DNS Performance and the Effectiveness of Caching

Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris
MIT Laboratory for Computer Science
200 Technology Square
Cambridge, MA 02139
{jyjung, sit, hari, rtm}@lcs.mit.edu

Abstract—**This paper presents a detailed analysis of traces of DNS and associated TCP traffic collected on the Internet links of the MIT Laboratory for Computer Science and the Korea Advanced Institute of Science and Technology (KAIST). The first part of the analysis details how clients at these institutions interact with the wide-area DNS system, focusing on performance and prevalence of failures. The second part evaluates the effectiveness of DNS caching.**

In the most recent MIT trace, 23% of lookups receive no answer; these lookups account for more than half of all traced DNS packets since they are retransmitted multiple times. About 13% of all lookups result in an answer that indicates a failure. Many of these failures appear to be caused by missing inverse (IP-to-name) mappings or `NS` records that point to non-existent or inappropriate hosts. 27% of the queries sent to the root name servers result in such failures.

The paper presents trace-driven simulations that explore the effect of varying TTLs and varying degrees of cache sharing on DNS cache hit rates. The results show that reducing the TTLs of address (`A`) records to as low as a few hundred seconds has little adverse effect on hit rates, and that little benefit is obtained from sharing a forwarding DNS cache among more than 10 or 20 clients. These results suggest that the performance of DNS is not as dependent on aggressive caching as is commonly believed, and that the widespread use of dynamic, low-TTL `A`-record bindings should not degrade DNS performance.

I. Introduction

The Domain Name System (DNS) is a distributed database mapping names to network locations, providing information critical to the operation of most Internet applications and services. Internet applications depend on the correct operation of DNS, and users often wait while browsing because of delays in name resolution rather than object download. Yet, despite its importance, little has been published over the last several years on how well this essential service performs.

This research was sponsored by Defense Advanced Research Projects Agency (DARPA) and the Space and Naval Warfare Systems Center San Diego under contract N66001-00-1-8933, and by IBM Corporation.

The last large-scale published study of DNS was Danzig *et al.*'s analysis of root server logs in 1992 [1], which found a large number of implementation errors that caused DNS to consume about twenty times more wide-area network bandwidth than necessary. Since 1992, both the Internet and DNS have changed in significant ways: for example, today's traffic is dominated by the Web, and the past few years have seen DNS being used in unanticipated ways, such as in Web server selection.

This paper seeks to understand the performance and behavior of DNS and explain the reasons for its scalability. Specifically:

1. How reliable and efficient is DNS from the point of view of clients?
2. What role do its scaling mechanisms—hierarchical organization and aggressive caching—play in ensuring scalable operation?

It is widely believed that two factors contribute to the scalability of DNS: hierarchical design around administratively delegated name spaces, and the aggressive use of caching. Both factors reduce the load on the root servers at the top of the name space hierarchy, while successful caching reduces delays and wide-area network bandwidth usage. The DNS caching design favors availability over freshness. In general, any DNS server or client may maintain a cache and answer queries from that cache, allowing the construction of hierarchies of shared caches. The only cache coherence mechanism that DNS provides is the time-to-live field (TTL), which governs how long an entry may be cached.

One way to gauge the effectiveness of caching is to observe the amount of DNS traffic in the wide-area Internet. Danzig *et al.* report that 14% of all wide-area packets were DNS packets in 1990, compared to 8% in 1992 [1]. The corresponding number from a 1995 study of the NSFNET by Frazer was 5% [2], and from a 1997 study of the MCI backbone by Thompson *et al.* was 3% [3]. This downward trend might suggest that DNS caching is working well.

However, the 1997 3% number should be put in perspective by considering it relative to network traffic as a whole. The same study showed that DNS accounts for 18% of all flows, where a flow is defined as *uni-directional* traffic streams with unique source and destination IP addresses and port numbers and IP protocol fields. If one assumes that applications typically precede each TCP connection with a call to the local DNS resolver library, this suggests a DNS cache miss rate of a little less than 25%. However, most TCP traffic is Web traffic, which tends to produce groups of about four connections to the same server [4]; if one assumes one DNS lookup for every four TCP connections, the "session-level" DNS cache miss rate appears to be closer to 100%. While an accurate evaluation requires more precise consideration of the number of TCP connections per session and the number of DNS packets per lookup, this quick calculation suggests that DNS caching is not very effective at suppressing wide-area traffic.

An analysis of the effectiveness of DNS caching is especially important in light of several recent changes in the way DNS is used. Content distribution networks (CDNs) and popular Web sites with multiple servers are increasingly using DNS as a level of indirection to help balance load across servers, or for fault tolerance, or to direct each client request to a topologically nearby server. Because cached DNS records limit the efficacy of such techniques, many of these multiple-server systems use TTL values as small as a few seconds or minutes. Another example is in mobile networking, where dynamic DNS together with low-TTL bindings can provide the basis for host mobility support in the Internet [5], [6]. Again, this use of DNS conflicts with caching.

A. *Summary of Results*

This paper explores the following questions:
1. What performance, in terms of latency and failures, do DNS clients perceive?
2. What is the impact on caching effectiveness of choice of TTL and degree of cache sharing?

These questions are answered using a novel method of analyzing traces of TCP traffic along with the related DNS traffic. To facilitate this, we captured all DNS packets and TCP `SYN`, `FIN`, and `RST` packets at two different locations on the Internet. The first is at the link that connects MIT's Laboratory for Computer Science (LCS) and Artificial Intelligence Laboratory (AI) to the rest of the Internet. The second is at a link that connects the Korea Advanced Institute of Science and Technology (KAIST) to the rest of the Internet. We analyze two different MIT data sets, collected in January and December 2000, and one KAIST data set collected in May 2001.

23% of all client lookups in the most recent MIT trace fail to elicit any answer, even a failure indication. The query packets for these unanswered lookups, including retransmissions, account for more than half of all DNS query packets in the trace. 3% of all unanswered lookups are due to loops in name server resolution, and these loops on average cause over 10 query packets to be sent for each (unanswered) lookup. In contrast, the average answered lookup sends about 1.3 query packets.

In the same trace, 13% of lookups result in an answer that indicates an error. Most of these errors indicate that the desired name does not exist. While no single cause seems to predominate, inverse lookups (translating IP addresses to names) often cause failures, as do `NS` records that point to non-existent servers. 27% of lookups sent to root servers resulted in errors.

While median name resolution latency was less than 100 ms, the latency of the worst 10% grew substantially between January and December 2000. Roughly 15% of all lookups require a query packet to be sent to a root or top-level domain server.

The relationship between numbers of TCP connections and numbers of DNS lookups in the MIT traces suggests that the hit rate of DNS caches inside MIT is between 70% and 80%. This includes client and application caches, including the caching done by Web browsers when they open multiple TCP connections to the same server. Thus this hit rate cannot be considered good.

Finally, the percentage of TCP connections made to names with low TTL values increased from 12% to 25% between January and December 2000, probably due to the increased deployment of DNS-based server selection for popular sites.

The captured TCP traffic helps us perform trace-driven simulations to investigate two important factors that affect caching effectiveness: (i) the TTL values on name bindings, and (ii) the degree of aggregation due to shared client caching. Our trace-driven simulations show that `A` records with 10-minute TTLs yield almost the same hit rates as substantially longer TTLs. Furthermore, the distribution of names was Zipf-like in our traces; consequently, we find that a cache shared by as few as ten clients has essentially the same hit rate as a cache shared by the full traced population of over 1000 clients. These results suggest that DNS works as well as it does despite ineffective `A`-record caching, and that the current trend towards more dynamic use of DNS (and lower TTLs) is not likely to be harmful. On the other hand, we also find that `NS`-record caching is critical to DNS scalability by reducing load on the root servers.

The rest of this paper presents our findings and substantiates these conclusions. Section II presents an overview of DNS and surveys previous work in analyzing its performance. Section III describes our traffic collection methodology and some salient features of our data. Section IV analyzes the client-perceived performance of DNS, while Section V analyzes the effectiveness of caching using trace-driven simulation. We conclude with a discussion of our findings in Section VI.

II. Background

In this section, we present an overview of DNS and survey related work.

A. DNS Overview

The design of the Internet DNS is specified in [7], [8], [9]. We summarize the important terminology and basic concepts here.

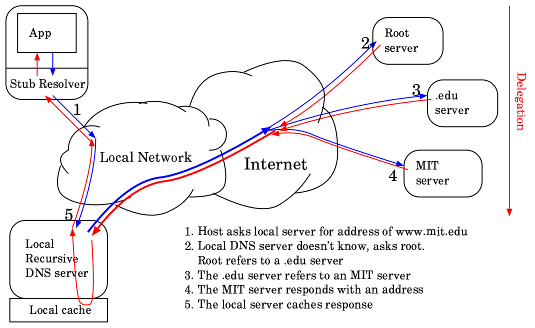
The basic function of DNS is to provide a distributed database that maps between human-readable host names (such as `chive.lcs.mit.edu`) and IP addresses (such as `18.31.0.35`). It also provides other important information about the domain or host, including reverse maps from IP addresses to host names and mail-routing information. Clients (or *resolvers*) routinely query name servers for values in the database.

The DNS name space is hierarchically organized so that sub-domains can be locally administered. The root of the hierarchy is centrally administered, and served from a collection of thirteen (in mid-2001) *root servers*. Sub-domains are *delegated* to other servers that are *authoritative* for their portion of the name space. This process may be repeated recursively.

At the beginning of our study, most of the root servers also served the top-level domains, such as `.com`. At the end, the top-level domains were largely served by a separate set of about a dozen dedicated "generic top-level domain" (gTLD) servers.

Mappings in the DNS name space are called *resource records*. Two common types of resource records are address records (`A` records) and name server records (`NS` records). An `A` record specifies a name's IP address; an `NS` record specifies the name of a DNS server that is authoritative for a name. Thus, `NS` records are used to handle delegation paths.

Since achieving good performance is an important goal of DNS, it makes extensive use of caching to reduce server load and client latency. It is believed that caches work well because DNS data changes slowly and a small amount of staleness is tolerable. On this premise, many servers are not authoritative for most data they serve, but merely cache



App
Stub Resolver
1
Local Network
Internet
2
Root server
3
.edu server
4
MIT server
5
Local Recursive DNS server
Local cache
Delegation
1. Host asks local server for address of www.mit.edu
2. Local DNS server doesn't know, asks root. Root refers to a .edu server
3. The .edu server refers to an MIT server
4. The MIT server responds with an address
5. The local server caches response

Figure 1. Example of a DNS lookup sequence.

responses and serve as local proxies for resolvers. Such proxy servers may conduct further queries on behalf of a resolver to complete a query *recursively*. Clients that make recursive queries are known as *stub resolvers* in the DNS specification. On the other hand, a query that requests only what the server knows authoritatively or out of cache is called an *iterative* query.

Figure 1 illustrates these two resolution mechanisms. The client application uses a stub resolver and queries a local nearby server for a name (say `www.mit.edu`). If this server knows absolutely nothing else, it will follow the steps in the figure to arrive at the addresses for `www.mit.edu`. Requests will begin at a well-known root of the DNS hierarchy. If the queried server has delegated responsibility for a particular name, it returns a *referral* response, which is composed of name server records. The records are the set of servers that have been delegated responsibility for the name in question. The local server will choose one of these servers and repeat its question. This process typically proceeds until a server returns an answer.

Caches in DNS are typically not size-limited since the objects being cached are small, consisting usually of no more than a hundred bytes per entry. Each resource record is expired according to the time set by the originator of the name. These expiration times are called *Time To Live (TTL)* values. Expired records must be fetched afresh from the authoritative origin server on query. The administrator of a domain can control how long the domain's records are cached, and thus how long changes will be delayed, by adjusting TTLs. Rapidly changing data will have a short TTL, trading off latency and server load for fresh data.

To avoid confusion, the remainder of this paper uses the terms "lookup," "query," "response," and "answer" in specific ways. A *lookup* refers to the entire process of translating a domain name for a client application. A *query* refers to a DNS request packet sent to a DNS server. A *response* refers to a packet sent by a DNS server in reply to a query packet. An *answer* is a response from a DNS server that terminates the lookup, by returning either the requested name-to-record mapping or a failure indication. Valid responses that are not answers must be referrals.

This means, for example, that a lookup may involve multiple query and response packets. The queries of a lookup typically ask for the same data, but from different DNS servers; all responses but the last one (the answer) are typically referrals. This distinction can be seen in Figure 1; the packets in steps 1–4 are all part of the same lookup (driven by the request from the application); however, each step represents a separate query and response.

B. Related Work

In 1992, Danzig *et al.* [1] presented measurements of DNS traffic at a root name server. Their main conclusion was that the majority of DNS traffic is caused by bugs and misconfiguration. They considered the effectiveness of DNS name caching and retransmission timeout calculation, and showed how algorithms to increase resilience led to disastrous behavior when servers failed or when certain implementation faults were triggered. Implementation issues were subsequently documented by Kumar *et al.* [10], who note that many of these problems have been fixed in more recent DNS servers. Danzig *et al.* also found that one third of wide-area DNS traffic that traversed the NSFnet was destined to one of the (at the time) seven root name servers.

In contrast to Danzig *et al.*'s work, our work focuses on analyzing client-side performance characteristics. In the process, we calculate the fraction of lookups that caused wide-area DNS packets to be sent, and the fraction that caused a root or gTLD server to be contacted.

In studies of wide-area traffic in general, DNS is often included in the traffic breakdown [2], [3]. As noted in Section I, the high ratio of DNS to TCP flows in these studies motivated our investigation of DNS performance.

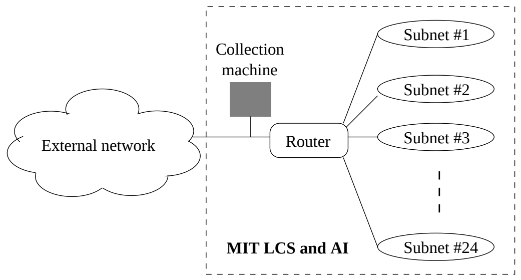
It is likely that DNS behavior is closely linked to Web traffic patterns, since most wide-area traffic is Web-related and Web connections are usually preceded by DNS lookups. One result of Web traffic studies is that the popularity distribution of Web pages is heavy-tailed [11], [12], [13]. In particular, Breslau *et al.* conclude that the Zipf-like distribution of Web requests causes low Web cache hit rates [14]. We find that the popularity distribution of DNS names is also heavy-tailed, probably as a result of the same underlying user behavior. On the other hand, DNS cache performance might be expected to differ from Web cache performance: DNS caches typically do not incur cache misses because they run out of capacity, and exclusively use TTL-based caching; DNS also caches information about each component of a hierarchical name separately (`NS` records); and many Web documents are present under a single DNS name. However, we find that DNS caches are nevertheless similar to Web caches in their overall effectiveness.

A recent study by Shaikh *et al.* shows the impact of DNS-based server selection on DNS [15]. This study finds that extremely small TTL values (on the order of seconds) are detrimental to latency, and that clients are often not close in the network topology to the name servers they use, potentially leading to sub-optimal server selection. In contrast, our study evaluates client-perceived latency as a function of the number of referrals, and analyzes the impact of TTL and sharing on cache hit rates. We also study the performance of the DNS protocol and DNS failure modes.

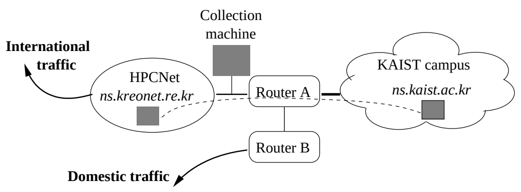
Wills and Shang studied NLANR proxy logs and found that DNS lookup time contributed more than one second to approximately 20% of retrievals for the Web objects on the home page of larger servers. They also found that 20% of DNS requests are not cached locally [16], which we consider a large percentage for the reasons explained before. Cohen and Kaplan propose proactive caching schemes to alleviate the latency overheads of synchronously requesting expired DNS records [17]; their analysis is also derived from NLANR proxy log workload. Unfortunately, proxy logs do not capture the actual DNS traffic; thus any analysis must rely on on measurements taken after the data is collected. This will not accurately reflect the network conditions at the time of the request, and the DNS records collected may also be newer. Our data allows us to directly measure the progress of the DNS lookup as it occured. Additionally, our data captures all DNS lookups and their related TCP connections, not just those associated with HTTP requests.

III. The Data

Our study is based on three separate traces. The first two were collected in January and December 2000 respectively, at the link that connects the MIT LCS and AI labs to the rest of the Internet. At the time of the study, there were 24 internal subnetworks sharing the router, used by over 500 users and over 1200 hosts. The third trace was collected in May 2001 at one of two links connecting KAIST to the rest of the Internet. At the time of data collection there were over 1000 users and 5000 hosts connected to the KAIST campus network. The trace includes only international TCP traffic; KAIST sends domestic traffic on a path that was not traced. However, the trace does include all external DNS traffic, domestic and international: the primary name server of the campus, `ns.kaist.ac.kr`, was configured to forward all DNS



(a) MIT LCS: There are 24 internal subnetworks sharing the border router.



(b) KAIST: The collection machine is located at a point that captures all DNS traffic, but only international traffic of other types. This because the primary KAIST name server, `ns.kaist.ac.kr`, forwards all DNS queries through the traced link to `ns.kreonet.re.kr`.

Figure 2. Schematic topology of the traced networks.

queries to `ns.kreonet.re.kr` along a route that allowed them to be traced. Figure 2 shows the configurations of the two networks.

The first trace, `mit-jan00`, was collected from 2:00 A.M. on 3 January 2000 to 2:00 A.M. on 10 January 2000; the second, `mit-dec00`, was collected from 6:00 P.M. on 4 December to 6:00 P.M. on 11 December 2000. The third set, `kaist-may01`, was collected at KAIST from 5:00 A.M. on 18 May to 5:00 A.M. on 24 May 2001. All times are EST.

A. *Collection Methodology*

We filtered the traffic observed at the collection point to produce a data set useful for our purposes. As many previous studies have shown, TCP traffic (and in particular, HTTP traffic) comprises the bulk of wide-area traffic [3]. TCP applications usually depend on the DNS to provide the rendezvous mechanism between the client and the server. Thus, TCP flows can be viewed as the major driving workload for DNS lookups.

In our study, we collected both the DNS traffic and its driving workload. Specifically, we collected:
1. Outgoing DNS queries and incoming responses, and
2. Outgoing TCP connection start (`SYN`) and end (`FIN` and `RST`) packets for connections *originating* inside the traced networks.

In the `mit-jan00` trace, only the first 128 bytes of each packet were collected because we were unsure of the space requirements. However, because we found that some DNS responses were longer than this, we captured entire Ethernet packets in the other traces.

The trace collection points do not capture *all* client DNS activity. Queries answered from caches inside the traced networks do not appear in the traces. Thus many of our DNS measurements reflect only those lookups that required wide-area queries to be sent. Since we correlate these with the driving workload of TCP connections, we can still draw some conclusions about overall performance and caching effectiveness.

In addition to filtering for useful information, we also eliminated information to preserve user (client) privacy. In the MIT traces, any user who wished to be excluded from the collection process was allowed to do so, based on an IP address they provided; only three hosts opted out, and were excluded from all our traces. We also did not capture packets corresponding to reverse DNS lookups (`PTR` queries) for a small number of names within MIT, once again to preserve privacy. In addition, all packets were rewritten to anonymize the source IP addresses of hosts inside the traced network. This was done in a pseudo-random fashion—each source IP address was mapped using a keyed MD5 cryptographic hash function [18] to an essentially unique, anonymized one.

Our collection software was derived from Minshall's `tcpdpriv` utility [19]. `tcpdpriv` anonymizes `libpcap`-format traces (generated by `tcpdump`'s packet capture library). It can collect traces directly or post-process them after collection using a tool such as `tcpdump` [20]. We extended `tcpdpriv` to support the anonymization scheme described above for DNS traffic.

B. *Analysis Methodology*

We analyzed the DNS traces to extract various statistics about lookups including the number of referrals involved in a typical lookup and the distribution of lookup latency. To calculate the latency in resolving a lookup, we maintained a sliding window of the lookups seen in the last sixty seconds; an entry is added for each query packet from an internal host with a DNS query ID different from any lookup in the window. When an incoming response packet is seen, the corresponding lookup is found in the window. If the response packet is an answer (as defined in Section II-A), the time difference between the original query packet and the response is the lookup latency. The

		mit-jan00	mit-dec00	kaist-may01
1	Date and place	00/01/03-10, MIT	00/12/04-11, MIT	01/05/18-24, KAIST
2	Total lookups	2,530,430	4,160,954	4,339,473
3	Unanswered	595,290 (23.5%)	946,308 (22.7%)	873,514 (20.1%)
4	Answered with success	1,627,772 (64.3%)	2,648,025 (63.6%)	1,579,852 (36.4%)
5	Answered with failure	281,855 (11.1%)	545,887 (13.1%)	1,834,942 (42.2%)
6	Zero answer	25,513 (1.0%)	20,734 (0.5%)	51,165 (1.2%)
7	Total iterative lookups	2,486,104	4,107,439	396,038
8	Answered	1,893,882	3,166,353	239,874
9	Total query packets	6,039,582	10,617,796	5,326,527
10	Distinct second level domains	58,638	84,490	78,322
11	Distinct fully-qualified names	263,984	302,032	219,144
12	Distinct internal query sources	221	265	405
13	Distinct external name servers	48,537	61,776	8,114
14	TCP connections	4,521,348	5,347,003	665,361
15	#TCP : #valid A answers, *sans* black-lists	4.62	3.53	–
16	Distinct TCP clients	992	1,233	5,754
17	Distinct TCP destinations	59,588	204,192	11,511

Table 1. Basic trace statistics. The percentages are with respect to total number of lookups in each trace.

actual end-user DNS request latency, however, is slightly longer than this, since we see packets in mid-flight. If the response is not an answer, we increment the number of referrals of the corresponding lookup by one, and wait until the final answer comes. To keep track of the name servers contacted during a lookup, we maintain a list of all the IP addresses of name servers involved in the resolution of the lookup.

This method correctly captures the list of servers contacted for iterative lookups, but not for recursive lookups. Most lookups in the MIT traces are iterative; we eliminated the small number of hosts which sent recursive lookups to name servers outside the traced network. The KAIST traces contain mostly recursive lookups sent to a forwarding server just outside the trace point; hence, while we can estimate lower bounds on name resolution latency, we cannot derive statistics on the number of referrals or the fraction of accesses to a top-level server.

C. Data Summary

Table 1 summarizes the basic characteristics of our data sets. We categorize lookups based on the DNS code in the response they elicit, as shown in rows 3-6 of Table 1. A lookup that gets a response with non-zero response code is classified as *answered with failure*, as defined in the DNS specification [8]. A *zero answer* is authoritative and indicates no error, but has no `ANSWER`, `AUTHORITY`, or `ADDITIONAL` records [10]. A zero answer can arise, for example, when an `MX` lookup is done for a name that has no `MX` record, but does have other records. A lookup is *answered with success* if it terminates with a response that has a `NOERROR` code and one or more `ANSWER` records. All other lookups are are considered *unanswered*.

	mit-jan00	mit-dec00	kaist
A	60.4%	61.5%	61.0%
PTR	24.6%	27.2%	31.0%
MX	6.8%	5.2%	2.7%
ANY	6.4%	4.6%	4.1%

Table 2. Percentage of DNS lookups across the most popular query types.

Clients can make a variety of DNS queries, to resolve hostnames to IP addresses, find reverse-mappings between IP addresses and hostnames, find mail-record bindings, etc. There are twenty query types defined in the DNS specification [8]. Table 2 lists the four most frequently requested query types in each of our traces. About 60% of all lookups were for hostname-to-address `A` records and between 24% and 31% were for the reverse `PTR` bindings from addresses to hostnames.

Although most answered `A` lookups are followed by a TCP connection to the host IP address specified in the returned resource record, there are a small number of exceptions. The most important of these are to so-called reverse black-lists, which in our traces were `A`-record lookups made to `rbl.maps.vix.com` to ask if an IP address `a.b.c.d` (in standard dotted notation) is on the list of known mail spammers maintained at `maps.vix.com`. To do this, the client sends a `A`-record lookup for the name `d.c.b.a.rbl.maps.vix.com`. If the corresponding IP address is on the list, the server answers with a `TXT` record; if it is not on the list, it sends an `NX-DOMAIN` answer. Therefore, `A`-record answers resulting from those queries do not have any associated TCP connection, and should be excluded in calculating the ratio of TCP connections to DNS lookups (in estimating caching effectiveness). We found 8,397 such lookups (0.33%) for `mit-jan00`, 6,456 (0.15%) for `mit-dec00`, and 3,226 (0.07%) for `kaist-may01`.

One of the major motivations for our work was the ratio of DNS lookups to TCP connections in the wide-area Internet, as described in Section I. The data in Table 1 (rows 4 and 14) and Table 2 (`A`-record percentage) allow us to estimate this ratio for the traced traffic, as the ratio of the number of TCP connections to the number of successfully answered lookups that are `A` records and are not on lists like `rbl.maps.vix.com`. These numbers are shown for `mit-jan00` and `mit-dec00` in row 15, suggesting a DNS cache hit ratio (for `A`-records) for the MIT traces of between 70% and 80%. As explained in Section I, this hit rate is not particularly high, since it includes the caching done by Web browsers when they open multiple connections to the same server.

The total number of successfully answered DNS lookups was over 1.5 million in the `kaist-may01` trace, but only about 0.67 million TCP connections showed up in the trace. This trace recorded *all* DNS traffic from the campus, but only the *international* TCP traffic.

IV. Client-perceived Performance

This section analyzes several aspects of client-perceived DNS performance. We start by discussing the distribution of time it took clients to obtain answers. We then discuss the behavior of the DNS retransmission protocol and the situations in which client lookups receive no answer. We also study the frequency and causes of answers that are error indications and the prevalence of negative caching. Finally, we look at interactions between clients and root/gTLD servers.

A. Latency

Figure 3 shows the cumulative DNS lookup latency distribution for our data sets. The median is 85 ms for `mit-jan00` and 97 ms for `mit-dec00`. Worst-case client latencies became substantially worse—the latency of the 90th-percentile increased from about 447 ms in `mit-jan00` to about 1176 ms in `mit-dec00`. In the `kaist-may01` data, about 35% of lookups receive responses in less than 10 ms and the median is 42 ms. The KAIST trace has more low-latency lookups than the MIT traces because the requested resource record is sometimes cached

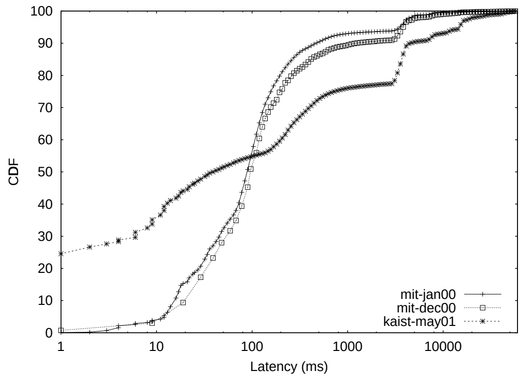


Figure 3. Cumulative distribution of DNS lookup latency.

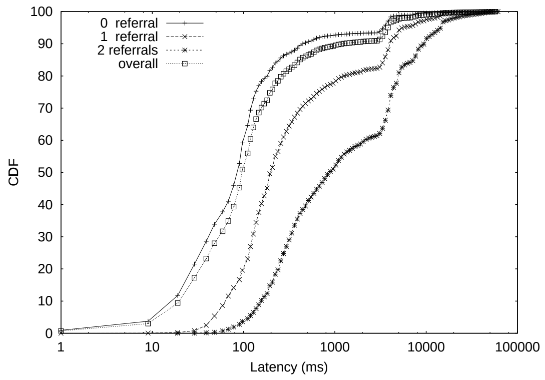


Figure 4. Latency distribution vs. number of referrals for the `mit-dec00` trace.

at `ns.kreonet.re.kr`, which is close to the primary name server for the campus (see Figure 2(b)). However, the worst 50% of the KAIST distribution is significantly worse than that of MIT. Many of these data points correspond to lookups of names outside Korea.

Latency is likely to be adversely affected by the number of referrals. Recall that a referral occurs when a server does not know the answer to a query, but does know (*i.e.,* thinks it knows) where the answer can be found. In that case, it sends a response containing one or more `NS` records, and the agent performing the lookup must send a query to one of the indicated servers. Table 3 shows the distribution of referrals per lookup. About 80% of lookups are resolved without any referral, which means they get an answer directly from the server first contacted, while only a tiny fraction (0.03%–0.04% for MIT) of lookups involve four or more referrals.

Figure 4 shows the latency distribution for different numbers of referrals for the `mit-dec00` data set. For lookups with one referral, 60% of lookups are resolved in less than 100 ms and only 7.3% of lookups take more than 1 second. However, more than 95% of lookups take more than 100 ms, and 50% take more than 1 second, if they involve two or more referrals.

	mit-jan00	mit-dec00	kaist-may01
0	74.62%	81.17%	86.09%
1	24.07%	17.86%	10.43%
2	1.16%	0.87%	2.10%
3	0.11%	0.07%	0.38%
≥ 4	0.04%	0.03%	1.00%

Table 3. Percentage of lookups involving various numbers of referrals. The number of lookups used in this analysis for each trace is shown in row 8 of Table 1. The average number of queries to obtain an answer, not counting retransmissions, was 1.27, 1.2, and 1.2, respectively.

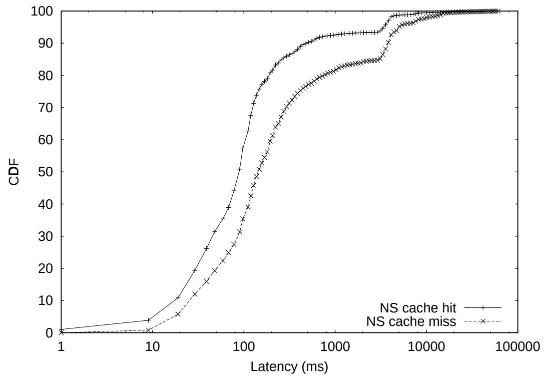


Figure 5. Distribution of latencies for lookups that do and do not involve querying root servers.

To illustrate the latency benefits of cached NS records, we classify each traced lookup as either a *hit* or a *miss* based on the first server contacted. We assume a *miss* if the first query packet is sent to one of the root or gTLD servers and elicits a referral. Otherwise, we assume that there is a *hit* for an NS record in a local DNS cache. About 70% of lookups in the MIT traces are hits in this sense. Figure 5 shows the latency distribution for each case. It shows that caching NS records substantially reduces the DNS lookup latency even though it may involve some referrals to complete the lookup. Cached NS records are especially beneficial because they greatly reduce the load on the root servers.

B. Retransmissions

This section considers lookups that result in no answer, and lookups that require retransmissions in order to elicit an answer. This is interesting because the total number of query packets is much larger than the total number of lookups; the previous section (and Table 3) show that the average number of query packets for a successfully answered query is 1.27 (mit-jan00), 1.2 (mit-jan00), and 1.2 (kaist-may01). However, the average number of DNS query packets in the wide-area per DNS lookup is substantially larger than this.

	mit-jan00	mit-dec00
Zero referrals	139,405 (5.5%)	388,276 (9.3%)
Non-zero referrals	332,609 (13.1%)	429,345 (10.3%)
Loops	123,276 (4.9%)	128,687 (3.1%)

Table 4. Unanswered lookups classified by type.

We can calculate this ratio, r, as follows. Let the total number of lookups in a trace be L, of which I are iteratively performed. This separation is useful since the trace is less likely to show all the retransmitted query packets that traverse the wide-area Internet for a recursively resolved query. We do not observe several of these query packets because a $L - I$ lookups are done recursively; however, we may observe retransmissions for recursive lookups in some cases. Let the number of query packets corresponding to retransmissions of recursive lookups be X. Let Q be the total number of query packets seen in the trace. Then, $L-I+rI = Q-X$ or $r = 1+(Q-L-X)/I$. The values of L, I, and Q for the traces are shown in rows 2, 7, and 9 of Table 1.

The value of r is relatively invariant across our traces: 2.40 for mit-jan00 ($X = 34,728$), 2.57 for mit-dec00 ($X = 18,478$), and 2.36 for kaist-may01 ($X = 448,396$). Notice that in each case r is substantially larger than the average number of query packets for a successfully answered lookup. This is because retransmissions account for a significant fraction of all DNS packets seen in the wide-area Internet.

A querying name server retransmits a query if it does not get a response from the destination name server within a timeout period. This mechanism provides some robustness to UDP packet loss or server failures. Furthermore, each retransmission is often targeted at a different name server if one exists, e.g., a secondary for the domain. Despite retransmissions and server redundancy, about 24% of lookups in the MIT traces and 20% of in the KAIST traces received neither a successful answer nor an error indication as shown in the third row of Table 1.

We break the unanswered lookups into three categories, as shown in Table 4. Lookups that elicited *zero referrals* correspond to those that did not receive even one referral in response. Lookups that elicited one or more referrals but did not lead to an eventual answer are classified as *non-zero referrals*. Finally, lookups that led to loops between name servers where the querier is referred to a set of two or more name servers forming a querying loop because of misconfigured information are classified as *loops*. We distinguish the *zero referrals* and *non-zero referrals* categories because the former allows us to isolate and understand the performance of the DNS retransmission mechanism. We do not report this data for `kaist-may01` since most lookups in that trace were recursively resolved by a forwarder outside the trace point.

The packet load caused by unanswered queries is substantial for two reasons: first, the rather persistent retransmission strategy adopted by many querying name servers, and second, referral loops between name servers.

On average, each lookup that elicited zero referrals generated about *five times* (in the `mit-dec00` trace) as many wide-area query packets before the querying name server gave up, as shown in Figure 6. This figure also shows the number of retransmissions for queries that were eventually answered (the curves at the top of the graph)—over 99.9% of the answered lookups incurred at most two retransmissions, and over 90% involved no retransmissions. What is especially disturbing is that the fraction of such wasted query packets increased substantially between January and December 2000; in the `mit-jan00` trace, the worst 5% of unanswered lookups caused 6 retransmissions, while they caused 12 retransmissions in the `mit-dec00` trace.

Given that the queries corresponding to these lookups do not elicit a response, and that most queries that do get a response get one within a small number (two or three) of retransmissions, we conclude that many DNS name servers are too persistent in their retry strategies. Our results show that it is better for them to give up sooner, after two or three retransmissions, and rely on client software to decide what to do. We believe this is appropriate because these retransmissions are being done by name servers that are recursively resolving queries for client resolver libraries that are often on different hosts that may also be retrying name requests.

Interestingly, between 12% (January 2000) and 19% (December 2000) of unanswered lookups involved no retransmission within 1 minute. This suggests an inappropriate option setting of the number of retries or an extremely large timeout value of more than 60 seconds. We believe this conclusion is correct because all these retransmissions are being done by name servers that are recursively resolving queries on behalf of clients that are on other hosts, so even if the client were to terminate the connection (e.g., because no progress was being made), the recursive name resolution by the name servers would still continue as if nothing happened.

Figure 7 shows the CDFs of the number of query packets generated for the *non-zero referrals* and *loops* categories of unanswered lookups. As expected, the non-zero referrals (which do not have loops) did not generate as many packets as the loops, which generated on average about ten query packets. Although unanswered lookups caused by loops correspond to only about 4.9% and 3.1% of all lookups, they cause a large number of query packets to be generated.

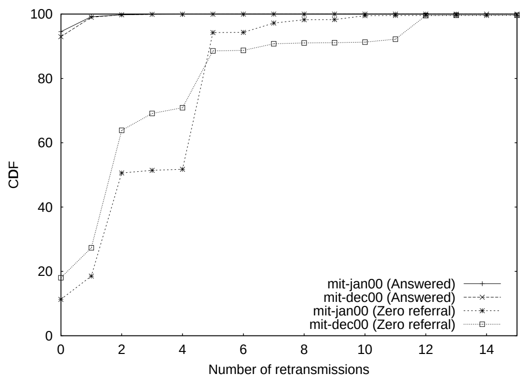


Figure 6. Cumulative distribution of number of retransmissions for answered (top-most curves) and unanswered lookups.

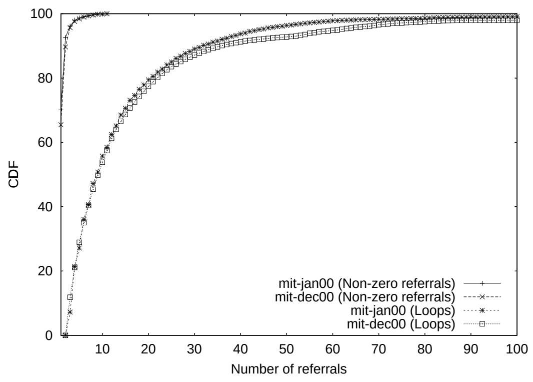


Figure 7. Too many referrals of the lookups in *loops*: this graph compares cumulative distributions of number of referrals for the lookups that cause *loops* and that cause *non-zero referrals*.

This analysis shows that a large fraction of the traced DNS packets are caused by lookups that end up receiving no response. For example, `mit-dec00` included 3,214,646 lookups that received an answer; the previous section showed that the average such lookup sends 1.2 query packets. This accounts for 3,857,575 query packets. However, Table 1 shows that the trace contains 10,617,796 query packets. This means that over 63% of the traced DNS query packets were generated by lookups that obtained no answer! The corresponding number for `mit-jan00` is 59%. Obviously, some of these were required to overcome packet losses on Internet paths. Typical average loss rates are between 5% and 10% [4], [21]; the number of redundant DNS query packets observed in our traces is substantially higher than this.

Lookups	Name
mit-jan00	
7,368	loopback
5,200	ns1.cvnet.com
3,078	jupiter.isq.pt
2,871	shark.trendnet.com.br
2,268	213.228.150.38.in-addr.arpa
1,913	mail.geekgirl.cx
1,887	swickhouse.w3.org
1,642	ns3.level3.net
1,547	mickey.payne.org
1,377	239.5.34.206.in-addr.arpa
mit-dec00	
26,308	33.79.165.208.in-addr.arpa
11,503	195.70.45.1
11,270	195.70.35.34
10,271	112.221.96.206.in-addr.arpa
9,440	104.196.168.208.in-addr.arpa
5,772	110.221.96.206.in-addr.arpa
5,290	216.4.7.226.ehomecare.com
5,235	216.4.7.227.ehomecare.com
4,137	auth01.ns.u.net
3,317	ns.corp.home.net

Table 5. The ten most common names that produced NXDOMAIN errors in the mit-jan00 trace (top) and the mit-dec00 trace (bottom). A total of 194,963 lookups produced NXDOMAIN in mit-jan00; 47,205 distinct names were involved. A total of 464,761 lookups produced NXDOMAIN in mit-dec00; 85,353 distinct names were involved.

C. Failures

As shown in Table 1, between 10% and 42% of lookups result in an answer that indicates an error. Most of these errors are either NXDOMAIN or SERVFAIL. NXDOMAIN signifies that the requested name does not exist. SERVFAIL usually indicates that a server is supposed to be authoritative for a domain, but does not have a valid copy of the database for the domain; it may also indicate that the server has run out of memory.

Table 5 shows the ten most common names that resulted in NXDOMAIN errors in the mit-jan00 and mit-dec00. The largest cause of these errors are inverse (in-addr.arpa) lookups for IP addresses with no inverse mappings.

For mit-jan00 in-addr.arpa accounted for 33,272 out of 47,205 distinct invalid names, and 79,734 of the 194,963 total NXDOMAIN responses. Similarly, for mit-dec00 in-addr.arpa accounted for 67,568 out of 85,353 distinct invalid names, and 250,867 of the 464,761 total NXDOMAIN responses. Other significant causes for NXDOMAIN responses include particular invalid names such as loopback, and NS and MX records that point to names that do not exist. However, no single name or even type of name seems to dominate these NXDOMAIN lookups.

SERVFAILs accounted for 84,906 of the answers in mit-jan00 (out of 4,762 distinct names) and 61,498 of the answers in mit-dec00 (out of 5,102 distinct names). 3,282 of the names and 24,170 of the lookups were inverse lookups in mit-jan00, while 3,651 of the names and 41,465 of the lookups were inverse lookups in mit-dec00. Most of the lookups were accounted for by a relatively small number of names, each looked up a large number of times; presumably the NS records for these names were misconfigured.

D. Negative Caching

The large number of NXDOMAIN responses suggests that negative caching may not be as widely deployed as it should be. To see how many name servers implement negative caching, we sent two consecutive queries for a non-existent name (a.b.c.com) to 53,774 distinct name servers found in the traces and recorded the response. The recursion-desired bit was set in the first query to trigger complete name resolution to the targeted name server. Then, the second query was sent without the recursion-desired bit. The idea is that if negative caching were implemented, then the second query would return NXDOMAIN, while it would return a referral if it were not implemented.

Table 6 shows the results of this experiment. The first NXDOMAIN answer implies that corresponding name servers honored the recursion-desired bit and did recursive lookups for that query. We found that 90.8% of the contacted name servers responded with NXDOMAIN for the second of the consecutive queries, suggesting that at least this fraction of the contacted name servers implemented some form of negative caching. On the other hand, 2.4% name servers returned a referral with a list of authoritative servers of .com domain with the NOERROR response code, which indicates that they were running an old version of a name server without negative caching.

The overall conclusion from this experiment is that over 90% of the contacted name servers spread across the Internet implement negative caching.

E. Interactions with Root Servers

Table 7 shows the percentage of lookups forwarded to root and gTLD servers and the percentage of lookups that resulted in an error. We observe that 15% to 18% of lookups contacted root or gTLD servers and the percentage

1st answer	2nd answer	
NXDOMAIN	NXDOMAIN	48831(90.8%)
NOERROR	NOERROR	1874 (3.5%)
NXDOMAIN	NOERROR	1272 (2.4%)
REFUSED	REFUSED	1124 (2.1%)

Table 6. Responses when two consecutive `a.b.c.com` lookups were sent to 53,774 name servers in December 2000 from MIT LCS.

	mit-jan00	mit-dec00
Root Lookups	406,321 (16%)	270,413 (6.4%)
Root Errors	59,862 (2.3%)	73,697 (1.7%)
gTLD Lookups	41,854 (1.6%)	353,295 (8.4%)
gTLD Errors	2,676 (0.1%)	16,341 (0.3%)

Table 7. The total number of lookups that contacted root and gTLD servers, and the total number of failure answers received. The percentages are of the total number of lookups in the trace.

slightly decreased between January and December 2000. This was probably caused by an increase in the popularity of popular names (see Section V and Figure 9), which decreased DNS cache miss rates. The table also shows that load on root servers has been shifted to gTLD servers over time. The gTLD servers at end of 2000 were serving more than half of all top-level domain queries.

Between 15% and 27% of the lookups sent to root name servers resulted in failure responses. Most of these appear to be mis-typed names (e.g. `prodiy.net`), bare host names (e.g. `loopback` and `electric-banana`), or other mistakes (e.g. `index.htm`). The percentage increased between January and December because much of the load of serving legitimate names moved to the gTLD servers.

V. Effectiveness of Caching

The previous sections analyzed the collected traces to characterize the actual client-perceived performance of DNS. This section explores DNS performance under a range of controlled conditions, using trace-driven simulations. The simulations focus on the following questions in the context of A-records:

1. How useful is it to share DNS caches among many client machines? The answer to this question depends on the extent to which different clients look up the same names.
2. What is the likely impact of choice of TTL on caching effectiveness? The answer to this question depends on locality of references in time.

	mit-jan00	mit-dec00	kaist-may01
Fully-qualified	0.88	0.91	0.94
Second-level	1.09	1.11	1.18

Table 8. Popularity index α for the tail of the domain name distribution.

We start by analyzing our traces to quantify two important statistics: (i) the distribution of name popularity, and (ii) the distribution of TTL values in the trace data. These determine observed cache hit rates.

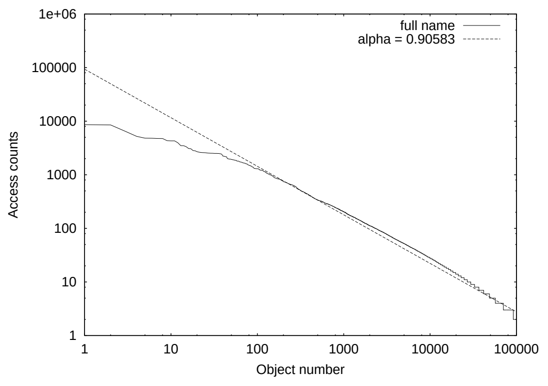
A. Name Popularity and TTL Distribution

To deduce how the popularity of names varies in our client traces, we plot access counts as a function of the popularity rank of a name, first considering only "fully qualified domain names." This graph, on a log-log scale, is shown in Figure 8(a). To understand the behavior of the tail of this distribution, and motivated by previous studies that showed that Web object popularity follows a Zipf-like distribution [11], we represent the access count as a function a/x^α, where α is termed the *popularity index*.. If this is a valid form of the tail, then a straight line fit through the tail would be a good one, and the negative of the slope would tell us what α is. This straight line is also shown in the figure, with $\alpha \approx 0.91$.

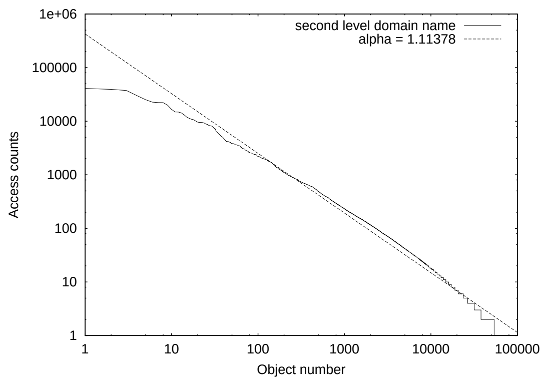
We also consider whether this tail behavior changes when names are aggregated according to their domains. Figure 8(b) shows the corresponding graph for second-level domain names, obtained by taking up to two labels separated by dots of the name; for example, `foo.bar.mydomain.com` and `foo2.bar2.mydomain.com` would both be aggregated together into the second-level domain `mydomain.com`. The α for this is closer to 1, indicating that the tail falls off a little faster than in the fully qualified case, although it is still a power-law. The slopes calculated using a least-square fit for each trace are shown in Table 8.[1]

Figure 9 illustrates the extent to which lookups are accounted for by popular names. The X-axis indicates a fraction of the most popular distinct names; the Y-axis indicates the cumulative fraction of answered lookups accounted for by the corresponding X most popular names. For example, the most popular 10% of names account for more than 68% of total answers for each of the three traces.

[1]We calculated the least-square straight line fit for all points ignoring the first hundred most popular names to more accurately see the tail behavior.



(a) Full name



(b) Second level domain name

Figure 8. Domain name popularity in the `mit-dec00` trace.

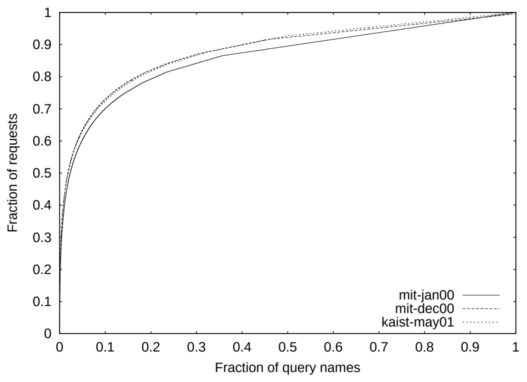


Figure 9. Cumulative fraction of requests accounted for by DNS name, most popular first. The popular names appear to have become even more popular in December 2000 compared to January 2000, although they are not necessarily the same names.

However, it also has a long tail, and a large proportion of names that are accessed precisely once. For instance, out of 302,032 distinct names involved in successful `A` lookups in `mit-dec00`, there were 138,405 unique names accessed only once, which suggests that a significant number of root queries will occur *regardless* of the caching scheme.

Figure 10 shows the cumulative distribution of TTL values for `A` and `NS` records. `NS` records tend to have much longer TTL values than `A` records. This helps explain why only about 20% of DNS responses (including both referrals and answers in Table 1) come from a root or gTLD server. If NS records had lower TTL values, essentially all of the DNS lookup traffic observed in our trace would have gone to a root or gTLD server, which would have increased the load on them by a factor of about five. Good `NS`-record caching is therefore critical to DNS scalability.

Figure 10 shows how TTL values are distributed, but

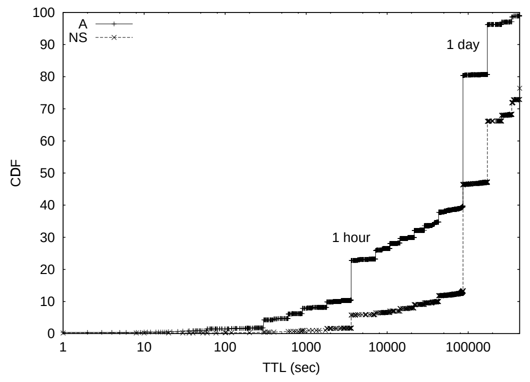


Figure 10. TTL distribution in the `mit-dec00` trace.

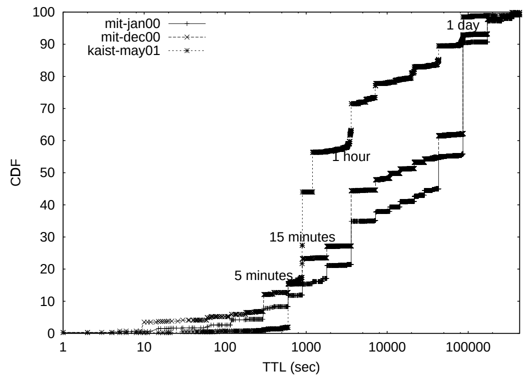


Figure 11. TTL distribution weighted by access counts for `mit-jan00`, `mit-dec00` and `kaist-may01` traces.

does not consider how frequently each name is accessed. If it turns out (as is plausible) that the more popular names have shorter TTL values, then the corresponding effect on caching would be even more pronounced. Figure 11 shows the TTL distribution of names, weighted by the fraction of TCP connections that were made to each name. We show this for both `mit-jan00` and `mit-dec00`, and draw two key conclusions from this. First, it is indeed the case that shorter-TTL names are more frequently accessed, which is consistent with the observation that DNS-based load-balancing (the typical reason for low TTL values) makes sense only for popular sites. Second, the fraction of accesses to relatively short (sub-15 minute) TTL values has doubled (from 12% to 25%) in 2000 from our site, probably because of the increased deployment of DNS-based server selection and content distribution techniques during 2000.

B. Trace-driven Simulation Algorithm

To determine the relative benefits of per-client and shared DNS caching of A-records, we conducted a trace driven simulation of cache behavior under different aggregation conditions. First, we pre-processed the DNS answers in the trace to form two databases. The "name database" maps every IP address appearing in an A answer to the domain name in the corresponding lookup. The "TTL database" maps each domain name to the highest TTL appearing in an A record for that name. After building these databases, the following steps were used for each simulation run.

1. Randomly divide the TCP clients appearing in the trace into groups of size s. Give each group its own simulated shared DNS cache, as if the group shared a single forwarding DNS server. The simulated cache is indexed by domain name, and contains the (remaining) TTL for that cached name.
2. For each new TCP connection in the trace, determine which client is involved by looking at the "inside" IP address; let that client's group be g. Use the outside IP address to index into the name database to find the domain name n that the client would have looked up before making the TCP connection.
3. If n exists in g's cache, and the cached TTL has not expired, record a "hit." Otherwise, record a "miss."
4. On a miss, make an entry in g's cache for n, and copy the TTL from the TTL database into the n's cache entry.

At the end of each simulation run, the hit rate is the number of hits divided by the total number of queries.

This simulation algorithm is driven by the IP addresses observed in the traced TCP connections, rather than domain names, because DNS queries that hit in local caches do not appear in the traces. This approach suffers from the weakness that multiple domain names may map to the same IP address, as sometimes occurs at Web hosting sites. This may cause the simulations to overestimate the DNS hit rate. The simulation also assumes that each client belongs to a single caching group, which may not be true if a client uses multiple local forwarding DNS servers. However, because DNS clients typically query servers in a strictly sequential order, this may be a reasonable assumption to make.

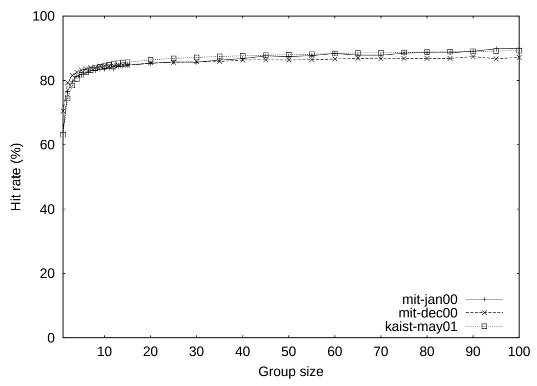


Figure 12. Effect of the number of clients sharing a cache on cache hit rate.

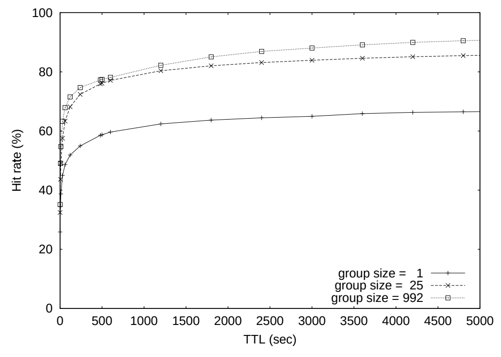
C. Effect of Sharing on Hit Rate

Figure 12 shows the hit rates obtained from the simulation, for a range of different caching group sizes. Each data point is the average of four independent simulation runs. With a group size of 1 client (no sharing), the average per-connection cache hit rate is 64% for `mit-jan00`. At the opposite extreme, if all 992 traced clients share a single cache, the average hit rate is 91% for `mit-jan00`. However, most of the benefits of sharing are obtained with as few as 10 or 20 clients per cache.

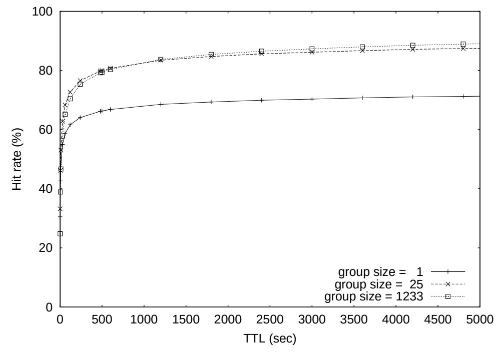
The fact that domain name popularity has a Zipf-like distribution explains these results. A small number of names are very popular, and even small degrees of cache sharing can take advantage of this. However, the remaining names are large in number but are each of interest to only a tiny fraction of clients. Thus very large numbers of clients are required before it is likely that two of them would wish to look up the same unpopular name within its TTL interval. Most cacheable references to these names are likely to be sequential references from the same client, which are easily captured with per-client caches or even the per-application caches often found in Web browsers.

D. Impact of TTL on Hit Rate

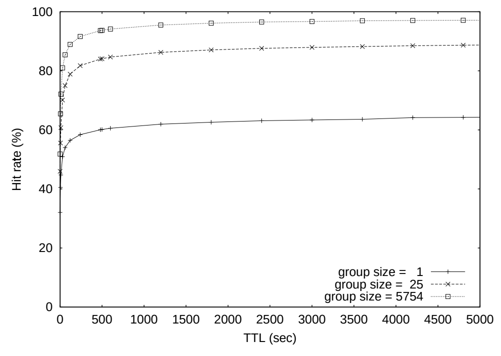
The TTL values in DNS records affect cache rates by limiting the opportunities for reusing cache entries. If a name's TTL is shorter than the typical inter-reference interval for that name, caching will not work for that name. Once a name's TTL is significantly longer than the inter-reference interval, multiple references are likely to hit in the cache before the TTL expires. The relevant interval depends on the name's popularity: popular names will likely be cached effectively even with short TTL values, while unpopular names may not benefit from caching even with



(a) `mit-jan00`



(b) `mit-dec00`



(c) `kaist-may01`

Figure 13. Impact of TTL on hit rate.

very long TTL values. In turn, a name's popularity among a group of clients that share a cache is to some extent determined by the number of clients in the group.

To gauge the effect of TTL on DNS cache hit rate, we perform simulations using a small modification to the algorithm described in Section V-B. Instead of using TTL values taken from the actual DNS responses in the traces, these simulations set all TTL values to specific values. Figure 13 shows the results, with one graph for each of the three traces. Each graph shows the hit rate as a function of TTL. Since the results depend on the number of clients that share a cache, each graph includes separate curves for per-client caches, groups of 25 clients per cache, and one cache shared among all clients. We use a group of size 25 because Section V-C showed that for the actual TTL distribution observed in our traces, a group size of 25 achieves essentially the same hit-rate as the entire client population aggregated together.

As expected, increasing TTL values yield increasing hit rates. However, the effect on the hit rate is noticeable only for TTL values less than about 1000 seconds. Most of the benefit of caching is achieved with TTL values of only a small number of minutes. This is because most cache hits are produced by single clients looking up the same server multiple times in quick succession, a pattern probably produced by Web browsers loading multiple objects from the same page or users viewing multiple pages from the same Web site.

These results suggest that giving low TTL values to `A` records will not significantly harm hit rates. Thus, for example, the increasingly common practice of using low TTL values in DNS-based server selection probably does not affect hit rates much.

In general, caching of `A` records appears to have limited effectiveness in practice and in potential. Even eliminating *all* `A`-record caching would increase wide-area DNS traffic by at most a factor of four, almost none of which would hit a root or gTLD server. Eliminating all but per-client caching would little more than double DNS traffic. This evidence favors recent shifts towards more dynamic (and less cacheable) uses of DNS, such as mobile host location tracking and sophisticated DNS-based server selection.

We conclude this section by noting that we do not suggest that it is a good idea to make the TTL values low on `NS`-records. On the contrary, doing so would increase the load on the root and gTLD servers by about a factor of five and significantly harm DNS scalability.

VI. Conclusion

This paper presented a detailed analysis of traces of DNS and associated TCP traffic collected on the Internet links of the MIT Laboratory for Computer Science and the Korea Advanced Institute of Science and Technology. We analyzed the client-perceived performance of DNS, including the latency to receive answers, the performance of the DNS protocol, the prevalence of failures and errors, and the interactions with root/gTLD servers. We conducted trace-driven simulations to study the effectiveness of DNS caching as a function of TTL and degree of cache sharing.

A significant fraction of lookups never receive an answer. For instance, in the most recent MIT trace, 23% of lookups receive no answer; these lookups account for more than half of all traced DNS packets in the wide-area since they are retransmitted quite persistently. In addition, about 13% of all lookups result in an answer that indicates a failure. Many of these failures appear to be caused by missing inverse (IP-to-name) mappings or NS records that point to non-existent or inappropriate hosts. We also found that over a quarter of the queries sent to the root name servers result in such failures.

Our trace-driven simulations yield two findings. First, reducing the TTLs of address (A) records to as low as a few hundred seconds has little adverse effect on hit rates. Second, little benefit is obtained from sharing a forwarding DNS cache among more than 10 or 20 clients. These results suggest that the performance of DNS is not as dependent on aggressive caching as is commonly believed, and that the widespread use of dynamic, low-TTL A-record bindings should not degrade DNS performance. The reasons for the scalability of DNS are due less to the hierarchical design of its name space or good A-record caching (as seems to be widely believed), than to the cacheability of NS-records that efficiently partition the name space and avoid overloading any single name server in the Internet.

Acknowledgments

We are grateful to Garrett Wollman and Mary Ann Ladd, who run the network infrastructure at MIT LCS, for facilitating this collection process and for their prompt help at various times. We thank Dorothy Curtis for help with data collection. This study would not have been possible without the cooperation of the LCS and AI Lab communities at MIT. We are also grateful to Heonkyu Park and Taejin Yoon who helped collect traces at KAIST. We also thank Sarah Ahmed and David Andersen for several useful discussions, and Nick Feamster for useful comments.

References

[1] P. Danzig, K. Obraczka, and A. Kumar, "An Analysis of Wide-Are Name Server Traffic: A Study of the Internet Domain Name System," in *Proc ACM SIGCOMM*, Baltimore, MD, Aug. 1992, pp. 281–292.

[2] K. Frazer, "NSFNET: A Partnership for High-Speed Networking," `http://www.merit.edu/merit/archive/nsfnet/final.report/`, 1995.

[3] K. Thompson, G. Miller, and R. Wilder, "Wide-area Traffic Patterns and Characteristics," *IEEE Network*, vol. 11, no. 6, pp. 10–23, November/December 1997.

[4] H. Balakrishnan, V. Padmanabhan, S. Seshan, M. Stemm, and R. Katz, "TCP Behavior of a Busy Web Server: Analysis and Improvements," in *Proc. IEEE INFOCOM*, San Francisco, CA, Mar. 1998, vol. 1, pp. 252–262.

[5] A. Snoeren and H. Balakrishnan, "An End-to-End Approach to Host Mobility," in *Proc. 6th ACM MOBICOM*, Boston, MA, Aug. 2000, pp. 155–166.

[6] J. Loughney and J. Yu, *Roaming Support with DNS*, July 2000, Internet-Draft draft-loughney-enum-roaming-00.txt; expires January 2001. Available from `http://www.ietf.org/internet-drafts/draft-loughney-enum-roaming-00.txt`.

[7] P. Mockapetris, *Domain Names—Concepts and Facilities*, Nov. 1987, RFC 1034.

[8] P. Mockapetris, *Domain names—Implementation and Specification*, Nov. 1987, RFC 1035.

[9] P. Mockapetris and K. Dunlap, "Development of the Domain Name System," in *Proc. ACM SIGCOMM*, Stanford, CA, 1988, pp. 123–133.

[10] A. Kumar, J. Postel, C. Neuman, P. Danzig, and S. Miller, *Common DNS Implementation Errors and Suggested Fixes*, Oct. 1993, RFC 1536.

[11] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "On the implications of Zipf's law for web caching," Technical Report CS-TR-1998-1371, University of Wisconsin, Madison, Apr. 1998.

[12] M. Crovella and A. Bestavros, "Self-similarity in World Wide Web Traffic: Evidence and Possible Causes," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 835–846, Dec. 1997.

[13] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. Levy, "On the scale and performance of cooperative web proxy caching," in *17th ACM SOSP*, Kiawah Island, SC, Dec. 1999, pp. 16–31.

[14] S. Glassman, "A Caching Relay for the World-Wide Web," *Computer Networks and ISDN Systems*, vol. 27, pp. 165–173, 1994, `http://www.research.compaq.com/SRC/personal/steveg/CachingTheWeb/paper.html`.

[15] A. Shaikh, R. Tewari, and M. Agrawal, "On the Effectiveness of DNS-based Server Selection," in *Proc. IEEE INFOCOM*, April 2001.

[16] C. Wills and H. Shang, "The Contribution of DNS Lookup Costs to Web Object Retrieval," Tech. Rep. TR-00-12, Worcester Polytechnic Institute, July 2000, `http://www.cs.wpi.edu/~cew/papers/tr00-12.ps.gz`.

[17] E. Cohen and H. Kaplan, "Proactive Caching of DNS Records: Addressing a Performance Bottleneck," in *Proc. Symp. on Applications and the Internet (SAINT)*, San Diego, CA, January 2001.

[18] R. Rivest, *The MD5 Message-Digest Algorithm*, April 1992, RFC 1321.

[19] G. Minshall, "Tcpdpriv," `http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html`, Aug. 1997.

[20] V. Jacobson, C. Leres, and S. McCanne, "tcpdump," `http://www.tcpdump.org/`.

[21] V. Paxson, "End-to-End Internet Packet Dynamics," in *Proc. ACM SIGCOMM*, Cannes, France, Sept. 1997, pp. 139–152.