

An Empirical Study of the Performance, Security and Privacy Implications of Domain Name Prefetching

Srinivas Krishnan and Fabian Monrose

Department of Computer Science,

University of North Carolina at Chapel Hill, Chapel Hill, NC, 27599, USA

{krishnan, fabian}@cs.unc.edu

Abstract—An increasingly popular technique for decreasing user-perceived latency while browsing the Web is to optimistically pre-resolve (or prefetch) domain name resolutions. In this paper, we present a large-scale evaluation of this practice using data collected over the span of several months, and show that it leads to noticeable increases in load on name servers—with questionable caching benefits. Furthermore, to assess the impact that prefetching can have on the deployment of security extensions to DNS (DNSSEC), we use a custom-built cache simulator to perform trace-based simulations using millions of DNS requests and responses collected campus-wide. We also show that the adoption of domain name prefetching raises privacy issues. Specifically, we examine how prefetching amplifies information disclosure attacks to the point where it is possible to infer the context of searches issued by clients.

Keywords—Domain Name System; Measurements; Security; Privacy

I. INTRODUCTION

Most savvy Internet users would probably not be surprised if told that detailed dossiers of their browsing habits were being kept by the online merchants they do business with. In fact, many people understand that getting better personalization on the Web often means divulging information about themselves. This sentiment has not gone unnoticed, and today, new markets for tracking and sharing online activities are blossoming [1]. As the surreptitious collection of personal information becomes more rampant, however, we are “losing our ability to understand and control those trade-offs to choose, *consciously and with awareness of the consequences*, what information about ourselves we disclose and what we don’t” [2].

This loss of control is exacerbated by several recent optimizations within modern browsers and monolithic search engines, all of which are geared towards improving responsiveness on the world’s largest distributed system — i.e., the Web. To help people find the answers to the questions they seek (e.g., directions to a weekend getaway, rec-

ommendations for that restaurant they just drove by, etc.) more readily, designers of such systems are constantly fiddling with new ways to improve responsiveness on the Web.

The soup du jour for decreasing user perceived latency is to optimize the use of the domain name system by *pre-resolving* (or *prefetching*) names in hyperlinks. Since DNS is responsible for translating human-readable names into IP addresses, nearly every initial visit to a website involves a name resolution. Thus, by proactively resolving hyperlinks in pages a user visits, the sites being referred to can be immediately contacted if, and when, the user decides to click on one of the links.

At first blush, this seems like a relatively neat idea. DNS itself was designed with high availability in mind, but because of its distributed nature, resolving a domain name often involves communication with at least one remote server, and in some cases, might require following referral chains across many servers—a process that could take seconds to complete. Prefetching, therefore, can lead to a much faster browsing experience for clients. As of April 2011, four of the top five browsers support prefetching (some more aggressively than others), and one major search engine (Google) employs an “Instant Search” feature wherein prefetching of links in the results page is performed *while* the user types in the search bar.

This trend is sure to continue, but as is the case with many of the classic trade-offs that arise in the design of any highly distributed system, this increase in speed does come at a cost. Indeed, one contribution of this paper is in raising awareness that DNS prefetching introduces significant performance overheads on name servers; we observed decreases in throughput of over 15% in our trace-based evaluations, high volumes of requests to non-existent domain names, and limited caching benefits overall. Moreover, we believe that the current state of the practice has dire implications

for the deployment of security extensions (called DNSSEC) aimed at making DNS more resilient [3]. These security extensions have witnessed a recent resurgence in interest, partly because of highly visible cache poisoning attacks [4]—that take advantage of the lack of security protections in DNS—to deceive unsuspecting users.

We also show that current practices for DNS prefetching enable new privacy threats that are ripe for abuse. In particular, we believe that prefetching introduces new ways for infringing on users’ privacy, to the extent that the *context* of their searches can be gleaned simply by inspecting DNS queries. The success of these attacks relies on the fact that prefetching inserts a significant amount of information into the cache of the name servers contacted by clients, allowing domain name operators to glean far more detailed insights than when this feature is turned off.

The rest of the paper is outlined as follows. Related work is presented in Section II. Section III covers our experimental setup and the data used in our large-scale evaluation. In Section IV, we provide an empirical study of the effect of prefetching, followed by an analysis of its impact on DNSSEC in Section V. In Section VI, we show how prefetching enables new disclosure attacks that unveil the context of searches performed by clients. We conclude in Section VII.

II. RELATED WORK

The domain name system plays a critical role in the operation of Internet applications, and so it is not surprising that understanding its performance has been the topic of much research (e.g., [5, 6]). These works all share the common goal of understanding how to improve DNS performance bottlenecks. In particular, Jung et al. [7, 8] provide extensive analysis of DNS performance and the effectiveness of caching. Additionally, several studies (e.g., [9, 10]) explore performance characteristics of DNSSEC. Unlike this paper, however, none of these work focus on DNS prefetching as it is a very recent practice. Also, to the best of our knowledge, the dataset and evaluation presented in this paper is the largest study on this subject.

More closely related to this paper is the work on DNS cache snooping. Grangeia [11] provides an excellent review of how to remotely inspect a cache for evidence of a *specific* lookup (e.g., www.nytimes.com). Remote cache inspection of this type has been used for a number of measure-

ment studies that include, for example, inferring the relative popularity of websites [12] and tracking malware infections [13]. In contrast, in this paper we explore how DNS prefetching amplifies new privacy threats, allowing one to gain far more insights than these prior techniques envisioned.

To date, a few proposals have been suggested for improving the responsiveness of connection establishment by optimistically issuing DNS queries. These ideas include prefetching of domain names based on popularity, prefetching of related domain names using piggyback schemes, and pre-caching of records based on several renewal policies (e.g., [14, 15]). However, none of these works perform large scale analyses or study the privacy and security implications for any of their prescribed prefetching policies.

Lastly, this paper significantly extends our preliminary work [16] to include a large-scale empirical evaluation on campus-wide data, new techniques for applying instance-based learning approaches to boost the accuracy of our search reconstruction process, as well as the design and implementation of a cache simulator. The simulator is used to study the impact that prefetching can have on the deployment of DNSSEC [3].

III. OVERVIEW

To aid in our pursuit of understanding the implications of browser-based DNS pre-resolution, we collected and analyzed two DNS datasets from several name servers at our campus. The first dataset (Summer) was collected from June to August 2010, and spans the summer break. The second dataset (Fall) was collected from September to November 2010. The trace collection was planned to span these two periods to allow for trend estimation as the population on campus increased. The monitored campus servers included the primary name servers for the entire wireless network, as well as the authoritative name servers for the University. This pool of servers was used by over 26,000 internal clients per day during the summer, over 44,000 internal clients in the Fall, and serviced an average of 42 and 63 million queries per day, respectively. Next, we first present our collection infrastructure and the techniques we used for isolating prefetching events.

A. Data Collection Infrastructure

Our data collection configuration consists of two core components, namely: *i*) a DNS trace collector

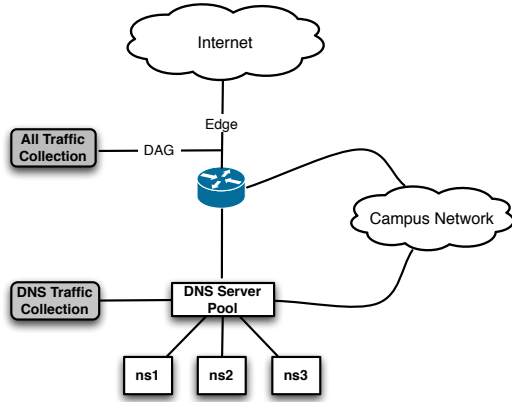


Figure 1. Campus-wide data collection and control framework.

and *ii*) a campus network trace collector (see Figure 1). The campus network is served by a single upstream provider, and is connected via a pair of optical links. The server pool we monitored is situated behind a load-balancer, and all wireless clients using the campus network are assigned a name server from this pool during their initial DHCP registration. DHCP leases on this network are bound to the client’s MAC address, and remain in effect for at least a few weeks. The DNS traces are collected via a mirrored port on the load-balancer switch, using CoralReef’s [17] engine to process and anonymize [18] the client IPs. The payloads are stored at a secure location housing the collection server.

To quantify the utility of a prefetching event, we also require access to network traces of connections to the resolved domains. To obtain this insight, we tapped the optical links from the campus to the upstream provider and mirrored the traffic to a network monitoring system. The network monitoring system is a Linux server with a specialized Data Acquisition and Generation (DAG) card that allows us to capture network traces at line rate with negligible loss and nanosecond timestamp accuracy. Since all campus traffic must traverse these links, we can observe all post name-resolution events for any arbitrary domain and determine whether TCP connections are made for that resolved name. For two five-day periods during the monitoring timeframe, we also captured all incoming and outgoing network traffic from the campus during peak hours. In this case, only packet headers are collected, and all client addresses were anonymized in a consistent fashion with the DNS traces. Our collection servers were

synchronized with a common NTP server.

Data Generation Framework: Apart from the real-world data we collected, we also required the ability to instrument and collect data from browsers in a controlled manner. To do so, we built a framework that allows us to fully automate our data generation process. This framework provides the basic functionality needed to inject keystrokes into several browsers and to automatically collect the resulting DNS data. To simulate user interaction, our framework accepts a set of terms, actions, and a desired typing rate, and injects keystrokes into a given browser. The actions dictate how the framework interacts with the browser, e.g., whether it enters keystrokes into the location bar or search engine. This framework is used for injecting the queries used as ground truth in our analysis in Section VI. We note that all subsequent performance results presented in this paper *excludes* any queries/responses resulting from the data generation framework.

B. Finding Prefetching Events

Recall that our main goal is to study the effects of browser-based DNS pre-resolution. To effectively filter DNS traffic not relevant to the study at hand, we apply heuristics to select only those queries that are created in response to browsing events; i.e., as a precursor to a web connection. To find the queries of interest, we take advantage of the fact that browsers rarely make non-recursive queries, i.e., they delegate the task of discovering the internet address of a qualified name to an external name server. Hence, the Recursion-Desired (RD) flag in DNS query packets for internal source addresses can be used to discard extraneous (i.e., non-browser related) DNS queries. Secondly, we filter address ranges for known services that use DNS (e.g., Active Directory and mail) within the university. Lastly, using the campus network traces, we discard DNS queries that result in ensuing connections to non-http/https servers (e.g., `itunes.apple.com`).

Having isolated the browser-related queries, the next challenge is in determining whether or not the queries belong to a pre-fetching event. Recall that pre-resolution of domain names is implemented as a means to reduce response latency on a potential click of a link on a website. This is realized in the browsers by extracting the `<href>` tags from each rendered page, and automatically performing lookups for the resulting domains. This implies

	All clients		Internal clients		Internal browser clients	
	Summer	Fall	Summer	Fall	Summer	Fall
Average Queries per day	42.4M	63.3M	32.1M	51.4M	20.3M	31.3M
Average Unique Clients per day	242,676	330,665	26,100	44,026	18,905	38,400
Average Cache Miss rate per day	20.8%±4.5%	22.0% ±4.3%	57.1% ±5.1%	55.2% ±3.1%	73.2% ±5.6%	69.1% ±4.1%

Table I
SUMMARY STATISTICS FOR THE SUMMER AND FALL 2010 DATASETS

that all the queries for the extracted domains will arrive at the name server within a small timeframe.

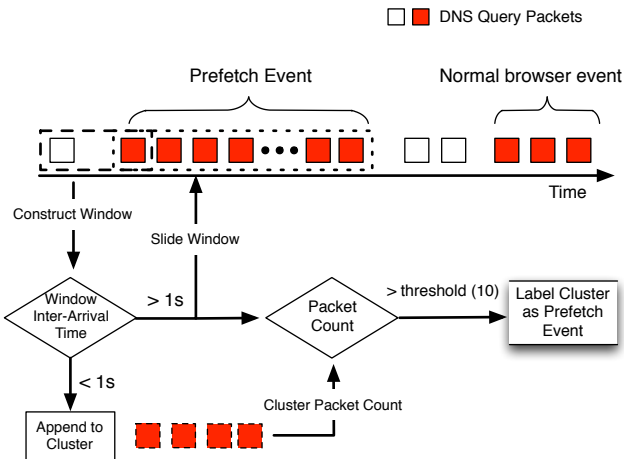


Figure 2. Identifying and labeling prefetching events.

We validated this observation by extracting all *http/https* connections in the campus trace during peak periods for 10 days. The related DNS queries were extracted for each web request using the connection’s destination address, and queries that did not match any web connection were labeled as ‘extraneous’. Close inspection of the query inter-arrival times (not shown) revealed that for a given client a valid query would be followed by a set of extraneous ones, and the inter-arrival time for 95% of these sets was less than 1 second. We take advantage of this observation to implement a straightforward approach for identifying prefetching events (see Figure 2).

A final task is to identify legitimate queries that are needed to successfully render a page, and separate them from extraneous queries within each event. Using our data generation framework, we rendered each webpage for Alexa’s Top-1000 websites and studied the DNS query patterns with prefetching turned off. We found that most pages induced queries for locating elements such as images, multimedia components, advertisements and trackers. Furthermore, these queries were usually to a select few domains (e.g., *akamai*, *doubleclick*,

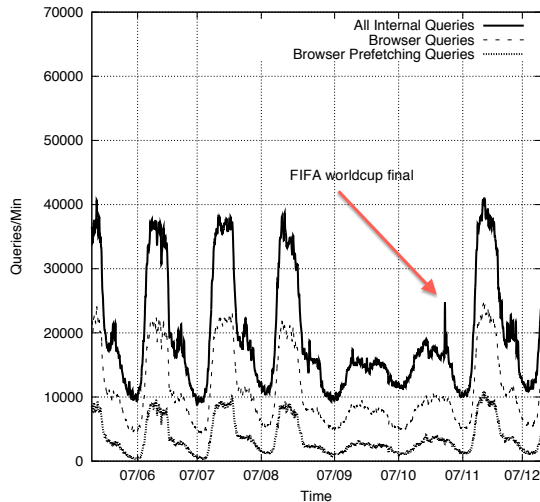
googleads) and were fairly easily identifiable by the prefix used in the query (e.g., *img*, *ads*, *cdn*). Having identified these prefixes, we took special care to eliminate them as part of any prefetching groups in which they appeared, since we considered them to be “required” elements for rendering the webpage in question. We validated the correctness of this step by comparing the DNS queries we deemed as being necessary for rendering a page to the actual observance of *http/https* connections in the campus trace. We found that 96.7% of the required queries were mapped to subsequent Web connections. To be clear, we do not count these required queries in the prefetching overheads.

IV. ON THE IMPACT OF PREFETCHING

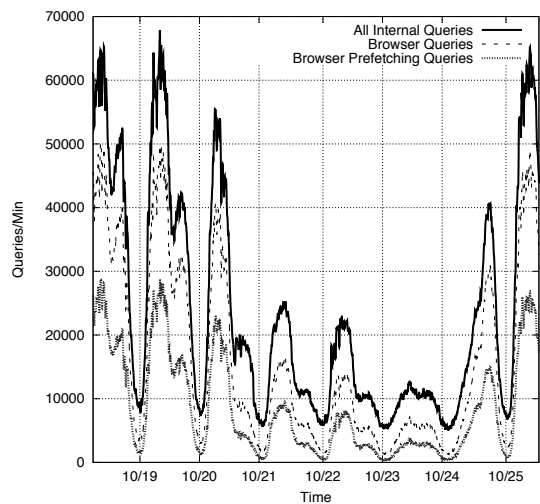
Before exploring the security and privacy implications of DNS pre-resolution, we first analyze the overhead it imposes on the name servers in our study. Table I summarizes some of the key statistics of our datasets. Note the increase in traffic in the Fall dataset, which corresponds to increased campus population with the start of the school year. The cache miss rate in Table I is calculated by observing whether the name server has to make an upstream query in order to satisfy a client’s request. The overall low cache miss rate when considering “all clients” is due to the fact that most queries from external clients result in cache hits because the vast majority of such queries are for authoritative records. The high cache-miss rate for internal request by browser clients, however, suggests a power-law distribution for domain name accesses. We return to this observation later in Section IV-A and Section VI.

Figures 3(a) and 3(b) depicts the load pattern for the internal UNC clients in 1-minute intervals for the Summer and Fall datasets, respectively. The graph shows that the weekly query load follows a diurnal pattern with most of activity happening during a workday. Peak requests hovered around 40,000 queries over the Summer and over 68,000 queries during the Fall semester with the influx of returning students.

Approximately 60% of the internal requests seen in the Fall are due to browser-related queries. We



(a) Query Load in July, 2010 (Summer)



(b) Query Load in October, 2010 (Fall)

Figure 3. Query load in 1-minute intervals, including the contribution of prefetching traffic for both datasets

then identified prefetching events for the browser queries using the approach outlined in Figure 2. During the summer they constitute, on average, 35% of the browser related queries, with peaks near 44%. Several spikes in the browser-related load over the summer can be attributed to searches for ‘hot topics’ (e.g., the peak during the FIFA world-cup final on 7/11) using Google’s search engine — in this case, a dynamic results page with real-time scrolling feeds is returned, wherein pre-resolution was performed.

As expected, prefetching activity increased in the Fall, and contributes over 45% of the browser

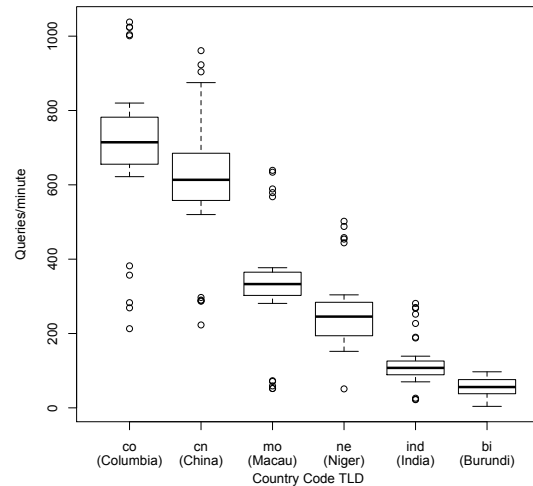


Figure 4. Box-and-Whisker plots for the number of non-existent (NX) answers observed per minute over 3 weeks

related queries with peaks near 55%. Interestingly, this increased load is not due solely to the increased student population, but could also have been influenced by the launch of ‘Instant Search’ by Google in October 2010. Instant search provides dynamically updating results as a user types in the search box—all the while pre-resolving on each new update to the search results. Overall, the induced load from prefetching is a disturbing sign, especially given the fact that the browser with over 60% market share (Internet Explorer) has yet to turn on DNS pre-resolution.

For ease of presentation, for the remainder of this section we present analysis for a representative week of data from the Fall dataset.

Typed-in Navigation: As eluded to earlier, some modern browsers try to guess the site the user is trying to visit, providing suggestions along the way to get her to the intended destination quickly. As the browser attempts to guess the user’s intention, DNS queries are created — most times after only a few characters are typed; For example, if a user starts to type `www.cnn.com` in a browser’s location bar, the browser attempts to autocomplete the user’s intended site generating queries such as `www.cn.`, `www.cnn.co.` and in some extreme cases `w.`, `ww.`, `www..` While some of these queries can be easily discarded as non-existent (NX) domains (`w.`, `ww.`, `www.`), several will trigger queries to nameservers since the queries inadvertently contain valid domains hosted in Colombia or China.

As a cursory examination, we studied the pre-resolutions that occur during typed-in navigation

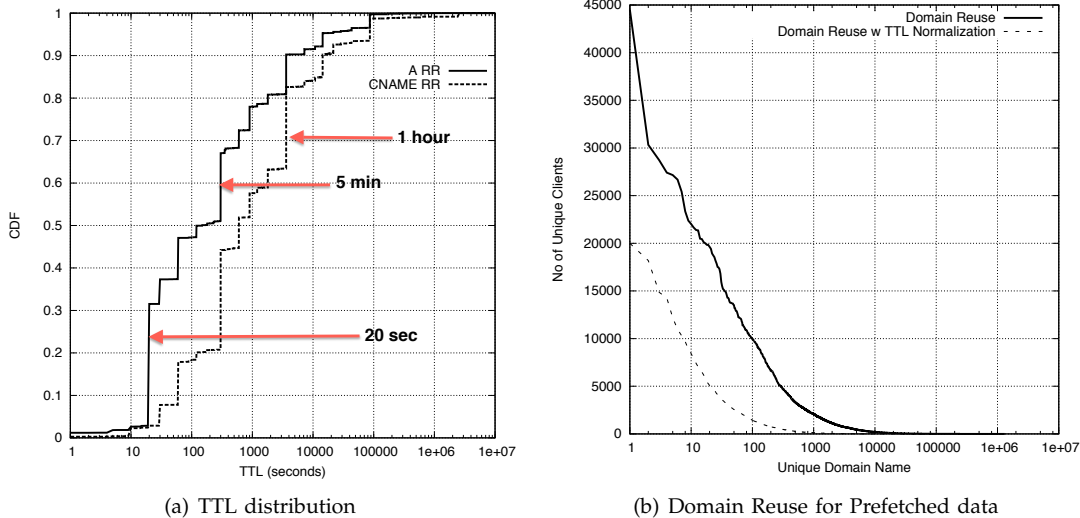


Figure 5. Domain Reuse Patterns and TTL distribution for common TTL value ranges

in Chrome. For the most part, the majority of the prefetches result in non-existent (NX) responses, or valid domains for sites other than the intended domain. Moreover, certain top-level domains are unduly affected by this practice, and receive the majority of these bogus queries. Figure 4 shows the responses per minute for browser auto-completion during a three week period. The box plot shows the mean, lower and upper quartiles, while the whiskers cover the min and max values. Circles denote outliers. Here we only consider a prefetching event as being related to auto-completion if it is followed by queries to the same name, but with a different gTLD, in a short window of time. Notice that for certain gTLDs (.co and .cn) the rate is very high (nearing 1000 queries/min in some cases); most are auto-completions for a user intending to visit a website with a .com TLD. Similar NX responses are generated from gTLDs shown in Figure 4 for users intending to visit websites with .mobi, .net, .info and .biz TLDs.

A. Impact on Caching

One would expect that active pre-resolving of domains on a per-client basis might not only improve user-perceived latency, but possibly have benefit to other clients. Such benefit would arise if the records fetched as part of a prefetching event remain in the server’s cache for extended periods of time. Figure 5(a) plots the TTL values of A and CNAME records in the Fall dataset. As we can see, 67.2% of the A records have TTL values of less than 5 minutes. Even worse, 31.2% of the records have

TTLs less than 20 seconds. The use of such low TTL values can be attributed to content distribution networks (CDNs). CDNs often set the TTL value for their A records on the order of minutes as a means of supporting dynamic load balancing [19].

Figure 5(a) also shows that CNAMEs, on the other hand, can reside in cache for long periods of time. CNAMEs are often used for redirection to a CDN, e.g. `<www.realestate.yahoo.com CNAME TTL 3600>` might redirect to `<rel.fy7.b.yahoo.com A TTL 300>`. Figure 5(a) shows that 37% of the CNAME records have TTLs of at least 1 hour, and over 7% of the records last longer than a day. A side effect of this is that CNAMEs offer many hints as to a site’s topical classification, and their high lifetime enables remote cache inspection attacks [16].

To explore the caching benefits of prefetching, we analyzed the reuse for domain names requested in prefetching events. The reuse value for a domain is computed based on the access frequency of all clients requesting that name. Figure 5(b) shows a Zipf-like reuse distribution, with 98% of the domain names accessed less than 10 times. However, from a caching perspective, an entry is only valid within its TTL window, so we normalized the access frequencies to take TTLs into consideration. The result shows that the overall usage pattern (even for popular domains) falls by about 40%. This means that for many of these domains, by the time they are requested again (*if at all*), the entry has already expired in the cache; thus the prefetched domains are of little value to others.

V. IMPACT ON DNS SECURITY EXTENSIONS

Over the past few years there has been steady progress in efforts to deploy DNSSEC, partly in response to highly visible cache poisoning attacks. Although common wisdom suggests that the overhead of DNSSEC is non-trivial, we examine the impact that prefetching can have on its deployment [3]. Specifically, we examine the additional overhead imposed by prefetching, particularly as it relates to the verification of responses for extraneous requests.

Before discussing the details of our analysis, we briefly review some key elements of DNSSEC. In short, DNSSEC extends the existing DNS architecture to use public key cryptography for securing the transactions between servers and clients. Essentially, each response in a DNSSEC enabled zone must be authenticated by building a “chain of trust” to a trusted anchor. These zones are signed using a *Zone Signing Key* (ZSK). The key is authenticated by traversing the zone hierarchy until a trusted anchor is reached. For a DNS response to be considered authentic, each step in the verification chain must succeed. The key elements [20] of DNSSEC pertinent to how verification works (and implemented in our simulator) are:

- a Resource Record Set (RRSet): A set of DNS Resource Records (RR) of the same type and TTL. In DNSSEC, signatures are created for the entire RRSet included in a response.
- a DNSKEY: The public key of the zone that signed the RRSet.
- a RRSIG: A signed digest of a RRSet.
- a Delegation Signer (DS): A record that contains the digest of a child zones’ DNSKEYs.

To study the impact of prefetching, we designed and implemented a cache simulator that performs trace driven simulations using the datasets described earlier (see Section III). The cache simulator has two components: a local caching resolver and a “remote server”. The caching resolver sends queries to the remote server and performs verification of the responses. The remote server emulates the DNS zone hierarchy and the nameservers for each zone. The zone hierarchy is created by first processing our network traces to identify the Top-Level Domains and Second-Level Domains. Each zone is then assigned a unique 1024-bit RSA key-pair, of which the private key is used to sign the RRsets and the DS record of its child zone.

The caching resolver drives the simulation by re-

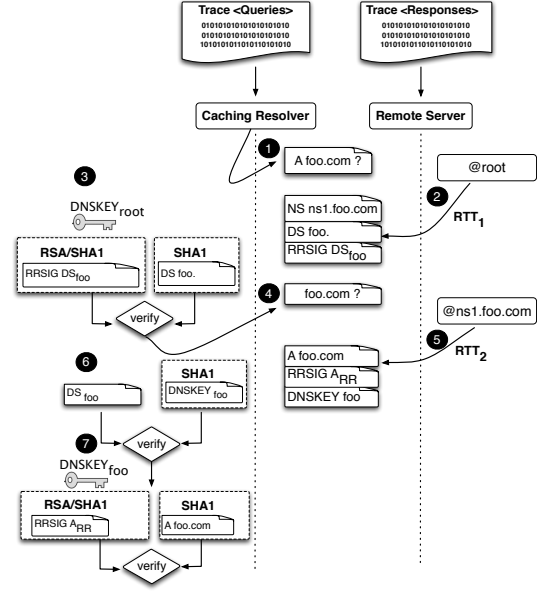


Figure 6. An example DNSSEC simulation. RTT_1 and RTT_2 are derived from the actual network trace

playing the queries from the trace. The queries are forwarded to the remote server where the correct (signed) response based on the trace is returned. For each response we also simulate the network latency for contacting the remote resolver using a normalized average of the observed RTT from the network trace. Once the caching resolver receives a response, it verifies the response and caches the result for the specified TTL. As in Bindv9, we use the OpenSSL cryptographic library to perform all cryptographic operations. We used SHA1 as our digest function. Subsequent queries within the TTL period are served directly from cache.

Figure 6 depicts an example interaction between the caching resolver and remote server components of the simulator for resolving a query for `foo.com`. The simulator is initialized with only the public key of the root zone ($DNSKEY_{root}$). *Step 1*: the query is “forwarded” to the remote server. *Step 2*: the reply from the server (on behalf of the root) contains the NS record and also a signed digest (i.e., a DS record) of the public key ($DNSKEY_{foo.com}$) for the `foo.com` zone. *Step 3*: the DS record is verified using $DNSKEY_{root}$. *Step 4*: the resolver then contacts the nameserver for the `foo.com` zone and receives from `ns1.foo.com`, the A record for `foo.com`, a signed digest of the A record (i.e., its RRSIG), and also $DNSKEY_{foo.com}$. *Step 5*: the resolver verifies the authenticity of $DNSKEY_{foo.com}$ by comparing the cryptographic hash of $DNSKEY_{foo.com}$ with the

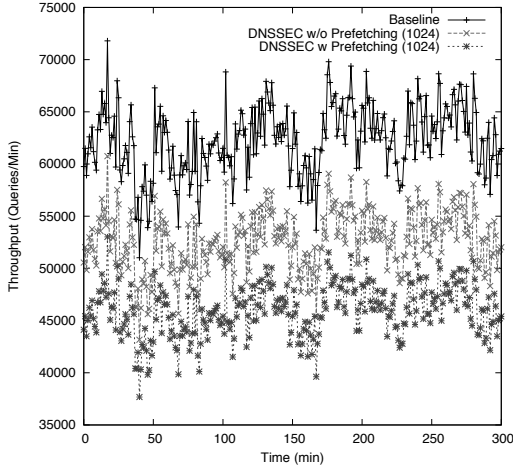


Figure 7. Resolver throughput

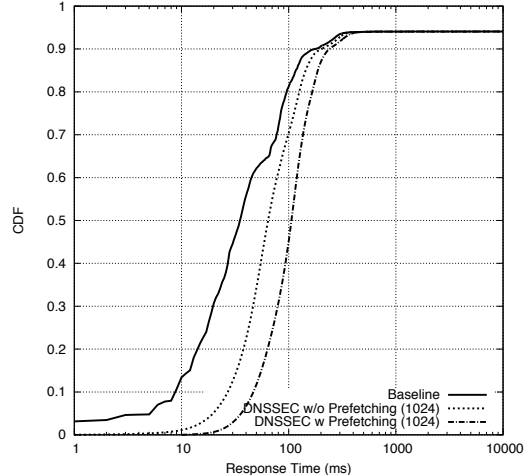
DS record obtained earlier from the root. Finally in Steps 6 and 7: the resolver verifies the A record by checking the authenticity of the associated RRSIG using `DNSKEYfoo.com`, and compares the embedded cryptographic hashes in the signed digest to the hash of the returned A record.

A. Results

In what follows, we use a 4 hour period during peak load in the Fall dataset as input to our trace-driven simulations. The trace contains 23.6 million records. The overheads are computed relative to a baseline where DNSSEC capability was disabled.

Obviously, as verification of responses requires additional CPU cycles, it has a direct effect on the throughput of the resolver. Figure 7 shows the throughput achieved by the cache simulator for fully resolving DNS queries. The average throughput of 62,345 queries per min (*qpm*) decreases by 24.7% (to 46,964 *qpm*) when verification is enabled. However, when prefetched records are excluded, the drop in throughput relative to the baseline is about 16.1% (to 52,307 *qpm*).

The drop in throughput causes a significant increase in response times as observed by the clients. Here, response times reflect both the simulated RTT (to fetch the response) as well as the verification time at the resolver itself. Figure 8 shows the response time for the same experiment. Notice that over 80% of clients get a response within 100ms in the baseline. With verification on, but with pre-resolutions excluded, over 70% of the clients still get answers within 100ms. However, when responses for extraneous requests must also be verified, only 48% of the clients receive an answer



within 100ms. This result is particularly worrying since prefetching was introduced as optimization to reduce client response times, but when DNSSEC is turned on, even valid requests see a considerable increase in response time.

Finally, notice that for clients that witnessed response times over 300 ms, all three scenarios show similar performance. This is because the high RTT (from the trace) observed by these clients masks the cost of verifying responses. We now turn our attention to the privacy implications of this practice of pre-resolving domain names.

VI. PRIVACY IMPLICATIONS

As mentioned earlier, we believe DNS prefetching offers Internet providers yet another avenue to infringe on a user’s privacy, to the extent that the context of their searching activity may no longer be private. Indeed, very detailed information can be harvested from these requests without ones knowledge. The threat model we consider here assumes access to DNS logs (e.g., as a DNS provider would have), which we argue is a realistic threat since many DNS providers have already acknowledged their interests to “aggregate non-personally-identifying information about the behavior of visitors to its websites and customers of its DNS services” [21].

We examine the potential for abuse under this threat model by describing an inference attack that allows an attacker to reconstruct users search queries by simply observing their DNS queries. The key observation here is that domain names often contain identifiable words as to the nature of a website, and so if an adversary can observe the

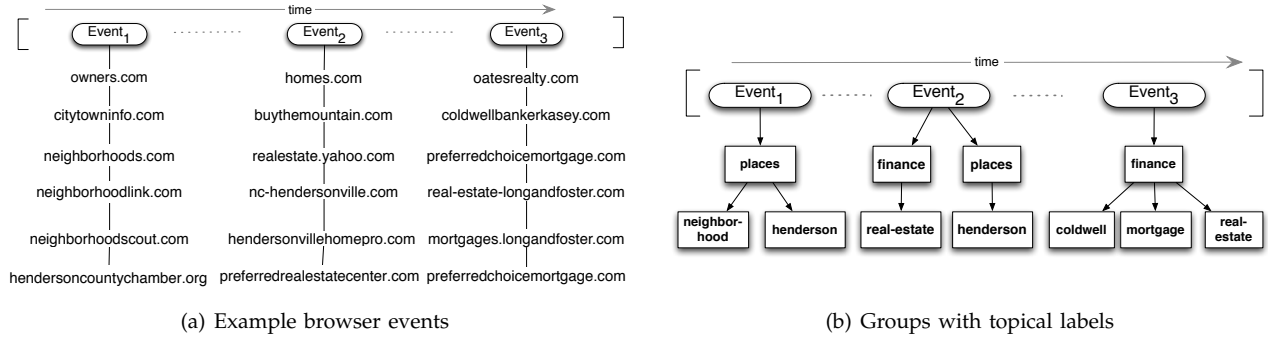


Figure 9. Clients browsing activity and reconstructed search terms due to prefetching

domain names that were contained in, for example, a search results page, then she would be able to reliably reconstruct the client’s search term. Since prefetching creates identifiable clusters of DNS queries, we propose a technique for reconstructing the search terms by first identifying prefetching events, and then applying algorithms similar to the auto-completion techniques used by search engines to provide suggestions to users. In doing so, we are able to make intelligent guesses as to the client’s likely search terms. The overall approach we take is as follows:

Step 1 Grouping related entries: For pedagogical purposes, we remind the reader how users typically find information on the Web. Loosely speaking, they input a set of keywords into a search engine, and then explore the ranked set of returned links. Figure 9(a) shows an example browsing session with three distinct groupings of search based prefetching events from real data. These groupings were identified using the heuristic described in Figure 2. Obviously, while one could manually sift through the data and find relationships across groups, with thousands of clients and hundreds of queries per second, this would be impractical. We provide an automated technique for achieving our goals with high accuracy.

Step 2 Keyword extraction: Once events have been grouped, the next task is to extract meaningful keywords from the domains names in each group. To do so, each domain name is tokenized into keywords using a sliding window algorithm. We generate template words for each window by using a n -recommend algorithm [22], which provides word suggestions given a prefix. The process is initialized by using first m -characters ($m = 4$) of a domain name as the prefix. The longest word template match on the domain name

is selected as an identified keyword, the window slides over to the end of the match and the entire process is repeated until the entire domain name is broken down into a set of recognizable words. The result is a list of keywords, ordered from left to right, for each domain name. If a domain name cannot be broken down into constituent words, we categorize it using a content classification engine (e.g. Google Insights) that labels the domain with a broad topical classification. In this case, only the domain names (i.e., no client specific information) is sent to the topical classification engine. At this stage, a sample output for Event₁ in Figure 9(a) would be $\langle neighborhood, scout, city, town, henderson, county, chamber, owner \rangle$.

Step 3 Query Reconstruction: After the keywords are tokenized and ranked, we locate groups containing a search engine domain name (e.g., *google, bing*) and then attempt to recreate the actual search query within each group. In order to reconstruct the search query, we again take advantage of an n -recommender systems, and provide as input all the first order words associated with each grouping. This yields a list of suggested queries. Each suggestion is compared with our list of ordered words, and we output (as our inferred query) the suggestion with the maximum number of matches. For the running example, the output for Event₁ is now: $\langle Henderson Neighborhood \rangle$.

As an optimization, we take advantage of the observation that a user’s browsing history, or her interactions on social networks, can be used to find identifying patterns in her access behavior [23]. Specifically, we augment our query reconstruction scheme to utilize related tokens across multiple browsing events. Revisiting the earlier example, the events in Figure 9(a) shows that the search for $\langle Henderson Neighborhood \rangle$ (Event₁) was followed



(a) Snapshot of popular searches (July)



(b) Aggregate of individual Queries (July)

Figure 10. A tag cloud depicting several automatically reconstructed searches

by a search for $\langle \text{real-estate henderson} \rangle$ ($Event_2$) and $\langle \text{mortgage real-estate} \rangle$ ($Event_3$). This additional contextual information can be used to link the reconstructed queries of each event, thereby allowing for a more specific picture of the user’s interests: *buying a house in henderson*.

To automate this process, we apply an instance-based learning method to relate individual browsing events and recreate a user’s browsing profile. In order to find similar browsing events, we first use a topical classifier to label each event under a broad-category. For example, real-estate and mortgage can be broadly classified as related to Finance. The resulting data has each group labeled with a set of possible topics as shown in Figure 9(b). Once all the elements have been labeled with a topic, we then apply a pattern matching algorithm to find related terms by checking if a event contains topics in its groupings that match to an already scanned event. If a match is found, we consider the two groupings as being related. For example, we would consider all three events as related because of the linkage made by the elements `finance` and `places`→`henderson`; here, the symbol “→” denotes a subcategory. A sample output is now: $\langle \text{Real Estate Henderson Neighborhood} \rangle, \langle \text{Mortgage Henderson Neighborhood} \rangle$.

A. Evaluation

For illustrative purposes, Figure 10(a) presents a tag cloud of the most popular reconstructed browsing activity during the first week of July. Part (b) presents a similar illustration, but for a 4 hour period for 10 randomly picked client IPs. For privacy reasons, we display the results as a group, as doing otherwise would leak very specific information about the browsing behavior of some clients. While these results aptly demonstrate the scope of the privacy breaches, evaluating the accu-

racy of our approach calls for ground truth.

To allow us to perform such an analysis, we make use of our data generation framework and use it to generate queries from 10 machines simulating clients that run a combination of browsers (Chrome, Safari, Firefox, Internet Explorer, and Opera). The searches made by these clients were randomly chosen from a list of 1,000 topics derived from Google Trends (from January to June 2010), controversial topics from Wikipedia, and Alexa Hot Topics (from February to May 2010). The list is labeled by topic, with each topic having a set of associated domain names.

During a week long period, we performed two non-overlapping experiments: one where each client selects a term with uniform probability, and the other where the selection is biased by assigning higher probabilities to domains and queries in related topical areas (e.g., if the topic picked is health-care, there is a high probability that subsequent queries by that client will be related to health-care). Once a client picks a topic, it randomly performs either a search query, click on a link on the returned search results, or use the location bar to navigate to a domain name associated with the topic. The client does so for no more than 20 minutes, with uniformly random chosen “think times” of 1-5 minutes. At the end of the browsing session, the client chooses a new search topic. No two clients are allowed to choose the same topic. All the activity is timestamped and logged.

Next, using all the recorded data spanning that time period (recall queries from real clients are also occurring simultaneously) we attempted to reconstruct the searches. Our results are then compared with the data logged at the generation framework in order to compute our true positive and false positive rates. Let the set of words in the original

(a) Using single events					(b) Instance-based learning				
Source	Prefetching On		Prefetching Off		Source	Uniform		Non-Uniform	
	TP%	FP%	TP%	FP%		TP%	FP%	TP%	FP%
Hot Trends	91.2	8.8	7.9	42.6	Google Insights	85.1	6.5	89.0	4.5
Google Insights	87.9	6.5	6.8	48.9	Wikipedia	87.3	7.1	93.4	5.2
Wikipedia	86.5	4.5	4.3	47.1	Alexa	87.4	6.2	94.1	4.1
Alexa	88.2	4.1	8.2	46.1					

Table II
ACCURACY OF RECONSTRUCTION TECHNIQUES USING BOTH SINGLE EVENTS (A) AND INSTANCE-BASED LEARNING (B).

logged query be defined as Q_w , and the set of words in the result be R_w . Then, the *true positive* and *false positive* rates for R_w are computed as:

$$TP = \begin{cases} \frac{|R_w \cap Q_w|}{|Q_w|} & \text{if } R_w \subseteq Q_w \\ 1.0 & \text{if } R_w = Q_w \\ 0.0 & \text{if } Q_w \subseteq R_w \end{cases}$$

$$FP = \begin{cases} \frac{|R_w \setminus Q_w|}{|Q_w|} & \text{if } Q_w \subseteq R_w \\ 1.0 & \text{if } R_w \neq Q_w \\ 0.0 & \text{if } R_w \subseteq Q_w \end{cases}$$

The results of our experiments are shown in Table II, broken down by each source for which we recreated a search query. Table II(a) shows the accuracy when the reconstruction is restricted to using data from a single prefetching event. In this case, our accuracy is dependent on the number of identifiable keywords we are able to recreate from the domains in a prefetching event. An interesting observation here is the high true positive rate for recreating “Hot Trends”. This is a result of there being far more entries in prefetching events for hot search terms, especially in the case where Google’s results page includes dynamically updated references for that search. These additional requests lead to the extraction of more keywords, thereby improving our ability to reconstruct these searches. At the same time, however, the false positive rate is higher than the other target cases because many of the domains are shared among searches of hot trends happening around the same time. Notice, however, that our reconstruction accuracy is abysmal when prefetching is turned *off*.

Table II(b) depicts our results for each client, where we only consider events in 15 minute intervals. As expected, the non-uniform case—which better reflects how people search in practice—outperforms the case where the selection is not biased, and achieves true positive rates of over 92%, with false negatives under 5%, on average. In lieu of our earlier observation regarding the poor reconstruction accuracy when prefetching is turned off, we omit results for the instance-based learning

case as performance there is highly dependent on accurately reconstructing single events.

VII. CONCLUSION

Our main objective in this work is to highlight the fact that if left unchecked, rapid enhancements in *when* and *how* DNS prefetching is performed can have significant performance implications, and can also lead to new security and privacy issues. To date, prefetching is rapidly being deployed within modern browsers, and has already been activated in browsers of mobile devices (e.g., Safari on the iPhone and Chrome on Android)—some of which offer no straightforward way to disable this feature. Lately, it appears that some browser architects have begun experimenting with alternative prefetching strategies along the lines of those suggested in our earlier work (e.g., only prefetching when the mouse hovers over a link) [16]. We applaud those changes, but believe much more should, and could, be done. We believe that it is prudent to stop and rethink the need for pre-resolution, as the adverse effects on name server load that comes with the expanded use of DNS prefetching is clear. Moreover, this practice paints a grim outlook for the adoption of DNSSEC. Lastly, our ability to reconstruct search queries when prefetching is enabled underscores the thin line we walk between increased performance and privacy.

VIII. ACKNOWLEDGEMENTS

We express our gratitude to Joni Keller, Jim Gogan, Danny Shue, Sid Stafford, Hiawatha Demby, Stan Waddell, Alex Everett (all from the Information Technology Service Office), Murray Anderegg and Bil Hayes (our local networking gurus) for their tremendous efforts in deploying the infrastructure for this study. The researchers and the University’s Technology Service Office have a longstanding memorandum of understanding (MoU) in place to collect anonymized network traffic on campus. The MoU covers specific uses and types of networking data, as well as conditions

for securing and accessing such data. We also thank Michael Bailey, Kevin Jeffay, Don Smith, Teryl Taylor and the anonymous reviewers for helpful suggestions. This work is supported by NSF grant CNS-0831245.

REFERENCES

- [1] J. Valentino-Devries, "What They Know About You," *Wall Street Journal*, July 2010.
- [2] N. Carr, "Tracking is an Assault on Liberty, With Real Dangers," *Wall Street Journal*, August 2010.
- [3] E. Osterweil, M. Ryan, D. Massey, and L. Zhang, "Quantifying the Operational Status of the DNSSEC Deployment," in *ACM IMC*, 2008, pp. 231–242.
- [4] S. Son and V. Shmatikov, "The Hitchhiker's Guide to DNS Cache Poisoning," *Intl. Conf. on Security and Privacy in Communication Networks*, Sept. 2010.
- [5] D. Wessels and M. Fomenkov, "Wow, That's a Lot of Packets," in *Passive and Active Measurement Workshop*, April 2003.
- [6] R. Liston, S. Srinivasan, and E. Zegura, "Diversity in DNS Performance Measures," in *ACM SIGCOMM Workshop on Internet measurement*, 2002, pp. 19–31.
- [7] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, "DNS Performance and the Effectiveness of Caching," *IEEE/ACM Trans. on Networking*, vol. 10, no. 5, pp. 589–603, 2002.
- [8] J. Jung, A. W. Berger, and H. Balakrishnan, "Modeling TTL-based Internet Caches," in *Infocom*, 2003.
- [9] B. Ager, H. Dreger, and A. Feldmann, "Predicting the DNSSEC Overhead Using DNS Traces," in *Information Sciences & Systems*, 2006, pp. 1484–1489.
- [10] W. Wijngaards and B. Overeinder, "Securing DNS: Extending DNS Servers with a DNSSEC Validator," *Security & Privacy*, vol. 7, no. 5, pp. 36–43, 2009.
- [11] L. Grangeia, "DNS Cache Snooping or Snooping the Cache for Fun and Profit," *SideStep Seguranca Digital*, Tech. Rep., Feb. 2004.
- [12] C. E. Wills, M. Mikhailov, and H. Shang, "Inferring Relative Popularity of Internet Applications by Actively Querying DNS Caches," in *ACM IMC*, 2003, pp. 78–90.
- [13] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A Multifaceted Approach to Understanding the Botnet Phenomenon," in *ACM IMC*, Oct., 2006, pp. 41–52.
- [14] E. Cohen and H. Kaplan, "Proactive Caching of DNS Records: Addressing a Performance Bottleneck," in *Symp. on Apps. and the Internet*, 2001, pp. 85–94.
- [15] H. Shang and C. E. Wills, "Piggybacking Related Domain Names to Improve DNS Performance," *Computing Networking*, vol. 50, no. 11, pp. 1733–1748, 2006.
- [16] S. Krishnan and F. Monrose, "DNS Prefetching and its Privacy Implications: When good things go bad," in *USENIX Workshop on Large-scale Exploits and Emergent Threats*, April 2010.
- [17] D. Moore, K. Keys, R. Koga, E. Lagache, and K. C. Claffy, "The CoralReef Software Suite as a Tool for System and Network Administrators," in *USENIX Conf. on System Admin.*, 2001, pp. 133–144.
- [18] J. Fan, J. Xu, M. Ammar, and S. Moon, "Prefix-preserving IP Address Anonymization: Measurement-based Security Evaluation and a New Cryptography-based Scheme," *Computer Networks*, vol. 46, no. 2, pp. 263–272, October 2004.
- [19] A.-J. Su, D. R. Choffnes, A. Kuzmanovic, and F. E. Bustamante, "Drafting Behind Akamai," *SIGCOMM Computing Comm. Review*, vol. 36, no. 4, pp. 435–446, 2006.
- [20] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS Security Introduction and Requirements (RFC 4033)," 2005.
- [21] OpenDNS, "Privacy Policy," See <http://www.opendns.com/privacy/>, July 2007.
- [22] M. Deshpande and G. Karypis, "Item-based Top-N Recommendation Algorithms," *ACM Transactions on Info. Sys.*, vol. 22, no. 1, pp. 143–177, 2004.
- [23] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida, "Characterizing User Behavior in Online Social Networks," in *ACM IMC*, 2009, pp. 49–62.

APPENDIX

For Chrome (version 10), DNS prefetching can be disabled by unmarking the check box "use DNS prefetching to improve page load performance" via the *Tools* → *Options* → *Under the Hood* sub-menu (this is true even on Android smartphones). For Firefox (version 4), disabling this feature is less obvious. Users can do so by setting the `network.dns.disablePrefetch` preference to true via the `about:config` method. For some versions of Firefox, it appears that the `network.dns.disablePrefetchFromHTTPS` preference should also be set to true in order to fully disable DNS prefetching. Similarly, for other Mozilla Necko-based apps (like Thunderbird), these preferences can be set by editing the `user.js` file in the user's profile folder.

Under MacOS X prefetching for Safari (version 5) can be turned off by typing `defaults write com.apple.safari WebKitDNSPrefetchingEnabled -boolean false` within the Terminal. Unfortunately we are not aware of any easy way to disable prefetching under iOS for mobile devices.