A Study of Caching in the Internet Domain Name System

by

Emil Sit

S.B. (Mathematics), Massachusetts Institute of Technology (1999)
S.B. (EECS), Massachusetts Institute of Technology (1999)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2000

Author . . . .
Department of Electrical Engineering and Computer Science
May 12, 2000

Certified by . . . .
Robert Morris
Asst. Professor, Laboratory for Computer Science
Thesis Supervisor

Certified by . . . .
Hari Balakrishnan
Asst. Professor, Laboratory for Computer Science
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

A Study of Caching in the Internet Domain Name System

by

Emil Sit

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 2000, in partial fulfillment of the
requirements for the degree of
Master of Engineering

Abstract

The Internet Domain Name System (DNS) is a distributed, hierarchical and flexible system for the mapping of names to matching resource records. Caching was designed into the DNS to improve response time and reduce wide-area network traffic. This thesis describes a study of a series of traces taken of DNS-related traffic at MIT's Laboratory of Computer Science. A macroscopic overview of the traffic types is presented as well as a more in-depth look at the actual performance of caching in the system.

We find that an DNS traffic is a very small percentage of the local network traffic. Caching appears to be 90% effective in answering queries out of cache. We examine the effects of aggregation and DNS time-to-lives on cache performance. Trace-based simulations allow us to conclude that the current levels of client aggregation and time-to-live values on DNS records work well for cache performance; increasing either would only give limited improvement. This work additionally provides a basis for future research in this area.

Thesis Supervisor: Robert Morris
Title: Asst. Professor, Laboratory for Computer Science

Thesis Supervisor: Hari Balakrishnan
Title: Asst. Professor, Laboratory for Computer Science

Acknowledgements Many people were instrumental in making this possible. Garrett Wollman was extremely helpful in making data on the MIT LCS network available. David Maziéres gave some insightful comments on protecting user privacy and how to better collect needed data. Tony Chao provided the original ideas for some of the data analysis when this was just a term project for a graduate networking class taught by Prof. Hari Balakrishnan. Chuck Blake was of invaluable assistance in coming up with real algorithms and data structures to efficiently analyze the data that was collected. In various discussions, many other members of the MIT and LCS community were helpful in focusing this work.

No electrons were harmed during the production of this paper. However, some trees were brutally exploited.

Contents

Chapter 1

Introduction

The Internet Domain Name System (DNS) is a distributed, hierarchical and flexible naming system which provides for the mapping of names to resource records. In particular, this system is used to map familiar hostnames to their corresponding Internet Protocol (IP) addresses [18]. Such a naming system provides two distinct benefits. First, a level of indirection allows administrators to assign specific physical resources to logical names. For example, when people refer to a name such as `www.mit.edu`, the naming system can be configured to refer to any machine that is a web server for `mit.edu`. This mapping could even be dynamically adjusted to provide a form of load balancing. Second, logical names are preferred by humans since they are much easier to remember than numeric addresses such as `18.181.0.31`. Because of these two features, the DNS plays a role in the establishment of almost every network connection across the Internet. A large number of different records are managed under this system and it must therefore be highly scalable in order to be successful.

Scalability of the DNS is handled in several ways. First, management is hierarchical and distributed. Second, replication and caching help reduce load on any single server. The architecture of DNS is such that no single organization is responsible for storing the entirety of the mapping between names and resource records. Rather, the namespace is broken up hierarchically and sub-trees are recursively delegated into manageable portions. In the end, a client asking for the value of a name must ask a server that has been appropriately designated as *authoritative* for the portion of the namespace (or *domain*) to which the name belongs. The process of searching the namespace for a particular mapping is called name *resolution.*

It is possible, and indeed likely, that the particular server responsible for a name's binding may be located far from the client performing the query. Because users often make network connections to hosts far from their local domains, there is necessarily some latency prior to the establishment of any network connection causeed by the name resolution process. Therefore, clients generally cache the results of past queries for use in future resolutions. Caches hold these results for an amount of time that is set by each authoritative server for each record. This time-out is called a *time-to-live* (or TTL). Since caching also obviates the need to contact servers, it also reduces the load on those servers that might otherwise be involved in the resolution.

A key research question is whether DNS caching is necessary and effective given today's workload. The actual effects of caching in reducing resolution latency and network traffic has not yet been extensively studied. While caches on processors often benefit from locality of reference, the workload of a DNS cache may be highly variable and not able to benefit from temporal locality at all. The goal of this thesis was to estimate the actual usefulness of caching as currently deployed in the Domain Name System under today's complex workloads. In particular, this thesis investigated the actual hit rate of DNS caches under a real workload and simulated the effects of different levels of client aggregation and different default TTL distributions.

We found that the apparent hit rate of DNS caches is high, at somewhere between 80–95%. Under simulation, we found that aggregation of clients using a single server tends to increase overall cache effectiveness but the usefulness falls off quickly beyond clusters of twenty to twenty-five clients. Finally, short TTLs negatively affect cache performance as might be expected, but again the improvement from increasing the TTL of a record is not useful beyond a certain point.

The rest of this chapter will present a more detailed introduction to the Domain Name System and then give the overall goals of the thesis. The chapter concludes with a survey of the related work in this field. Chapter 2 presents the methodology used in this study. The results of the analysis are then presented in chapter 3. Finally, our conclusions and suggestions for future work are summarized in chapter 4.

1.1 Overview of DNS

An excellent repository of information about the DNS is maintained online at the DNS Resources Directory [19]. This section describes the parts of the DNS relevant to this thesis. We begin with an overall view of the system's architecture. Within the context of this architecture, a detailed description of the resolution mechanism is given. We close with a discussion of caching. Figure 1-1 summarizes the various points described in these sections.

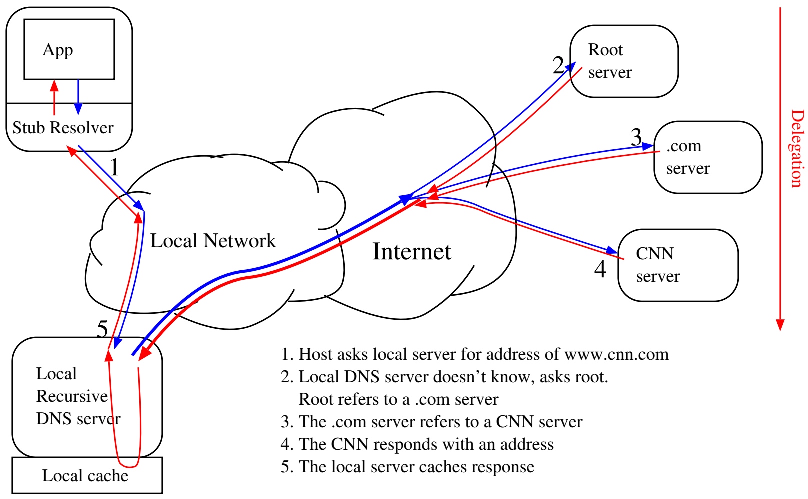

Figure 1-1: DNS Architecture

1.1.1 Overall design

The abstract model of the DNS is that of a database that maps human-readable names to records. Broadly, the namespace is divided into classes and then further into record types. Some allowed record types include addresses, aliases, nameserver pointers, inverse pointers. Several application specific types exist as well. For each record type, a given name may have multiple simultaneous mappings. This provides a very general purpose naming system that can be used for many purposes. However, for all practical purposes, all records fall into a single class, namely class `IN`, or the Internet class. This thesis considers mostly name-to-address mappings (known as `A` records).

The DNS implements this abstract model by storing the name bindings in a distributed fashion. Names take the form *e.d.c.b.a.* and responsibility for the namespace is divided hierarchically into domains and sub-domains. Segments of the name that are further to the right are higher in the hierarchy. Servers are then designated as *authoritative* for a sub-domain of the root. These servers can then choose to *delegate* further sub-portions of their domain to other servers. Any of these servers may be replicated to improve reliability. The set of names controlled by any particular server is called a *zone*.

For example, the root is nominally responsible for the name `wind.lcs.mit.edu.` However, since the root cannot maintain a list of all names, it delegates authority for that name to a server responsible for all `edu` names. This server in turn delegates control of `mit.edu` names to some servers controlled by MIT. Since LCS is a separate sub-domain, MIT has delegated control of it to an LCS internal server. That server is the authoritative source for information regarding the name `wind.lcs.mit.edu`.

1.1.2 Queries

Client software called a *resolver*s are used to query the DNS namespace. A resolver typically contains some amount of *glue* which allows it to find a server authoritative for the root of the namespace. When presented with a name to resolve, the resolver can ask a root server which will then *refer* the resolver to the appropriate server which has been delegated authority for the particular name in the query. The resolver is then responsible for following the *referral*s until the answer to the query has been found.

For example, suppose a resolver wishes to look up `wind.lcs.mit.edu.` Knowing nothing at all about the delegation, the resolver must ask the root server. The root will not know the answer, but it will know the authoritative server for `edu` (and possibly even `mit.edu`). The resolver must then ask the `mit.edu` name server, which returns the addresses of the authoritative servers for `lcs.mit.edu`. It is an LCS server that will return the actual answer to the resolver.

This style of resolution is known as an *iterative* lookup: the resolver iterates by repeating its request until it reaches a server that knows the answer. Client hosts, however, typically do not implement this full functionality. Instead, they only implement what is called a *stub resolver*. These resolvers know about a small fixed set of local servers to which it forwards all of its queries. By setting a *recursion desired* bit in its request, the stub resolver asks the local server to resolve the name on its behalf. Upon receiving such a query, these shared servers then either forward the query on to another server, or perform the iterative look up themselves. The stub resolver therefore performs a *recursive* lookup.

Figure 1-1 illustrates these two resolution mechanisms. The client application uses a stub resolver and queries a local nearby server for a name (say `www.cnn.com`). If this server knows absolutely nothing else, it will follow the steps in the figure to arrive at the addresses for `www.cnn.com`. Many machines use shared servers to offload the work of implementing and operating a full resolver.

1.1.3 Caching

The system described above would suffer from extreme load at the central server and long latencies at the clients who would have to incur multiple network round-trips before getting an answer. The standard solution to reduce such problems is to employ caching. In fact, caching has been an integral part of the DNS since its description in RFCs 1034 and 1035 [13, 14]. In general, caches exploit the idea of locality — since it is often the case that the set of items recently accessed will be accessed again in the near future, we can benefit by storing these items where we can get at them faster. Like caches in other systems, DNS caches also hope to exploit this locality of reference and relieve the problems of load and latency. This would benefit both servers and clients of the system.

Consistency. In any distributed data system, maintaining cache consistency is a difficult problem. The design of the Domain Name System sacrifices consistency in favor of improved access time. However, the source of any given record is allowed to control the amount of consistency sacrificed by trading off with latency. This is handled by allowing a zone administrator to specfiy a *time-to-live* (TTL) for each record. A record can then only be cached for the duration of the TTL. Upon expiration of this TTL (measured in real time), a cache must expunge the record. Thus, if a binding with TTL l is changed, the old value should remain active in the network until l seconds after the change took effect on the authoritative server. Implementations must be careful to correctly manage the TTL so as not to violate these semantics. For example, when a forwarder answers a query out of cache, it must set the TTL on the response to the current decremented value so that an answer is not valid beyond the time specified by the authoritative server when it was originally sent.

Ideally, the TTL would be very long for names with fairly static bindings and shorter for names which might change often. When a change is known to be pending, the ideal strategy would be for the source to lower the TTL of the name in advance of the actual change. After the change, the TTL could be restored to some reasonably large value. This would allow continued service and caching but restrict the amount of time in which stale data is kept in caches. However, very few people handle DNS name changes this way. Instead, names are generally just changed, while the TTL is held unchanged. Also, these values are generally fairly short and much less than the rate of change of the actual binding.

Replacement policy. Time-to-live values also function to keep caches from growing unboundedly, thus obviating the need for any specific replacement policy. In general, DNS caches do not limit the cache size (as recommended by RFC 1536 [11]). Also, caching is usually a manageable prospect since DNS records are reasonably small and memory is fairly cheap.

Multi-level caching. Caching of DNS records is permitted at multiple levels. First, any machine running a full resolver typically has a cache. This may be a local workstation or a shared server. Shared servers may also be chained to provide increasingly larger client sets. This idea of a cache hierarchy is similar to those deployed for web proxy caching. Because there is no real limit on the size of these caches, increased sharing improves cache performance when the machines sharing the cache also have overlap in the sets of names accessed.

In addition to resolver caching, there are also occasional application level caches. For example, web browsers like Internet Explorer and Netscape both do application level caching of names. However, since the library calls that do name resolution do not provide actual TTL information, these applications typically cache the results of all lookups for some fixed period of time. For example, most versions of Netscape use a strict timeout of fifteen (15) minutes, regardless of the actual TTL set by the server.

Cache misses. We can categorize the cache misses experienced by a collection of machines using the DNS into three types:

- *Compulsory* or *cold* misses — the very first time a name is accessed at a resolver, there is no way for the name to be in the cache. Thus, these misses are unavoidable.

- *Expiration* misses — records are removed from caches when their lease (time-to-live) expires. Thus, many non-compulsory misses occur because the TTL on a record has expired.

- *Sharing* misses — as discussed above, aggregation hopes to increase the effective locality of records that are resolved. In some cases, machines were not sufficiently aggregated and thus a request results in a cache miss that could have been answered by another cache.

1.1.4 Resolver context

One important problem of naming is determing the correct context in which to resolve a name. In most cases there is a search path which provides a series of default contexts to try. RFC 1535 [9] discusses a potential security flaw and a proposed correction in the search paths used by DNS resolvers. Prior to this proposal, resolvers would typically attempt to resolve names that are not fully qualified by tacking on the decreasing sub-names of the domain name of the host doing the query.

For example, at some company (say, in the domain `example.com`, one might type the name `www` and expect that the search path would provide `www.example.com`. An unfortunate side effect of this is that resolving the name `www.mit.edu` would first result in checking for `www.mit.edu.example.com`. When this fails, the resolver would then check `www.mit.edu.com`. A malicious person could exploit this behavior by registering the domain `edu.com` and placing appropriate names in that domain to point to his own machines. To solve this problem, RFC 1535 recommended that if a resolver implements an implicit search list, the scope of the list should be limited to local sub-domains. Any additional search lists outside of the local sub-domain must be explicitly specified by the user, and no default explicit search lists should be specified by the resolver software.

One interest effect of having a search path is that the resolution of a name provided by the user may require serveral searches before the correct context is found. These search path misses can result in excess network traffic. As a result, it will also increase the latency experienced by clients due to the increased number of network round-trips needed to resolve the name.

This effect is partially mitigated by a heuristic that causes resolvers to attempt to resolve a name with no additional context if it is presented with a dot in it. In other words, a query for `www` will result in the explicit behavior described above. However, a search for `www.us` will check for the presence of the name "`www.us.`" before using the search path to consider `www.us.example.com`. It is unknown what fraction of resolvers implement this behavior.

1.2 Project Goals

The primary goal for this project was to establish an understanding of the how well DNS caching performs in the real world. This goal is interesting for at least two reasons. First, very few studies have focused explicitly on this important part of the Internet's architecture. Second, the Internet has grown tremendously since the DNS was designed and its needs have changed since then. No longer are all the old assumptions about the namespace true — instead, there are mobile hosts, dynamically load-balanced bindings, and new extensions introduce larger and more complicated records.

Fundamentally, the DNS performance is affected by caching. A baseline system just like DNS but without caching would almost definitely incur noticeable network delays. In order to precisely assess the performance of the DNS, we focus on three hypotheses:

- The hit rate on DNS caches is very high.
- Aggregation of queries allows for improved cache hit rate.
- Longer TTLs would improve cache hit rate.

We now discuss each of these in turn.

1.2.1 Cache hit rate

Hit rate is generally the fundamental measure of the success of a cache. We would like to know that the implementation of the DNS today is giving us a good hit rate on caches. We will discuss several metrics for measuring this. Based on empirical metrics, we estimate that the actual hit rate is approximately 95%.

In addition to the actual hit rate, we would like to understand more specifically how and why we achieve certain hit rates. This can be done by examining the causes of misses.

1.2.2 Cache aggregation

The utility of aggregating clients has been well-studied in the area of web proxy caches (see discussion in section 1.3). One might reasonably suspect that similar results could be obtained for DNS workloads — however, it is unknown whether the DNS workload would lead to similar results. DNS also has a much better defined caching policy since all records are cachable and explicit cache timeouts exist on all records. Thus, we wanted to study how much aggregation currently exists between DNS clients and how much aggregation affects the overall hit rate. Many DNS clients also employ application level caching and this caching may actually have more impact than large aggregated caches.

Based on simulations of cache behavior driven by trace data taken at LCS, we believe that aggregation mostly has the effect of equalizing the cache hit rates between different caches. Cache hit rates quickly converge to roughly 80% as aggregation increases.

1.2.3 Impact of TTL settings

The performance of a cache is driven both by its workload and the elements that are present in the cache. In DNS caches, the contents of the cache are affected both by the workload and the lease times given for different records. Due to the desire for flexible DNS mappings, zone administrators have been using relatively short TTLs. However, many people believe that low TTLs would drastically decrease DNS cache performance and recommend setting TTLs on the order of one day.

We find that most TTLs are set near one day in length, but many are also shorter. Based on simulation results, we find that variation in the length of TTLs does not have a significant impact until it falls below two hours. However, this fall-off is fairly gentle until they fall below two minutes.

1.3 Related Work

1.3.1 Known statistics

The impetus for this thesis stemmed partially from a study of the DNS done in 1988 by Danzig *et al* [7]. Their analysis concluded that over 8% of packets and 4% of bytes on the NSFnet were due to DNS traffic. However, much of this was wasteful — during a 24 hour trace, DNS consumed roughly twenty times the amount of bandwidth than optimally necessary. The paper suggests a couple of problems with DNS but the most noticeable ones were a result of implementation errors in resolvers. These included:

- Incorrect handling of referrals in recursive queries. When a server can not answer a query, it will often return a referral to a server which it think can authoritatively answer the question. Sometimes, however, the follow-up server would issue a referral to the first. This would create loops, causing traffic without any useful result.

- Broken retransmission timing. Determining good back-off parameters is somewhat tricky and many resolvers were observed sending more packets than necessary because they did not wait long enough for a reply. A query would sometimes be retransmitted too quickly or to too many servers.

- No detection of server failure. Many resolvers would not correctly detect that a server had gone down; as a result, it would continue to retransmit its query to the server indefinitely.

Other factors contributing to DNS traffic include leaky caches and lack of negative caching. The paper concluded that as new versions of resolvers which correct many of the above implementation errors get deployed, some of these problems will disappear. However, since it is unlikely that a bug-free Internet will ever arise, they argued that DNS servers should be implemented to detect and correct the known resolver flaws.

Work by Thompson *et al* [21] provide additional affirmation on the significance of DNS related traffic in a wide area network. Their paper takes a very close look at the traffic on two of MCI's trunks over a one-week period in May 1997. They observed that DNS load tended to follow the link utilization, but traffic comprised 2–5% of packets and 1–2% of bytes but as much as 15–20% of flows on the links studied. The source of the traffic was mostly server-to-server, with only 10% of the traffic being between clients and servers. This last statistic makes sense since most clients are configured to send requests to local servers which then make wide-area queries.

This thesis hopes in part to extend on some of these results.

1.3.2 Aggregation and web caches

One area of extremely active research has been in assessing the performance of web caches. Two representative papers are [8, 22]. We make the hypothesis that DNS cache behavior will probably be somewhat similar to the observed behavior of web caches; this is plausible since the web is a major portion of TCP connections and DNS queries are largely driven by the need to find an IP address with which to make a TCP connection.

In general, cooperating web caches (i.e. increased aggregation) does improve the hit rate, but not beyond a certain point. In detailed comparisons, [22] concludes that aggregation is very useful when group sizes are still relatively small but it soon levels off so that increased size only leads to very small improvements in overall hit rate.

Our results show that DNS caches exhibit similar behavior.

1.3.3 Selecting good TTLs

The TTLs in the DNS system are very similar to the concept of leases. Leases were introduced in [10] where they are presented in the context of filesystems. While the paper deals largely with the problem of maintaining consistency in the face of writes, the model and arguments presented is of some relevance in deciding on TTLs for DNS hostnames. In particular, it makes the simple argument that files that are written often should have lower leases, whereas more static files can have longer leases.

In the specific context of the Domain Name System, RFC 1537 [6] makes the argument that timers in the DNS are generally set "(far) too low" and that "this causes unnecessary traffic over international and intercontinental links." Obviously, the choice of time-to-live is a problem but like [10], the RFC only offers rough guidelines. In this thesis, the effects of specific TTL settings is made clear.

Chapter 2

Methodology

In this chapter, we first describe and explore the details of the data collection process. Second, we outline the techniques used in our analysis.

2.1 Data Collection

To answer the questions proposed in section 1.2, it is necessary to observe DNS behavior at a very fine grained level. Ideally, one would observe queries as they orginated from clients and follow them as they got resolved. This would present a complete view of the system, from the contents of all messages to the latencies involved in at each level. This basic methodology was used by Barford *et al.* in a study of web traffic at Boston University [4].

Unfortunately, it would be very difficult to implement this sort of instrumentation in a broad study of the DNS for two key reasons:

- A larger number of hosts would need to be studied. Servers would need to be instrumented to record this data as well.

- At the time of the BU study, the only viable web browser was Mosiac. Since Mosaic was open source, this made making changes easy. Now, there are a large number of proprietary operating systems and web browser combinations deployed.

These factors make instrumentation essentially unfeasible for collecting any significant corpus of data.

We can avoid this problem by collecting network traces. By placing traffic collectors at certain well-chosen locations, one can capture a broad spectrum of client behavior. Of course, this is necessarily less precise than any data that could be obtained through instrumentation. Nonetheless, trace collection is the only data acquisition method used in this thesis.

2.1.1 Trace criteria

The architecture and protocols of the Internet DNS are independent of the network transport that it is deployed upon. However, most DNS traffic uses the User Datagram Protocol (UDP) [17], a connectionless protocol built on top of IP. This is because queries and replies generally fit into single packets. By using UDP, DNS avoids the overhead of a streaming, connection-oriented protocol such as TCP. For large transfers, DNS implementations fall-back to using TCP.

This study focuses exclusively on UDP DNS traffic. On the assumption that most packets are fairly short, we chose to collect the first 128-bytes of all UDP DNS traffic. This assumption later turned out to be incorrect.

To fully understand the behavior of the DNS, it is necessary to have an understanding of the workload that drives it. Since TCP is the primary transport for much Internet traffic (such as web browsing or file transfer), packets related to TCP connection maintenance were also collected. In particular, TCP packets indicating the start and end of connections were captured. Of course, these are not the sole driver of DNS traffic — many applications use different transports. However, bulk non-TCP transports are uncommon, non-standard and undocumented. The exact load of these transports is not known or considered.

2.1.2 Privacy

In the ideal case, traces would collect raw packets as they appeared over the wire to provide maximum flexibility in later analysis. However, user privacy is a critical issue to consider. On the basis of DNS lookups alone, it is possible to construct very good guesses about the behavior of users. For example, if Ben Bitdiddle was surfing over to `www.playboy.com`, he would first need to discover the IP address of Playboy's web server. A DNS trace would indicate that a DNS request for `www.playboy.com` is originating from a specific address. Since today it is often the case that computers have only one regular user, reasonable guesses can be made as to who is browsing what web pages (and when). TCP connection information is even more explicit since they signal intent to transfer data.

Thus, it was very important to ensure that the privacy of users on the network we were monitoring. To protect against the kinds of analysis described above, we rewrote the IP headers to hide the source and destination addresses. This allows the sending and receiving hosts can be partially anonymized easily and uniquely. Each address was replaced with the low-order bytes of its MD5 hash. This hash was salted with a random 8-byte string to make brute force guessing of the original address more difficult. Since, only the low-order bytes of the MD5 hash are used, the probability of collision is increased, roughly from 1 in 2^{128} to 1 in 2^{32}.

An additional goal was to be able to determine whether an address belonged to MIT, even after rewriting. This was motivated by the fact that internal traffic is presumably lower cost than traffic which goes off-net. In particular, DNS queries that stay within MITnet are not quite as bad as those which must contact root and other authoritative servers. The above address rewriting scheme was also augmented to optionally map all MIT network addresses from `18.0.0.0/8` into `10.0.0.0/8`. This further increases the probability of collision slightly. In hindsight, this scheme could have been further adapted to preserve additional topology (e.g. subnetting) information to allow for reasoning about machines as groups easier.

2.1.3 Collection software and hardware

The simplest way to collect this data was to hook up a machine to a connection and collect both the raw DNS queries and TCP connection initiation packets. In large-scale network traffic collection endeavors, special software and hardware are occasionally needed — one example might be the CoralReef project at CAIDA [2], which was used in [21]. For this project, it turned out that the links we could monitor were much lower bandwidth (less than 100Mbps).

A tool developed to anonymize tcpdump traces called tcpdpriv [12] was used to actual dump network packets from the interface card. This tool makes use of the `libpcap` packet capture library [3] to collect and store packets. It was necessary to extend tcpdpriv to support the anonymization scheme described above.

The collection machine was an Intel PPro200 with 128 MB RAM running Linux 2.2.10. The network card was a 100 Mbit Tulip Ethernet card. Data was dumped onto a 27GB IBM UltraIDE disk.

Time period	5 days
Distinct DNS clients	1,281
Distinct DNS servers	50,275
Query packets seen	5,486,669
Response packets seen	5,094,351
Distinct TCP clients	$\approx$ 1,700
Distinct TCP servers	45,938
TCP connections	2,345,783

Table 2.1: High-level Statistics for the LCS `week1` trace.

2.1.4 Trace description

Our data was collected on the border router of the MIT Laboratory for Computer Science (LCS). The LCS network is large subnetwork of the main MIT network. The overall network infrastructure at LCS is controlled by the LCS Computer Resource Services (CRS) group; many research groups use the network, each of which has a different networking configuration. The CRS group set up a read-only feed of all incoming and outgoing traffic to LCS to the data collection system described above. The attachment point is shown in figure 2-1.

Traces were collected from 7 December 1999 to 9 January 2000. Despite the large amount of data collected, most of the analysis in this thesis is done on the 5 days of DNS traces taken from 3 January 2000 to 7 January 2000 (inclusive). This is the first week where we might expect regular behavior from the network since the rest of the data was collected either during vacation time or finals week at MIT. Limiting the data set size also made it easier to process. For the rest of this thesis, this trace will be referred to as the LCS `week1` trace. High-level aggregate statistics for this sub-trace are presented in table 2.1.

Because this attachment point had access to all outgoing LCS traffic, it provided the ability to gain a very clear picture of the client behavior that drives DNS queries. To take that picture, the following information was recorded:

- Complete outgoing DNS requests and their responses.
- TCP connection maintenance packets.

Note that this trace contains only a subset of the total traffic sent over the link. As a result, any conclusions drawn based on the traffic counts are only a lower-bound estimate

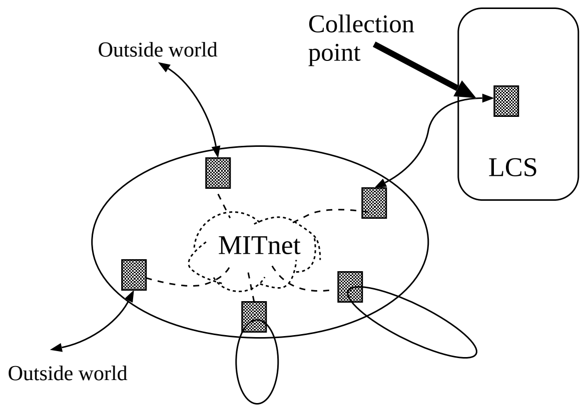

Figure 2-1: LCS Trace Collection setup

— incoming queries (and their outgoing replies) have an unknown impact on the overall distribution.

All IP addresses that originated from LCS were remapped as described in section 2.1.2, other IP addresses were left alone. Only packets related to outgoing DNS requests and outgoing TCP connections were collected. This gives a good picture of client behavior while still preserving the privacy of users. To further assure privacy, all packets from anyone who wished to opt-out of the collection were ignored. Finally, people who wanted to completely obfuscate the origins of their requests were suggest to use caching DNS servers and the LCS WWW caching proxy server.

2.2 Analysis Methodology

The collected data was analyzed in two ways. The first type of analysis involved direct analysis of the collected data. The second type used the collected data as a basis for simulation. In this section we describe the methods used for both of these analyses.

2.2.1 Direct analysis

Statistics were generated by post-processing the `libpcap`-format trace files. While there exist programs that already do some analysis of these trace files (e.g. `tcptrace` [16]), these programs are very general and do not perform very well on the multi-hundred megabyte trace files collected. Thus, custom C programs were written to summarize the various statistics presented in this thesis. The specific algorithms used vary and are described in chapter 3. Implementing the algorithms and appropriate data structures was one of the more time consuming (and educational) parts of this thesis. Performance was a major goal and to that end, a few general points are worth noting here:

- State is expensive and should be minimized — in general, aggregate handling of many trace files was difficult any time per query state was kept, due to the large number of different queries seen over time. This would lead to thrashing in the virtual memory system (even on machines with 256 MB of RAM) and extremely poor performance.

- Custom post-processed file formats can greatly speed repeated processing by essentially eliminating parsing costs. This greatly reduces the size of the data and also allows it be held in the buffer cache in Linux, which saves even more time by avoiding disk seek overhead.

- Designing data structures to be page-aligned allows for both more efficient and more effective use of memory.

2.2.2 Simulation

In several cases, it was useful to have an idea of the performance of the DNS under conditions different from those in which data was collected. We used trace-driven simulations to understand these conditions. The exact simulation algorithms are described in the relevant sections of chapter 3.

For some simulations, a database of TTLs for different records was useful. To avoid having to write a separate database system, the freely available SleepyCat Berkeley Database [15] was used. The SleepyCat system is easy to use and provides easy persistent storage of the database which allows re-use.

Chapter 3

Analysis

In this chapter, we present the technical analysis needed to determine the answers to the questions posed in section 1.2.

Fundamentally, we are concerned with the overall performance of the various DNS caches deployed on the Internet. Thus our analysis is focused on measuring this.

3.1 A Macroscopic View

Very few recent statistics are available about DNS traffic. This section describes some overall patterns of the traffic we see. We will find some of these numbers useful in our estimations later on.

3.1.1 General statistics

The amount of DNS traffic in LCS traces generally follows the standard diurnal patterns. For the `week1` trace, we can see this in figure 3-1. Though the actual levels vary widely, there are rough peaks in the middle of each day.

The basic statistics about DNS traffic is summarized in table 3.1. The total packet rate is for all packets (both incoming and outgoing) but all the other rates reflect only the outgoing subset of the total traffic transiting the LCS border. Nonetheless, we can see that egress DNS-related packets make up less than one percent of the total packets.

The query packet sizes are fairly consistent, with only a very small standard deviation. This result is reasonable since the variation in query packet sizes is primarily due to variation in the length of the name being looked up — names, therefore, do not vary significantly

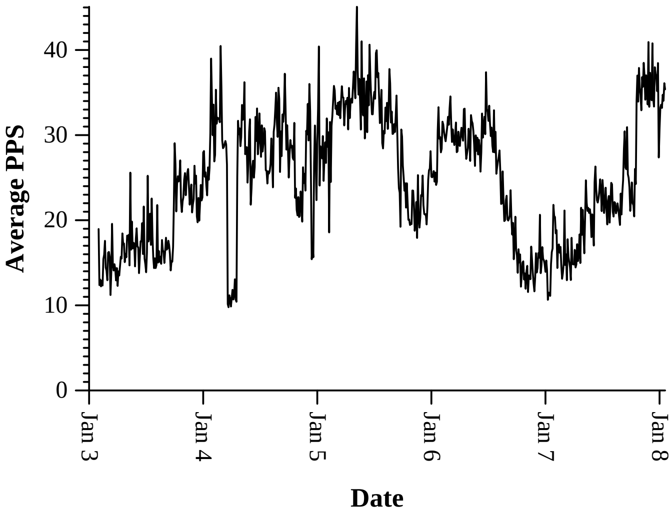

Figure 3-1: DNS Packet Rates for `week1` trace. Dates indicate midnight.

Average total packets/sec	≈ 9000	
Average new TCP connections/sec	10.14	
Average DNS packets/sec	25.94	(≈ 0.2%)
Average query packet size (bytes)	39.9	±6.2
Average response packet size (bytes)	160.8	±66.5
Internal to root+com queries	531,614	(10.4%)
Authoritative responses	1,815,123	(35.6%)
Referrals	3,005,402	(59.0%)

Table 3.1: Traffic statistics over the LCS `week1` trace

in length. On the other hand, response packets vary widely in size. In contrast to queries, responses tend to have widely varying contents — some packets merely contain a single record in answer to the query, but other packets may contain multiple records including references to name server and other additional information which were relevant to the query made. As a result we see a large standard deviation in the response packet size.

This information can also be useful in estimating the best packet capture size. Factoring in overhead for link, network and transport level headers, a packet capture size of 256 bytes should be used if capturing entire packets is desired. On the other hand, the ordering of data in the DNS packet is such that the answer section is in first so shorter capture lengths will still contain useful information.

On average, there are roughly two to three times as many DNS packets per second than new outgoing TCP connections. We will attempt to explain this in the next section. One possibile explanation is that the averages have been skewed by abnormal hosts.

3.1.2 Understanding the client population

As soon as some statistics were calculated, it become clear that there were certain hosts on the network which behaved in extremely non-average ways. In fact, in many cases where we were testing for particular conditions or actions, we found that a single host would generate between a third and a half of the total number of matches would be generated by one or two hosts. For the purposes of the statistics in this paper, we will be ignoring these hosts.

3.1.3 TTL distribution

In this chapter, we will often be using address records obtained from the traces themselves for simulation purposes. This gives a more accurate view of the state of the DNS than doing lookups at the time of simulation. It is also much more efficient. Figure 3-2 shows the distribution of TTLs associated with address records in the `week1` trace. These TTLs were collected by examining *all* DNS response packets in the trace and extracting the highest TTL for any particular IP address. This will likely be the TTL when the record is obtained from the authoritative server. Most TTLs are somewhere between six hours and one day in length, which is fairly reasonable [6].

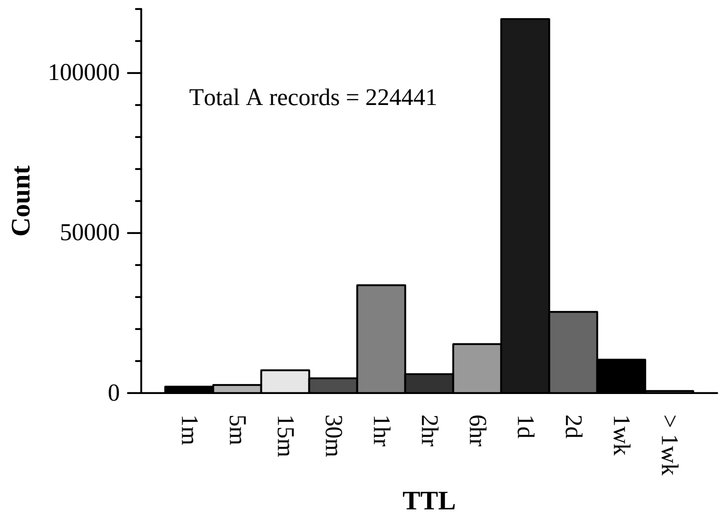

Figure 3-2: Distribution of address record TTLs

3.2 Measuring the Hit Rate

The most basic question to ask about a cache is how many of its accesses it can handle without having to request an answer from its backing store. In this section we try and estimate the actual hit rates experienced by caches in our study.

3.2.1 Hit rates from TCP connections

In some sense, every DNS query represents a cache miss. In the traces collected, each outbound query represents a query that could not be satisfied purely by the internal caching hierarchy used by a particular client. However, this does not give us an idea of how often DNS caches *succeed* in satisfying questions purely internally. In order to estimate this, we use the idea that TCP connections are the workload that drive DNS queries. In the

Protocol	Hit Rate
SMTP (Mail)	97.52%
Web	91.34%
Ident	93.15%
Login (ssh/telnet)	80.62%
Overall	95.68%

Table 3.2: Hit rates based on DNS responses that are correlated with TCP connections. Outlying hosts are ignored.

worst case, one would have a new DNS query for each time an outbound TCP conection is made. To estimate the hit rate, we count the number of TCP sessions whose start is closely preceded by a DNS response packet containing the destination of the TCP connection in its answer section. A four second sliding window was used to match DNS responses to TCP connections. Looking at the ratio of connections for which this is false to those which it is true gives us an estimate of the hit rate. Using this metric, we obtain the results shown in table 3.2 from looking at the `week1` trace.

For each of those misses we want some estimate of many DNS packets are involved. To estimate this, we look at the number of referral packets that we see over the network. Referrals happen when a server does not know the answer to a question, but *does* know where the answer can be found — in that case, it sends a response packet with this information in it. The typical case for this is a name that is delegated but for which none of the delegation information has been cached.

Approximately two-thirds of the DNS responses in a typical trace are referrals. This roughly implies that a typical DNS query takes three requests to resolve a typical name. However, the number of referrals is not necessarily equally distributed across queries as show in figure 3-3. The graph shows the ratio of referrals to queries sent on a log-log scale.

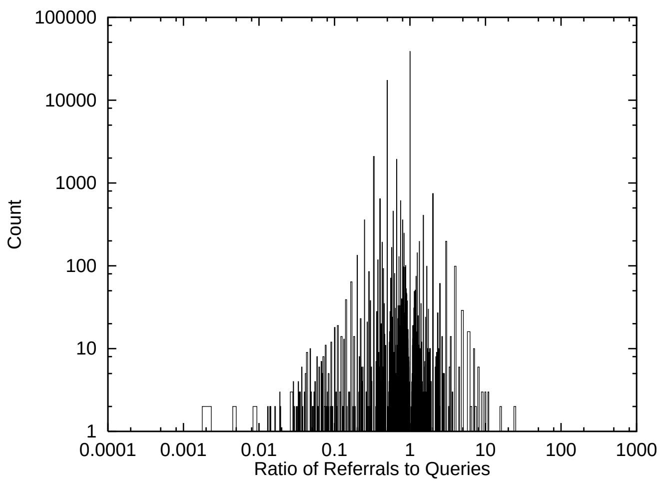

Figure 3-3: Distribution of referral ratios for LCS `week1` trace.

There are two main peaks — those that require one referral to resolve and those that require zero. This indicates that for the most part, queries require very few referrals to look up hosts. However, there are some names which receive more referrals than queries. These are likely to be error cases where servers are either down or misconfigured.

3.2.2 Evaluating hit rates

The LCS hit rates appear to tell us that on average, one in ten TCP connections to web servers requires a DNS resolution. In order to understand whether or not these results are actually good, we compare these results to known results for web proxy caches. As argued in section 1.3.2, it is plausible to believe that DNS cache behavior should be correlated in some way with web cache behavior.

It is generally accepted that web object popularity obeys a power law relationship such that the n-th most popular page has a frequency proportional to n^{-1} [5]. The absolute best case web cache performance seems to give approximately 70–80% hit rates [22]. Thus our observed hit rate of 91.34% seems to be much better. Most likely, DNS caches achieve hits when web caches would not since different web objects may be requested from the same host.

3.3 Cache Aggregation

One might hypothesize that cache hit rates could be improved by aggregating hosts. By aggregating hosts to use a single cache, one client might get a hit on an entry loaded into the cache by another client. This section discusses this concept and attempts to measure its actual usefulness.

3.3.1 Current levels

The first step in understanding our hit rates and the effects of aggregation is to examine the actual patterns of aggregation at LCS. This can be determined from the LCS trace data via an extension of the method used to estimate actual hit rate. In particular, a client is presumed to use a particular server s if a TCP connection originates from the client to some IP i "soon" after a DNS response containing an address record pointing to i is sent to s. By iterating over traces, an approximation of the actual clustering used by clients over the trace period.

The results of running this on the `week1` trace gives the numbers shown in figure 3-4. The graph displays first the number of servers that each distinct client uses. It also shows

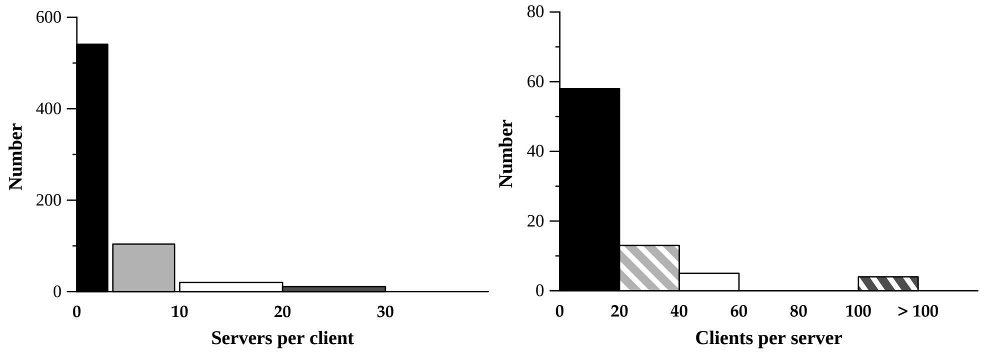

Figure 3-4: Client and Server aggregation at LCS

the number of clients that each distinct server appears to have. For clarity, these exact numbers are binned.

The vast majority of clients use a small number of servers — usually they use three or fewer different servers. This is shown in the tall (truncated) thin bar in the graph. On the other hand, there are some clients that appear to use more than twenty servers over the course of the week. These are probably dialup machines and laptops using DHCP — by using IPs as identifiers, we collapse different physical machines which may be resuing the same IP address. However, this has not been directly verified because of the IP masking used to anonymize the data.

Most servers have small groups of approximately 20 clients, though there are three servers that have a very large number of clients (reaching up to 400 each). This effect is likely due to the fact that each research group within LCS has a caching name server in addition to using a set of central servers which are available to the entire lab.

3.3.2 Simulated behavior for various aggregation levels

To estimate the effects of aggregation, a driver was written to take a TCP connection workload and simulate cache behavior given different parameters. We discuss how aggregation levels affected the results.

To begin, the DNS results from traces were combined to form a database of stock TTLs to be used for particular address records. This allows the simulator to provide realistic TTLs for the different addresses being contacted. The clients involved in the TCP workload were then divided into groups randomly and each group was assigned a single cache.

The TCP workload was then processed in the following manner. For each outgoing connection, let c be the client and i be the destination IP address.

1. Identify what group c is in and access the cache for that group.
2. If i is in the cache...
   - ...and *has not* expired due to the time-to-live, then this is a hit. There would have been no external DNS lookup caused by this TCP connection.
   - ...and *has* expired, this is an *expiration* miss. Record the cache miss and refresh the lifetime of the record.
3. If it is not in the cache then we have *compulsory* miss. Look it up in the TTL database and store it in the cache until the TTL expires.

At the end of the trace, we have a record of the number of total accesses for the group and the number of hits in the cache. Additionally, we have classified the misses seen.

This simulation methodology has a few weaknesses. First, it assumes that each host belongs only to a single group. While obviously not the case, this assumption may be acceptable in the sense that clients often query DNS servers in order, only trying other known servers when the primary server fails. While we do not have hard evidence to support this, the aggregation study above shows that most of the TCP clients actually appear to use only one server.

Second, this algorithm does caching based solely on destination IP address. Typically, TCP connections are driven by a request for a *name* and not an address. Thus, this algorithm makes the implicit assumption that the name to IP address mapping is essentially 1-1. In reality, a host may have two completely different names with the same IP and have different TTLs on them. There is also no guarantee that client would have contacted this exact IP (in the case of a record set where any of the entries would have sufficed). However, since TTLs for a binding are typically the same if they have the same label and type, the use of the IP address in place of the name probably has very little impact on the actual results.

This simulation was run through the gamut of different aggregation levels. At the lowest extreme, each host does solely local caching and there is no sharing done at all between caches. This case is of interest since it may be the case that the *cost* of deploying shared caching does not outweigh the benefits gained from improved hit rate. Above this level, we simulated groups of different sizes up to the case where all hosts were sharing a single large cache.

For each aggregation size, the 965 hosts involved in the trace were divided into random groups and cache performance based on the (static) workload were simulated. This process was repeated four times and the results averaged. The results of this are shown in figure 3-5 The ranges marked indicate the minimum and maximum hit rates seen over the four trial

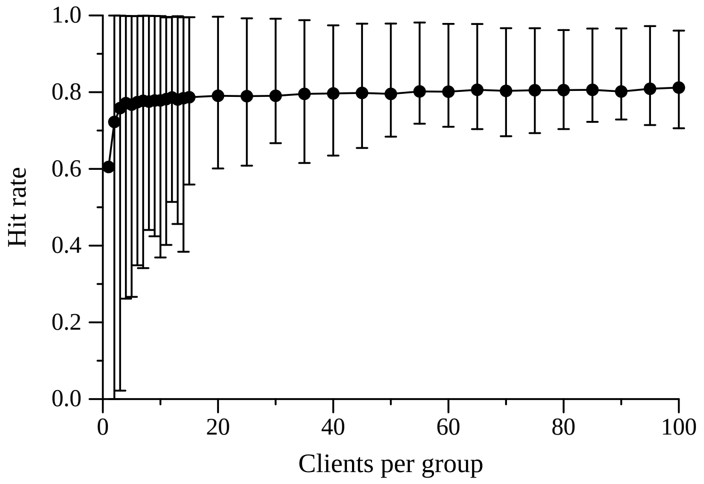

Figure 3-5: Cache hit rates under simulation. Verticial bars indicate range of hit rates.

runs. As can be seen, aggregation *does* give an improved hit rate over no aggregation, but this quickly peaks at approximately 80%. Worst-case cache performance definitely improves as well, though at a slightly slower rate, leveling out after approximately twenty hosts are in a group. This is almost exactly the level of aggregation that observed in the system. Of course, overall performance does increase slightly as we continue to aggregate. Though not shown on the graph, we can achieve a maximum hit rate of 84.9% if all hosts share a single cache. On the other hand, the average hit rate for the no-aggregation case is 60.5%.

3.3.3 Application caching: hit or miss?

To get an understanding of where cache hits actually occur, the simulator described above was extended to estimate the amount of hits that could have been handled by application-level caching. Application caches are simply a higher-level optimization; in reference to figure 1-1 the client would have its own local cache that would use the local network cache as essentially an L2 cache. Because there is no guarantee that the local host resolver does any caching, applications wishing to avoid the cost of even local network messages will do caching at the application-level. Popular examples of applications that do this are web-browsers such as Netscape and Internet Explorer. Thus, a cache hit could occur at many levels — the application, host resolver, a local recursive DNS server or even perhaps further down in a query chain. We would like to know how many of the hits occur at the application (or host) level as opposed to the number that occur at local recursive servers or other forwarders.

Unfortunately, this effect is somewhat difficult to quantify exactly since we do not actually have access to client applications and caches. We can only make deductions based on queries that leave the network and the TCP connections that are made in the meantime. One trick that we can take advantage of is that we know how long browsers cache records. We simulated the cache behavior of a two-level cache hierarchy against the `week1` trace.

Under simulation, each TCP connection first attempts to see if there is a record for the host in the local cache — this cache attempts to simulate an application cache that holds all entries for fixed period (e.g. fifteen minutes). If the host is not in this cache, we look at a group cache; these groups have been randomly assigned, just as in the simulation described above. This cache operates as normal and stores entries for the actual duration of the TTL specified by the server.

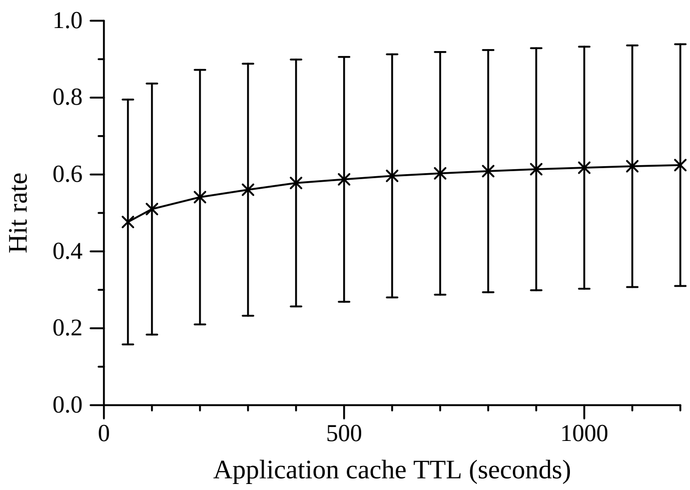

Figure 3-6: Average hit-rate for application caches with fixed DNS TTLs. Vertical bars indicate one standard deviation.

Figure 3-6 shows the average hit rate at each host's application cache as a function of fixed TTL used in the application cache. Each group contained (roughly) twenty-five hosts. Note that here the vertical bars here indicate the standard deviation instead of the range — in all cases, the range varied roughly from zero to one.

It is clear that increasing the fixed TTL does improve the local hit rate (as might be expected) but the average rate is very low, at around 60%. The standard deviation remains fairly constant, giving a large variation of about 30% in the performance of individual caches. Note in particular that for a fixed TTL of 900 seconds (fifteen minutes, the Netscape default), we have a hit rate of 61.4%. Thus, browser caching alone does not give very good performance, with easily one out of two TCP connections (on average) requiring a DNS lookup. On the other hand, this is not worse than using the correct TTLs — the hit rate in both cases is roughly 60%. This is likely to be true because most TTLs are sufficiently long so that the fixed TTL is usually smaller than the actual TTL. It is only the case of names that use very short TTLs that may result in stale data being stored and used from the cache.

In other words, even though fixed TTL application is technically a violation of the DNS caching policy, it does not significantly affect the actual performance of the cache so long as the fixed TTL is not too short.

3.3.4 Evaluating aggregation

Based on these simulation results, we can conclude that aggregation is a useful technique for improving performance of our DNS caches. By having a relatively small number of workstations together use a single cache, we can achieve much better hit rates than single host caching. However, by doing additional caching at the local host, we can satisfy many of the requests at an even lower cost, with only a small fraction of local accesses requiring a query to the local caching proxy.

3.4 Caching and Time-to-Live Settings

Naturally, caching is deeply intertwined with the choice of time-to-live settings. In this section, we explore the relation between TTL settings and cache behavior. The following results are based on different simulation results.

3.4.1 Expiration misses

Ideally, we would know exactly how many actual queries were the result of expired TTLs in a local cache within LCS. However, this is extremely difficult to measure.

In simulation of caches under aggregation, we know exactly which requests could be answered out of cache correctly, which requests were for previously seen hosts that have expired from the cache and which hosts are completely new to the cache. On average, it appears that 58.7% of the misses are for previously seen hosts whose TTLs have expired. This suggests that longer TTLs may be useful.

Running this against the TTL simulations (below) makes this very clear — shorter TTL cap means that a higher percentage of misses are caused by TTLs. This is not very surprising.

3.4.2 Varying TTL limits in simulation

In this section we explore some theoretical cases of how well caching would perform with different TTLs, given the current work-load of TCP connections.

Many records from popular domains have low TTLs. This may be for any number of reasons. One simple but likely common reason is that the domain administrator might not have thought to set a long TTL [6]. In modern systems requiring high availability and performance, short TTLs can be used to balance load between servers (by changing the mapping to point to the least loaded server) or for fast fail-over in the case of a server failure. For example, Akamai [1], a company which manages content-delivery, uses very short TTLs (on the order of 30 seconds) to accomplish very-fine grain load balancing and react quickly to changes in demand and network performance. In even more extreme cases, people have proposed using zero second TTLs for highly volatile name bindings [20].

We ran several simulations to estimate the effects of low TTLs on the cache hit rate. As in the browser cache case, all simulations were run with a group size of twenty-five. This simulation ran as normal using a group server (with no browser caching) except that all TTL values were capped at some fixed short value. The resulting hit rates are shown in figure 3-7 as a function of this capping TTL.The results are somewhat as one might expect. Obviously if the TTL is zero, the hit rate will be zero. However, caching for merely one minute greatly increases the overall hit rate. When caching for 120 seconds we see an

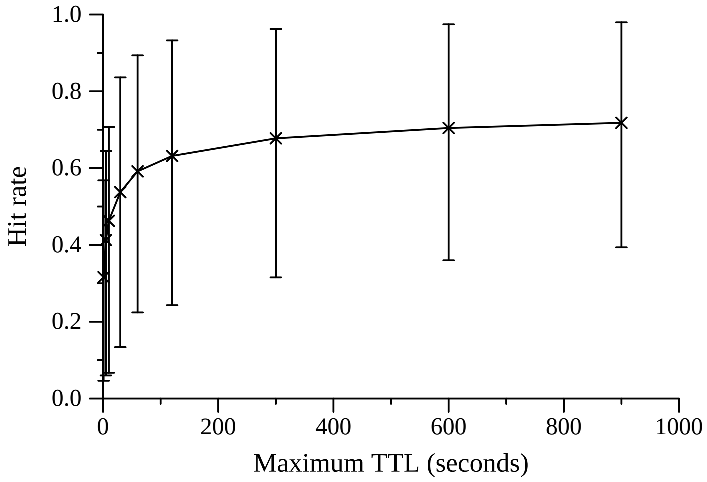

Figure 3-7: Average hit-rate when cache TTL is capped.

average hit rate of 63.4% which is comparable to browser-only caching. As we increase the capping TTL, the hit rate continues to rise, as we would expect. At a cap of two hours, we have an average hit rate of 76.2%. This trend continues until we begin to approach the normal 80% hit rate.

Chapter 4

Conclusions and Future Work

4.1 Summary of Results

We have presented an overview of DNS traffic and performance based on a traces collected at MIT's Laboratory for Computer Science. At this local level, we examined the characteristics of DNS traffic. We found that the network traffic load due to DNS is rather small relative to the total network traffic. Despite an overall packet rate of nine-thousand packets per second, we still only see an average of about thirty DNS packets per second (related to outgoing queries). This level of local traffic is quite mangeable, even if caches were not performing at their best.

On the other hand, we do find that approximately 95% of TCP connections did not require new DNS lookups. For the connections that do cause cache misses, we find that the majority of DNS queries can be resolved with one to two referrals, at most. Thus, caches in the real world generally have cached information about the most commonly used parts of the hierarchy down to the individual domains.

Certain specific factors affecting cache performance were tested via simulation. Like in web proxy caching, we found that increased aggregation is useful to a point, but not significantly useful beyond that limit. The ideal cache aggregation size appears to be around twenty hosts, beyond which there is only a very small gain in hit rate.

We found that longer TTLs improve the hit rate, as might be expected. However, this hit-rate does not significantly improve when varied above two minutes. This is of interest to those who may want to change their address bindings frequently. Use of TTLs below two minutes will give rise to a dramatically higher query rate to servers. Thus we recommend based on our simulation results a minimum TTL of 120 seconds. For more stable bindings, a minimum of at least four hours should be used.

4.2 Future Work

There are a number of ways in which this work could be extended. First, the traces analyzed in this study were limited to a single location over a relatively short time-period. It would be extremely useful to repeat some of these analyses on traces collected from different locations. By performing these tests on other traces, we could have stronger confidence in the validity of the results. Ideally, we could collect finer grain detail by taking traces at the attachment points of smaller client networks, especially those without local caching name servers. Such additional data would be useful in confirming or clarifying our results.

The simulation work was trace-driven, which is a fairly common method for handling simulation. However, there do exist realistic traffic generators which would allow for more flexibility in the kinds of traffic studied. For example, the SURGE system developed at Boston University [5] produces traffic patterns that closely match the characteristics of measured traces.

In addition to directly extending the work performed here, there are several other related areas that are of interest:

- DNS request latency — understanding the distribution of latencies involved in answering queries would give a more concrete notion of the actual cost of a cache miss.

- Negative caching — by doing negative caching, the DNS could conceivably reduce wide-area traffic for repeated questions with negative answers. However, the importance of this is somewhat unclear. An interesting research problem would be to determine just how useful negative caching would be.

- Name server load — caching also serves to reduce load on servers. A more complete study might try and estimate the load imposed on various servers under different caching conditions.

4.3 Conclusion

Overall, we see that DNS caching is actually performing reasonably well. While there are still some pathological hosts, for the most part hosts are well-behaved and can complete resolutions quickly. We have presented some concrete analysis of real trace data and simulation results that allowed us to make specific recommendations on how to improve cache hit rate. Finally, we have developed the tools and basic understanding necessary to continue in this direction.

Bibliography

[1] Akamai technologies, inc. `http://www.akamai.com/`.

[2] CoralReef software suite. `http://www.caida.org/Tools/CoralReef/`.

[3] Tcpdump home page. `http://www.tcpdump.org/`.

[4] Paul Barford, Azer Bestavros, Adam Bradley, and Mark Crovella. Changes in web client access patterns: Characteristics and caching implications. Technical Report 1998-023, Boston University, December 4, 1998.

[5] Paul Barford and Mark Crovella. Generating representative Web workloads for network and server performance evaluation. In *Proceedings of the ACM SIGMETRICS International Conference*, pages 151–160, July 1998.

[6] P. Beertema. *Common DNS Data File Configuration Errors*, October 1993. RFC 1537.

[7] Peter B. Danzig, Katia Obraczka, and Anant Kumar. An analysis of wide-area name server traffic: A study of the internet domain name system. In *Proceedings of the 1992 ACM SIGCOMM Symposium*, pages 281–292, August 1992.

[8] Bradley M. Duska, David Marwood, and Michael J. Feeley. The measured access characteristics of World-Wide-Web client proxy caches. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems, Monterey, California, December 8–11, 1997*, pages 23–35, December 1997. Also UBC CS TR-97-16.

[9] E. Gavron. *A Security Problem and Proposed Correction With Widely Deployed DNS Software*, October 1993. RFC 1535.

[10] Cary G. Gray and David R. Cheriton. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. In *Proceedings of the 12th ACM Symposium on Operating System Principles*, volume 5, pages 202–10, December 1989.

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

[12] Greg Minshall. Tcpdpriv. `http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html`, August 1997.

[13] P.V. Mockapetris. *Domain names - concepts and facilities*, November 1987. RFC 1034.

[14] P.V. Mockapetris. *Domain names - implementation and specification*, November 1987. RFC 1035.

[15] Michael Olson, Keith Bostic, and Margo Seltzer. Berkeley DB. In *Proceedings of the 1999 Summer USENIX Technical Conference*, June 1999.

[16] Shawn Ostermann. tcptrace. `http://jarok.cs.ohiou.edu/software/tcptrace/tcptrace.html`, 1999. v5.2.1.

[17] J. Postel. *User Datagram Protocol*. ISI, August 1980. RFC 768.

[18] J. Postel. *Internet Protocol*, September 1981. RFC 791.

[19] András Salamon. The dns resources directory. `http://www.dns.net/dnsrd/`.

[20] Alex C. Snoeren and Hari Balakrishnan. An end-to-end approach to host mobility. In *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000, to appear)*, August 2000.

[21] Kevin Thompson, Gregory J. Miller, and Rick Wilder. Wide-area traffic patterns and characteristics. *IEEE Network*, November/December 1997.

[22] Alec Wolman, Geoffrey M. Voelker, Nitin Sharma, Neal Cardwell, Anna Karlin, and Henry M. Levy. On the scale and performance of cooperative web proxy caching. In *Proceedings of the 17th ACM Symposium on Operating System Principles (SOSP '99)*, pages 16–31, December 1999.