g., not allowing resolution of some site a company does not wish employees to use while working). Using our approach of direct client resolution removes the DNS resolver as a control point in the network. However, our proposal does not preclude the use of a shared resolver in such cases. We simply view this as akin to web downloads where the expectation is that clients and web servers directly communicate, but in some cases a

proxy is placed in the path to implement policy.

Currently, some networks block DNS traffic (e.g., networks that use a captive portal to authenticate users or require users to read an Acceptable Use Policy). In these networks, DNS traffic not destined for the specific IP addresses of the network's recursive resolvers is blocked, presenting a barrier to the use of client resolution. However, we note that ISPs generally do not fall into this category of network and therefore devices in residential settings are not exposed to this barrier. Also, we again note that our proposal does not preclude the use of shared resolvers in special purpose networks.

### 7.5.3 Transitioning to Client Resolution

As previously discussed, direct client resolution is individually deployable because technical changes are only necessary at the stub resolver running on the client. However, the technical proficiency needed to make such a change means that wide scale deployment will not happen via individual user action. We see three actors that may need appropriate motivations before large scale transition to direct client resolution is possible.

First, operating system developers who package and distribute stub resolvers with their OS need to add the functionality to perform recursive resolution. Logically, developers will add a feature if the benefits of adding the feature outweigh the costs. Recursive resolver software is readily available as a starting point. In this chapter, we have laid out a case for why client resolution is advantageous for users over using a shared resolver—a case that may be used to provide a competitive edge between OS developers. We note that the FreeBSD operating system already ships with a recursive resolver included [Fre]. By default, it is configured to an upstream resolver, but it can perform the full iterative resolution process with a configuration change.

Second, ISPs typically provide DNS recursive resolvers as a service to their clients. In direct client resolution, this hardware is no longer necessary. Additionally, as sketched in § 7.4, the load increase on the network is minimal. Together, these two observations

indicate that direct client resolution does not have a negative impact on ISPs and may actually reduce their cost in servers and maintenance. We also note that no technical changes are required of the ISP for their clients to perform direct client resolution. Also, the availability of open services (e.g., Google DNS [Goob] and OpenDNS [Opeb]) means that the ISP may shift their clients that do not use client resolution over to an open service. Thus we conclude that ISPs *do not* require motivation for a transition to client resolution.

Third, the TLD operators who will experience additional load need to be willing to incur the expense of additional infrastructure to mitigate the load. While it is true that no technical changes are needed in the DNS protocol to allow client resolution, any large scale transition must include a consideration of how the TLDs will resolve the costs. Ma, et al. [MCL+11] discusses the economics behind ISP peering and show a Shapely value solution for fair paid-peering between different types of ISPs. We believe a similar solution may exist between ISPs and TLDs involving ISPs paying TLDs for resolution service based upon the DNS traffic volume between the ISP and the TLD. We highlight this as important future work before large scale transition to client resolution is practical.

Additionally, we note that Verisign—operator of the TLDs ".com", ".net", ".edu", and many others—maintains shared resolvers for public use [Ver]. Thus, Verisign implicitly expresses a willingness to interface directly with client stub resolvers.

## 7.6 Summary

Traditionally, our community's response to security problems is to harden a protocol or its implementation. In this chapter we take an alternate approach to DNS security, suggesting a different factorization of the work that eliminates shared DNS resolvers. The benefit of this approach is to reduce DNS' attack surface. Through an initial study of a single network, we show that while there are costs, those costs are modest and manageable. For instance, less than 10% of TCP connections will be delayed by direct client resolution. Further, the

$99.9^{th}$ percentile load does not increase at all for 90% of authoritative DNS servers and by a factor of two at the ".com" TLD server–with no effort to mitigate the additional load. There are policy and privacy concerns, as well, but we believe this initial investigation shows that leaning on clients to do their own lookups deserves serious consideration. Further, we believe this effort illustrates that revisiting the fundamental way we arrange networks in the context of modern network realities may well be useful across other components of the system, as well.

# Chapter 8

# Conclusion and Future Work

In this dissertation, we show that the topology of DNS ecosystem is complex and parts of the ecosystem are hidden from external observers. We measure many components of the ecosystem including the clients and the various parts of the client-side DNS resolver infrastructure. The different parts of the infrastructure exhibit varying behaviors, including violations of the DNS specification and security vulnerabilities. As a result, reasoning about and maintaining the system is difficult. Thus, we propose removing the client-side resolution infrastructure from the resolution path and have clients perform their own iterative resolutions. This modification simplifies the resolution path and reduces the attack surface of the DNS.

In studying the DNS ecosystem, we first turn to the DNS client-side infrastructure which is responsible for resolving names on behalf of clients. To better understand the DNS client-side infrastructure, we develop tools to enable discovery of components of the infrastructure for further measurement. Since the infrastructure may vary by organization and some components have short lifetimes, we present a methodology to enable efficient discovery and rediscovery of large samples of the infrastructure in Chapter 3. Using these methodologies, we assess the topology and confirm previous observations that recursive resolvers operate in "pools" and that some actors that we expect to be close to-

gether may actually are be far apart. Further, our measurements show that the number of open resolvers—which can be used in reflection attacks and record injection attacks—on the Internet doubled between our measurements and previous measurements.

In Chapter 4, we present a set of methodologies for teasing apart the behavior of the actors within the system including components that cannot be directly probed. Using these methodologies, we assess the behavior of the client-side DNS infrastructure with respect to caching, both in aggregate and separately for distinct actors. We show that the various actors within the infrastructure handle the record time-to-live (TTL) value differently. Some actors put an upper threshold on the maximum TTL value while others increase the TTL of records with low TTL values. Further, we demonstrate that some actors continue to modify the TTL after the initial query, indicating that repeated probing is needed to capture the actor's full behavior. We also find that cache evictions due to capacity limits occur infrequently in recursive resolvers even for rarely accessed records. At the same time, while the TTL is frequently mis-reported to DNS clients, the majority of resolvers do not retain records 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.

We switch from studying the behavior of the client-side DNS infrastructure to the behavior of individual DNS clients in Chapter 5. Using datasets from a University campus with a 4 month interval between them, we provide a characterization of client behavior along dimensions that will ultimately inform an analytical model. We find clients interact with the DNS in distinct ways and develop markers for distinguishing general purpose user-facing devices from other types of devices. Further, we find that DNS queries often occur in clusters of related names, and short clusters account for the majority of DNS traffic. Lastly, we show that clients have a "working set" of names the size of which varies among clients, and the working set is stable over time and fairly distinct from other clients. Our high-level insights hold across the time period of our datasets and with qualitatively different user populations: students in campus housing vs. University offices and labs.

In Chapter 6, we uncover and measure a new attack vector against the DNS client-side infrastructure: the preplay attack. While record injection attacks against most actors in the DNS infrastructure require effort on the part of the attackers, 7–9% of open resolvers are vulnerable to a record injection attack that requires no guessing on the attackers part at all! The vulnerable devices are typically residential routers and inspection of their caches suggests that they are in use. We develop and deploy a Web-based tool for users to determine whether they are vulnerable to preplay and other forms of record injection.

In our measurements, we demonstrate the complexity of the DNS client-side infrastructure and observe hidden components, potentially making the system difficult to manage and troubleshoot. Further, the infrastructure remains vulnerable to record injection attacks including a new attack that we uncover. In Chapter 7, we study an alternative approach to DNS resolution: iterative resolution by the clients themselves. The benefits of this approach are ($i$) reduced complexity of the system compared to using multiple layers of shared resolvers, thus making the DNS potentially easier to troubleshoot, and ($ii$) a reduced DNS attack surface. Through simulations of direct client resolution using a dataset from a single network, we examine two potential costs. First, we find that the cost in delay of DNS resolutions is modest—only 7% of TCP connections are delayed by more than a few milliseconds. Second, the cost in load on authoritative DNS servers may be manageable and we suggest mitigation methods. For example, the average load on the "com." TLD authoritative DNS servers increases by 3.4 times using client resolution and our mitigation techniques reduced the load increase to below 1.5 times. There are privacy and policy considerations as well. However, we believe that our initial work shows that direct client resolution deserves serious consideration.

# Bibliography


[AAL+05] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. *http://www.faqs.org/rfcs/rfc4033.html*, 2005.

[ADL+10] M. Antonakakis, D. Dagon, X. Luo, R. Perdisci, W. Lee, and J. Bellmor. A Centralized Monitoring Infrastructure for Improving DNS Security. In *Recent Advances in Intrusion Detection*, pages 18–37, 2010.

[AK05] R. Arends and P. Koch. DNS for Fun and Profit. DFN-CERT Workshop, 2005.

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

[AM07] S. Ariyapperuma and C.J. Mitchell. Security Vulnerabilities in DNS and DNSSEC. In *IEEE International Conference on Availability, Reliability and Security*, 2007.

[AMSU10] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig. Comparing DNS Resolvers in the Wild. In *ACM Internet Measurement Conference*, pages 15–21, 2010.

[ARS13] H.A. Alzoubi, M. Rabinovich, and O. Spatscheck. The Anatomy of LDNS Clusters: Findings and Implications for Web Content Delivery. In *International Conference on World Wide Web*, 2013.

[BC98] P. Barford and M. Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *ACM SIGMETRICS*, pages 151–160, 1998.

[BCF+99] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-Like Distributions: Evidence and Implications. In *IEEE International Conference on Computer Communications*, 1999.

[Ber08] D. Bernstein. Introduction to DNSCurve. `http://dnscurve.org/`, 2008.

[Cal12] T. Callahan. Understanding Internet Naming: from the Modern DNS Ecosystem to new Directions in Naming. *Doctoral dissertation, Case Western Reserve University*, 2012.

[CAR13] T. Callahan, M. Allman, and M. Rabinovich. On Modern DNS Behavior and Properties. *ACM SIGCOMM Computer Communication Review*, pages 7–15, 2013.

[Cas] Case Connection Zone. `http://www.caseconnectionzone.org/`.

[CCG+04] J. Cao, W.S. Cleveland, Y. Gao, K. Jeffay, F.D. Smith, and M. Weigle. Stochastic Models for Generating Synthetic HTTP Source Traffic. *IEEE International Conference on Computer Communications*, 2004.

[CCR+03] 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 Computer Communication Review*, pages 3–12, 2003.

[CvdGLK15] C. Contavalli, W. van der Gaast, D. Lawrence, and W. Kumari. Client Subnet in DNS Queries. `http://datatracker.ietf.org/doc/draft-ietf-dnsop-edns-client-subnet/`, 2015.

[DAD+09] D. Dagon, M. Antonakakis, K. Day, X. Luo, C.P. Lee, and W. Lee. Recursive DNS Architectures and Vulnerability Implications. In *Network and Distributed System Security Symposium*, 2009.

[DPLL08] D. Dagon, N. Provos, C.P. Lee, and W. Lee. Corrupted DNS Resolution Paths: The Rise of a Malicious Resolution Authority. In *Network and Distributed System Security Symposium*, 2008.

[EKSX96] M. Ester, H.P. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *International Conference on Knowledge Discovery and Data Mining*, 1996.

[FA13] N.C. Fofack and S. Alouf. Modeling Modern DNS Caches. In *ACM International Conference on Performance Evaluation Methodologies and Tools*, 2013.

[Fre] FreeBSD Handbook, Network Servers, Domain Name System (DNS). `https://www.freebsd.org/doc/handbook/network-dns.html`.

[Fuj12] K. Fujiwara. Number of Possible DNSSEC Validators Seen at jp. In *DNS-OARC Workshop*, 2012.

[GC11] O. Gudmundsson and SD Crocker. Observing DNSSEC Validation in the Wild. In *Workshop on Securing and Trusting Internet Names*, 2011.

[GCD99] P. Gauthier, J. Cohen, and M. Dunsmuir. The Web Proxy Auto-Discovery Protocol. IETF Internet Draft. `https://tools.ietf.org/html/draft-ietf-wrec-wpad-01`, 1999.

[GD11] A. Gerber and R. Doverspike. Traffic Types and Growth in Backbone Networks. In *Optical Fiber Communication Conference*, 2011.

[Gooa] Websites Using Google Analytics. http://trends.builtwith.com/analytics/Google-Analytics.

[Goob] Google Load-Balancing for Shared Caching. https://developers.google.com/speed/public-dns/docs/performance#loadbalance.

[Gooc] Google Safe Browsing. https://developers.google.com/safe-browsing.

[GSG02] K.P. Gummadi, S. Saroiu, and S.D. Gribble. King: Estimating Latency Between Arbitrary Internet End Hosts. In *2nd ACM SIGCOMM Workshop on Internet Measurment*, pages 5–18, 2002.

[GYC+13] H. Gao, V. Yegneswaran, Y. Chen, P. Porras, S. Ghosh, J. Jiang, and H. Duan. An Empirical Reexamination of Global DNS Behavior. In *ACM SIGCOMM*, 2013.

[HBL12] C. Huang, I. Batanov, and J. Li. A Practical Solution to the Client-LDNS Mismatch Problem. *ACM SIGCOMM Computer Communication Review*, 2012.

[HMLG11] C. Huang, D.A. Maltz, J. Li, and A. Greenberg. Public DNS System and Global Traffic Management. In *IEEE International Conference on Computer Communications*, 2011.

[HS13] A. Herzberg and H. Shulman. Fragmentation Considered Poisonous, or: One-domain-to-rule-them-all.org. In *IEEE Communications and Network Security*, 2013.

[HTT] HTTP Archive. http://httparchive.org.

[JB99] S. Jin and A. Bestavros. Temporal Locality in Web Request Streams: Sources, Characteristics, and Caching Implications. Technical report, Boston University Computer Science Department, 1999.

[JB01] S. Jin and A. Bestavros. GreedyDual* Web Caching Algorithm: Exploiting the Two Sources of Temporal Locality in Web Request Streams. *Computer Communications*, 2001.

[JBB03] J. Jung, A.W. Berger, and H. Balakrishnan. Modeling TTL-Based Internet Caches. In *IEEE International Conference on Computer Communications*, 2003.

[Kam08] D. Kaminsky. Black Ops 2008: It's the End of the Cache As We Know It. *Black Hat USA*, 2008.

[LL10] D. Leonard and D. Loguinov. Demystifying Service Discovery: Implementing an Internet-wide Scanner. In *ACM Internet Measurement Conference*, pages 109–122, 2010.

[LSZ02] R. Liston, S. Srinivasan, and E. Zegura. Diversity in DNS Performance Measures. In *ACM SIGCOMM Workshop on Internet Measurment*, pages 19–31, 2002.

[Mah97] B.A. Mah. An Empirical Model of HTTP Network Traffic. In *IEEE International Conference on Computer Communications*, 1997.

[MCD+02] 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 Annual Technical Conference*, pages 229–242, 2002.

[MCL+11] R.T.B. Ma, D.M. Chiu, J. Lui, V. Misra, and D. Rubenstein. On Cooperative Settlement Between Content, Transit, and Eyeball Internet Service Providers. *IEEE/ACM Transactions on Networking*, pages 802–815, 2011.

[MGST70] R.L. Mattson, J. Gecsei, D.R. Slutz, and I.L. Traiger. Evaluation Techniques for Storage Hierarchies. *IBM Systems Journal*, 1970.

[Moc83a] P. Mockapetris. RFC 882: Domain names: Concepts and Facilities. *http://www.ietf.org/rfc/rfc882.txt*, 1983.

[Moc83b] P. Mockapetris. RFC 883: Domain names: Implementation and Specification. *http://www.ietf.org/rfc/rfc883.txt*, 1983.

[Moc87a] P. Mockapetris. RFC 1034: Domain Names - Concepts and Facilities. *http://www.ietf.org/rfc/rfc1034.txt*, 1987.

[Moc87b] P. Mockapetris. RFC 1035: Domain Names - Implementation and Specification. *http://www.ietf.org/rfc/rfc1035.txt*, 1987.

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

[Opeb] OpenDNS. `http://www.opendns.com/`.

[OSRB12] J.S. Otto, M.A. Sánchez, J.P. Rula, and F.E. Bustamante. Content Delivery and the Natural Evolution of DNS: Remote DNS Trends, Performance Issues and Alternative Solutions. In *ACM Internet Measurement Conference*, 2012.

[PAS+04] J. Pang, A. Akella, A. Shaikh, B. Krishnamurthy, and S. Seshan. On the Responsiveness of DNS-based Network Control. In *ACM Internet Measurement Conference*, pages 21–26, 2004.

[Pax94] V. Paxson. Empirically Derived Analytic Models of Wide-Area TCP Connections. *IEEE/ACM Transactions on Networking*, 1994.

[Pax99] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, 1999.

[PO12] V. Pappas and E. Osterweil. Improving DNS Service Availability by Using Long TTL Values. IETF Draft. `http://tools.ietf.org/id/draft-pappas-dnsop-long-ttl-04.txt`, 2012.

[QMZ+07] H. Qian, E. Miller, W. Zhang, M. Rabinovich, and C.E. Wills. Agility in Virtualized Utility Computing. In *IEEE Workshop on Virtualization Technology in Distributed Computing*, 2007.

[RMTP08] 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, 2008.

[Ros] J. Roskind. DNS Prefetching (or Pre-Resolving). `http://blog.chromium.org/2008/09/dns-prefetching-or-pre-resolving.html`.

[SAR14] K. Schomp, M. Allman, and M. Rabinovich. DNS Resolvers Considered Harmful. In *ACM Workshop on Hot Topics in Networks*, 2014.

[Sca] DNS Vulnerability Scanner. `http://dnstool.exp.schomp.info/`.

[Sch93] C. Schuba. *Addressing Weaknesses in the Domain Name System Protocol*. PhD thesis, Purdue University, 1993.

[SCKB06] A.J. Su, D.R. Choffnes, A. Kuzmanovic, and F.E. Bustamante. Drafting Behind Akamai (Travelocity-based Detouring). *ACM SIGCOMM Computer Communication Review*, pages 435–446, 2006.

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

[SCRA13] K. Schomp, T. Callahan, M. Rabinovich, and M. Allman. On Measuring the Client-Side DNS Infrastructure. In *ACM Internet Measurement Conference*, 2013.

[SCRA14] K. Schomp, T. Callahan, M. Rabinovich, and M. Allman. Assessing DNS Vulnerability to Record Injection. In *Passive and Active Measurement Conference*, 2014.

[Sis10] G. Sisson. DNS Survey. `http://dns.measurement-factory.com/surveys/201010/`, 2010.

[SKAT12] C. Shue, A. Kalafut, M. Allman, and C. Taylor. On Building Inexpensive Network Capabilities. *ACM SIGCOMM Computer Communication Review*, pages 72–79, 2012.

[SSDA12] M. Sargent, B. Stack, T. Dooner, and M. Allman. A First Look at 1 Gbps Fiber-To-The-Home Traffic. Technical Report 12-009, International Computer Science Institute, 2012.

[STA01] A. Shaikh, R. Tewari, and M. Agrawal. On the Effectiveness of DNS-based Server Selection. In *IEEE International Conference on Computer Communications*, pages 1801–1810, 2001.

[STI+] F. Scholz, Teoli, D. Illsley, Znerd, and E. Shepherd. Controlling DNS Prefetching. `https://developer.mozilla.org/en-US/docs/Web/HTTP/Controlling_DNS_prefetching`.

[TDLDA12] S. Tzur-David, K. Lashchiver, D. Dolev, and T. Anker. Delay Fast Packets (DFP): Prevention of DNS Cache Poisoning. *Security and Privacy in Communication Networks*, pages 303–318, 2012.

[UC] US-CERT. `https://www.us-cert.gov/`.

[Ver] Verisign Public DNS. http://www.verisign.com/en_US/innovation/public-dns/index.xhtml.

[VS12] P. Vixie and V. Schryver. DNS Response Rate Limiting (DNS RRL). Technical Report ISC-TN-2012-1, Internet Systems Consortium, 2012.

[Wan] S. Wan. Google Public DNS now auto-detects nameservers that support edns-client-subnet. https://groups.google.com/forum/?hl=en#!topic/public-dns-announce/67oxFjSLeUM.

[WMS03] C.E. Wills, M. Mikhailov, and H. Shang. Inferring Relative Popularity of Internet Applications by Actively Querying DNS Caches. In *ACM Internet Measurement Conference*, pages 78–90, 2003.

[YKMC06] L. Yuan, K. Kant, P. Mohapatra, and C.N. Chuah. DoX: A Peer-to-Peer Antidote for DNS Cache Poisoning Attacks. In *IEEE International Conference on Communications*, pages 2345–2350, 2006.

[Zal] M. Zalewski. p0f: Passive OS Fingerprinting tool. http://lcamtuf.coredump.cx/p0f.shtml.