An Analysis of Wide-Area Name Server Traffic

A study of the Internet Domain Name System

Peter B. Danzig, Katia Obraczka, Anant Kumar
Computer Science Department
University of Southern California
Los Angeles, California 90089-0781
danzig@usc.edu

Abstract

Over a million computers implement the Internet's Domain Name System or *DNS*, making it the world's most distributed database and the Internet's most significant source of wide-area RPC-like traffic. Last year, over eight percent of the packets and four percent of the bytes that traversed the NSFnet were due to DNS. We estimate that a third of this wide-area DNS traffic was destined to seven *root* name servers. This paper explores the performance of DNS based on two 24-hour traces of traffic destined to one of these root name servers. It considers the effectiveness of name caching and retransmission timeout calculation, shows how algorithms to increase DNS's resiliency lead to disastrous behavior when servers fail or when certain implementation faults are triggered, explains the paradoxically high fraction of wide-area DNS packets, and evaluates the impact of flaws in various implementations of DNS. It shows that *negative caching* would improve DNS performance only marginally in an internet of correctly implemented name servers. It concludes by calling for a fundamental change in the way we specify and implement future name servers and distributed applications.

1 Introduction

The Domain Name System is the Internet's primary source of wide-area request–response traffic. This RPC-like traffic is responsible for over 8 percent of the packets that traverse the NSFnet backbone. This paper explains why, in two 24-hour traces of wide-area name server activity, DNS consumed twenty times more wide-area network bandwidth than was absolutely necessary. This is more than ten times the amount of excess traffic estimated from observations of DNS traffic in 1988 [13]. Despite the fact that much of the factor of twenty is due to known flaws in various DNS implementations, we argue that this calls for a fundamental change in the way we specify and implement distributed systems.

DNS is a distributed, replicated name service whose primary purposes are to map host names into corresponding Internet addresses, map Internet addresses into hostnames, and locate daemons for electronic mail transfer [12, 11]. Its name space is organized as an unbalanced tree, which, in late 1991, consisted of 16,000 different internal nodes known as *domains* and 1,000,000 leaf nodes corresponding to individual computers [8]. Essentially, all of these computers are clients of DNS. DNS client software is known as a *resolver*, and at least a dozen resolver implementations exist [7]. Some of these implementations have been re-released to repair various bugs. For example, the popular Berkeley UNIX resolver and name server software is in its seventh public release (BIND 4.8.3), and its source code lists four additional versions that were not released. We speculate that many more than thirty different versions of resolvers are currently in use by various UNIX, VMS, MacIntosh, and IBM PC computer systems.

The NSFnet Information Service reports that over the past two years the fraction of wide-area packets caused by DNS has dropped from 14% to 8% [9], while the volume of packets has doubled (see Figure 1). The fraction of DNS traffic dramatically dropped in 1990 as older, buggy versions of resolvers were upgraded. However, since early 1991, this fraction has stopped declining and may be increasing. One might argue that this fraction is not significant because DNS traffic constitutes less than 5% of the bytes transferred over the NSFnet because DNS packets tend to average a hundred bytes, which is smaller than the average packet on the NSFnet. While this is true, one should not take too much solace in this lower number because routing costs depend on packet count rather than packet size.

Most every wide-area TCP [14] conversation, whether

COMM'92-8/92/MD,USA

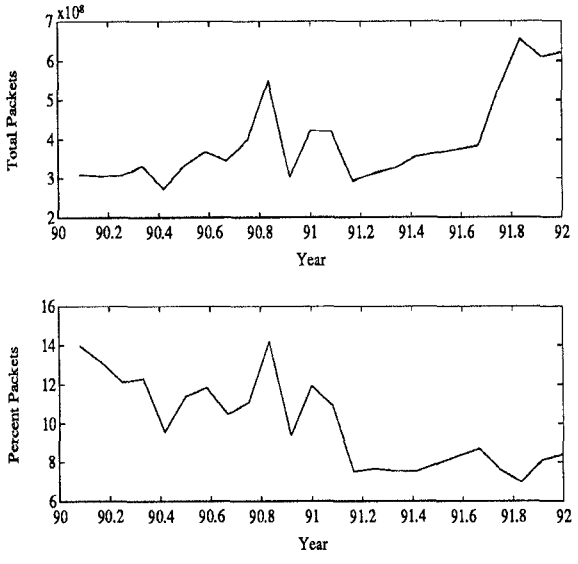
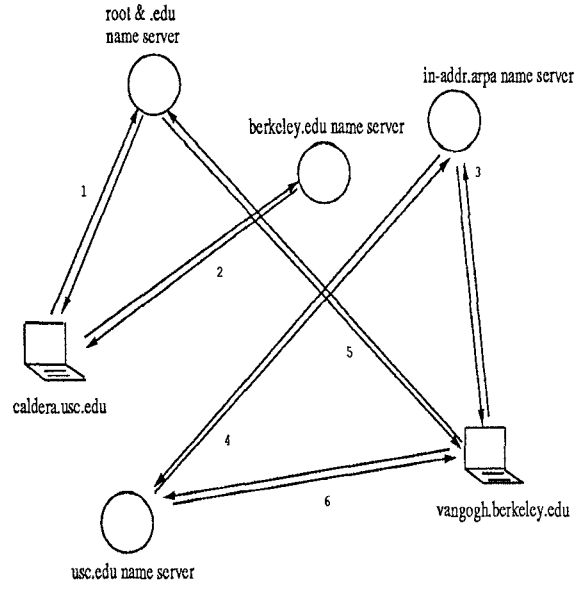


Figure 1: DNS contribution to NSFnet backbone packet counts for the past two years.



1. Caldera.usc.edu asks root for a berkeley.edu server.
2. Caldera.usc.edu asks berkeley.edu about vangogh.berkeley.edu.
3. Vangogh.berkeley.edu asks the in-addr.arpa server to map caldera.usc.edu's IP address into a host name.
4. The in-addr.arpa server chains this back to the usc.edu server.
5. Vangogh.berkeley.edu may go back to the root server to look up a usc.edu server.
6. Vangogh.berkeley.edu maps the host name returned in step 3 into an IP address.

Figure 2: DNS traffic caused by one rlogin.

it be mail, telnet or ftp, begins with two or three DNS requests. During an rlogin from *caldera.usc.edu* to *vangogh.berkeley.edu*, the initiator, *caldera*, maps hostname *vangogh.berkeley.edu* into an IP address *(128.32.133.1)*. *Vangogh*, upon receiving the rlogin request, maps *caldera*'s network address back into its hostname. Some implementations, as an additional security measure, map this hostname back to an IP address. Of course, this might not generate wide-area traffic because either or both of these mappings could be cached locally. Figure 2 illustrates the near worst case number of DNS transactions for this rlogin. *Caldera* first queries a replica of the root domain for the addresses of name servers for *berkeley.edu*. The root name server returns the list of servers for the *berkely.edu* domain. It can do this because most root name servers replicate the *.edu* domain as well. *Caldera* queries one of the *berkeley.edu* servers for hostname *vangogh.berkeley.edu*. *Vangogh*, upon receiving the rlogin request, queries a copy of the *.in-addr.arpa* name server for a name server that can translate *caldera's* network address, and then queries this name server.

Starting a new wide-area TCP conversation could require six wide-area remote procedure calls and twelve packets, but caching should eliminate most of these calls. Since the average TCP conversation currently sends about 200 packets, 100 in each direction [3, 1], one would expect the fraction of wide-area packets due to DNS would be less than 12/200 or 6%. The fact that 8% of wide-area packets belong to DNS either means that name caches are not functioning properly or something is fundamentally wrong.

2 Performance Issues

When considering the performance of any replicated, distributed name service, the obvious issues are the caching mechanism, the RPC retransmission timeout algorithm, and the algorithm for selecting alternative name servers. When DNS was initially deployed, these mechanisms were intentionally left unspecified because good solutions were not yet known. Perhaps this was done to separate the specification of mechanism and policy [4]. Today, nine years after DNS's initial deployment, reasonable solutions for each of these issues are known; this section presents one such solution.

2.1 Caching

DNS name servers cache responses to their successful queries. Usually, they do not cache responses indicating that a name is bad or indicating that the requested *resource record* is not bound to the queried name. Caching bad names or the absence of a resource record in order to answer future queries quickly is referred to as *negative caching*. The person managing a DNS domain assigns to each of the domain's resources a period of time, called a time-to-live, that other name servers can cache

it. Whenever a cached value is passed to another server, its time-to-live is reduced by the time it has remained in cache. Typical time-to-live values range from one to six days, and a value of zero disables caching.

Notice that the name server performs caching, not the client resolver code that users link with their programs. This has serious implications for a site with hundreds or thousands of computers, like a college campus. Every time a program issues a query, its resolver sends it to a name server. Well-run sites funnel all of their queries through a handful of local name servers. For example, each computer at our university sends its queries to one of ten local name servers. The likeliness of finding a working name server increases with the number of *central* servers. Ten is probably too many because it increases the likeliness that our site issues extra queries on the wide-area network. All ten servers may request the same name over the wide-area network, and, because each server sends less traffic, the quality of their wide-area RPC service time estimates decreases.

Surprisingly, several large sites have yet to channel their requests through central servers. Instead these sites run *caching-only* servers on each node which endows each node with a private cache. The effect of this policy can be seen in our traces which show four instances of 500 computers on a single campus issuing the same query. In two cases these queries targeted the same good name, and in two cases these queries targeted the same bad name. We believe neither site funnels requests through common, central name servers, but both employ caching-only servers on their individual nodes. The effectiveness of DNS name caching is discussed in Section 4.

2.2 Retransmission Algorithms

The most widely-used resolver and name server are distributed with Berkeley UNIX and, as mentioned above, are now in their seventh public release (BIND 4.8.3). This software is available by public ftp from *ftp.uu.net*, and is an example of a relatively bug-free implementation. Nearly all DNS transactions employ the UDP unreliable datagram protocol to avoid the additional messages and latency to set up and tear down a TCP connection. This places the issue of retransmissions on the user program. This section summarizes the retransmission algorithms employed by the Berkeley resolver and name server.

2.2.1 Resolver algorithm

The Berkeley resolver queries a maximum of three name servers for any request. This is parameterized and can be increased. The resolver sends the query to the first of the three servers and waits for a timeout. If it receives no reply, it sends it to the second server and waits. If it times out without receiving a reply from either server, it sends it to the third server and waits. If it times out again without receiving a reply from any of the three servers, it doubles the timeout interval and tries the three servers in sequence again.

The initial timeout interval is set to the maximum of 1 second and the integer part of 5 seconds divided by the number of servers. In this case the timeout starts at $\lfloor 5/3 \rfloor = 1$ second.

A server is tried no more than four times; thus, a total of 12 queries might be sent before giving up. Each of these queries carries an identifier that must match any response. The Berkeley resolver does not change this identifier upon retransmission. The resolver does not care which server responds nor which query packet generated this response.

2.2.2 Name server algorithm

The Berkeley name server keeps track of up to 16 different server addresses to resolve a request. It keeps the possible servers sorted by their expected response time to previous queries and queries the addresses in this order. It cycles through the servers up to three times, except when infinite looping occurs. The base retransmission timeout is set to the larger of 4 seconds and twice the expected service time of a server. After completing a cycle, it doubles the retransmission timeout, although in all cases this timeout is limited to 45 seconds.

If a response to a query is a pointer to another name server or an instruction to try a different *canonical* name, looping might occur. A canonical name is an alias for the query name; for example, *castor.usc.edu* is an alias for *girtab.usc.edu*. The server limits itself to a total of 20 referrals or 8 canonical name substitutions. If no referrals or canonical name substitutions occur, the server returns with status *server failure* after completing the cycle of $16 \cdot 3 = 48$ queries. The only exception to this occurs with queries for the set of root name servers. These are called *system* queries and are repeated hourly until they succeed.

The server drops retransmitted queries if it is still trying to resolve the original query. That is, if the server receives a retransmission of a request for which it is recursively querying another server, it does not process this new request. When the server has an answer, it forwards it to the resolver or name server as an answer to the original query. The name server records the query-identifier sent to it by the requester and uses this

identifier when it responds. It assigns a new identifier to queries that it forwards, and uses this new identifier for all retransmissions of that query. However, it generates a new identifier each time it substitutes canonical names or is referred to a different server.

Name servers need to be *primed* with the host names and addresses of several root name servers. This list is stored in the file system and can become inconsistent over time. A name server generates a system query when it uses its priming list to get a complete list of root name servers, which it caches for the indicated time-to-live.

2.3 The net effect

A Berkeley name server will send up to 3 requests to each address of another name server to resolve a query it receives. Since a resolver could query up to 3 name servers, a single query destined to an unreplicated, unreachable server would lead to about 9 query packets.

Because a name server must be able to contact at least one root server, system queries receive special treatment and are retried every hour until they succeed. This implies, for example, that if only one of two addresses in the priming file is good and this server is down, a total of 3x24 = 72 requests would be directed towards this server in the space of 24 hours. The same number of packets would be sent to the bad address.

Section 4 analyzes the effect of name server failures on other resolvers and name servers with more aggressive retransmission algorithms than the ones reviewed here.

3 Traces

This study is based on two 24-hour packet traces collected from DNS root name server *a.isi.edu*[1] and three other name servers that replicated various domains. Figure 3 illustrates the network configuration. The root name server had network interfaces on the Milnet and an Ethernet that lead to the Internet via the Los Nettos regional network. Note that the root was not a BIND name server. All four servers had interfaces on this same Ethernet. A fifth workstation, a Sun SPARCstation 1 with 10 millisecond clock resolution, collected all DNS packets that traversed the Ethernet, including DNS packets with both source and destination on the local Ethernet.

[1] We would have liked to collect at least one more trace. Unfortunately, by the time we had the analysis program working correctly, this node had retired and was no longer a root name server.

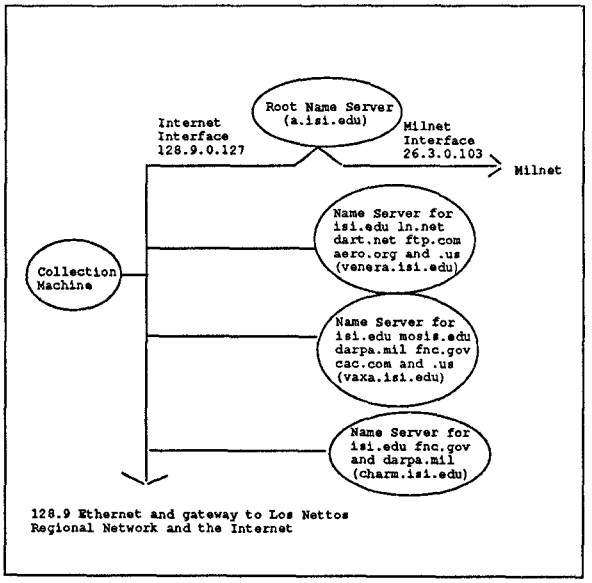


Figure 3: Network configuration and traced name servers.

Only packets sent over the Ethernet were collected. Queries arriving at the root name server's Milnet interface could not be traced. Surprisingly, however, replies to most of these queries *were* collected. As it turned out, to avoid per-packet networking charges on the Milnet, this machine manually routed packets not destined to the Milnet over the Ethernet to the Internet. Conceivably, some of these return routes were down. That is, the server was reachable on the Milnet, but the responses were dropped in the Internet. Our traces did not show evidence that this occured frequently. We reconstructed the majority of queries to the Milnet address from the responses captured on the Ethernet, and estimated the arrival rate of queries at the Milnet interface by assuming that the request to response ratio for the two interfaces were nearly the same. Thus, the estimated request rate was calculated as the request rate at the Internet interface multiplied by the ratio of the Milnet response rate to the Internet response rate.

3.1 Trace Collection

The collection machine used a network interface tap (NIT) program [10] to collect all TCP and UDP DNS packets, including non-transit local traffic. More than 98% of the bytes and packets were sent by UDP; for simplicity we did not analyze TCP packets. Table 1 summarizes the two traces. NIT recorded that fewer than 1% of packets were lost to overflowed NIT buffers. The only other place that packet loss could occur was the network interface.

Trace	Packets			
	udp	tcp	dropped	wide-area
5/91	944,164	8,889	2,913 (277,739)	880,370 (277,739)
9/91	1,075,921	14,377	8,176 (268,928)	903,652 (268,928)
Total	2,020,085	23,266	11,089	1,784,022

Table 1: Summary of traces. Derived Milnet query counts are indicated in parenthesis.

3.2 End-to-end loss rate

We derived a method to calculate end-to-end loss rate because our collection machine's network interface does not count dropped DNS packets. Although the end-to-end loss rate that we calculated was below 1%, we review this technique here because it can be applied to estimate loss rate for traces of any request-response protocol. The method assumes that the three name servers that provide domain *isi.edu* can reply quickly to queries against this domain by computers on the local Ethernet. This assumption means, for example, that if *zephyr.isi.edu* queries name server *charm.isi.edu* for the network address of *venera.isi.edu*, that *charm* transmits its response before *zephyr* gets impatient and retransmits its query. This assumption means that requests and responses come in pairs. If *charm* misses the request and fails to transmit a response, then this loss is charged to our collection machine.

Denote the total number of local queries transmitted during tracing by n and the probability that the trace gathering program loses a packet by p. The probability of capturing the query but dropping the corresponding response is $(1-p)p$. The probability of capturing the response but not the corresponding request is also $(1-p)p$, and the probability of capturing both the request and response is $(1-p)^2$.

Denote the number of unmatched local queries by q, the number of unmatched responses by r, and the number of matched query-response pairs by m. Summing the number of unmatched queries and the number of unmatched responses leads to

$$2np(1-p) \quad = \quad r+q.$$

The number of matched query-response pairs on the trace, m, should equal the probability of capturing both the query and its corresponding response multiplied by the total number of local query-response pairs, n,

$$n(1-p)^2 = m.$$

Dividing the first equation by the second leads to

$$2np(1-p)/n(1-p)^2 \quad = \quad (r+q)/m,$$

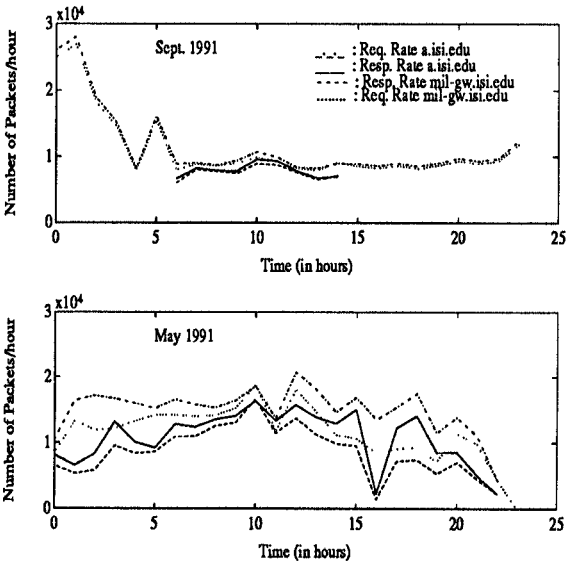


Figure 4: Request and response rate at root name servers.

$$2p/(1-p) \quad = \quad (r+q)/m.$$

Carrying out the algebra yields the loss rate

$$p = \frac{\frac{r+q}{m}}{2+\frac{r+q}{m}}.$$

We calculated the collection's loss rate by computing m, r, and q for local queries to name server *charm*, and estimated a trace loss rate of 0.77% for the new trace (m=13490, r=115, q=97).

3.3 Query and response rate

Figure 4 plots the query and response rate measured from the root name servers, and Figure 5 plots the request, response, and chained request and response rates for name server *charm*. Chained requests occur when a name server issues its own query to answer a query that it received. Chained responses are the answers to chained requests. Several points merit attention. In the second trace, the root name server was only up for nine hours. It first recovered in the seventh hour of the trace and crashed again nine hours later. In addition, the high arrival rates for name server *charm* on the second trace were caused by a single host; this is explained in the next section.

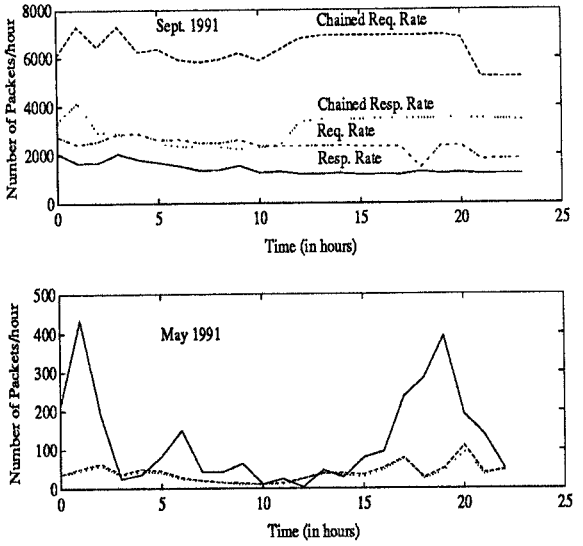


Figure 5: Request, response, and chained request and response rates at one of the other three name servers.

	5/91	9/91
Minimum necessary queries	43,535	15,787
Preventable queries	9,945	4,395
Total query packets	695,899	985,518
Possible improvement	16	61

Table 2: Possible DNS performance improvements.

4 Analysis

Analysis of the DNS traces shows that DNS consumes at least twenty times more wide-area bandwidth than is strictly necessary. This section explains the reasoning that leads to this conclusion and attempts to quantify the factors contributing to this performance. DNS bugs tend to exacerbate one another, and this complicated the analysis. This section starts by estimating the absolute minimum number of queries and packets that we would expect if all computers implemented DNS perfectly. Then it describes some of the difficulties in classifying DNS errors, presents the classification scheme that we devised, and discusses wide-area DNS performance using our scheme.

The first entry in Table 2 estimates the minimum number of query packets that the name servers would have received had name caching and negative caching been perfectly implemented throughout the Internet. It excludes preventable queries as explained in the next paragraph. If local caching worked perfectly, regardless of the number of computers on a given network, the network would query for a name only once per day, assuming that most data items have time-to-lives on the order of a day or more and that name servers do not fail.

The table's second entry adds preventable queries to the first number. One form of preventable query occurs when a resolver wildly appends a domain name to the original query name as a helpful search feature. A sample preventable query from the September trace is *bart.mit.edu.dec.com*, where some resolver appended *dec.com* to the otherwise good name *bart.mit.edu*.

The table's third entry lists the number of query packets on the trace, and the fourth entry lists the ratio of query packets to minimum necessary queries. This is our justification for conservatively claiming a bandwidth reduction of a factor of twenty.

4.1 An imperfect world

Various name server and resolver bugs interact with one another. Nearly 9,500 copies of query *bart.mit.edu.dec.com* were sent to both addresses of the root name server by three different Digital Equipment Corporation name servers. The root name server responded 3,097 times. We suspect that more copies of this query were sent to other root name servers as well. The same three name servers queried the root name server another 8,375 times for *bart.mit.edu* and received 2,688 replies.

What happened here? Some resolver sent the query for *bart.mit.edu* to one of its name servers. This name server tried to resolve the name with our root name server, but this name server was down. The resolver timed out and queried the other two name servers. Possibly the priming lists of root name servers for these three name servers were out of date, and none could find another root server to resolve the name. This is supported by the fact that all three name servers queried hundreds of times for a copy of the root domain while the root server was down. When the resolver failed to get its answer, it generated a query for *bart.mit.edu.dec.com* and sent this to all three name servers. All three name servers queried both names until the server recovered. When the root server recovered, it responded that *dec.com* was the appropriate domain to contact for query *bart.mit.edu.dec.com* and that *mit.edu* was the appropriate domain for query *bart.mit.edu*. The name servers did not accept these responses, but kept querying both names several thousand more times.

How should this bug be classified? These servers and resolvers suffer from a combination of problems. The resolver constructs preventable queries. The name servers probably are not properly primed with alternate roots, do not detect server failure, do not backoff their retransmission timeouts properly, probably don't cache properly, and suffer from a form of looping because they do not forward queries properly. Nevertheless, this section

Error Class	Packet Count	
	05/16/91	09/27/91
Recursion bug	189,868	172,482
Zero answer bug	14,947	52,597
Server failure detection	13,537	405,972
(Milnet Packets)	(12,860)	(333,180)
Name error bug	93,465	26,330
Centralized caching	–	–
Negative caching	6,136	701
Retransmission timeouts	636	488
Leaky caches	24,103	6,320

Table 3: Query packet counts due to specific errors. This table ignores queries packets to and from the Milnet interface.

describes how we managed to classify name server interactions from thousands of different servers and resolvers, many of which exhibited rather bizarre behavior[2].

4.2 Error Classification

Table 3 lists the seven classes of DNS implementation errors that we identified. The table estimates the number of packets due to each type of error.

4.2.1 Recursion

The implementation error that consumed the most wide-area bandwidth is a recursion or looping error. This occurs when a response to a query contains no answer, but contains a list of other name servers to contact. Particularly pernicious servers include themselves on this list, causing naive name servers to resend their query back to the same name server. Longer loops are possible. For example, on the September trace, two such queries caused 259,529 and 223,575 packets respectively, about half of the entire trace. In both cases, a machine at ISI, *zephyr.isi.edu* queried name servers *venera.isi.edu* and *charm.isi.edu* for the host name that corresponded to particular internet addresses (*in-addr.arpa* queries). In both cases, the ISI name servers contacted remote servers that included themselves on the list of other name servers to contact. The second case involved three such servers replicating the same domain. This problem was exacerbated by *zephyr*, which issued this query every five seconds throughout the trace. Note the difference in the request and chained request rates at name server *charm* in Figure 5 for the two traces. The root name server was the victim of recursion errors in both traces. Negative caching would have contained these queries to ISI's Ethernet.

[2]We admit that a classification of "brain-dead" was tempting.

We place a query in this class when it receives a reply with no answer, no error indication, but a list of additional servers to contact. This is reported in the first entry of Table 3.

4.2.2 Zero answer bug

The zero answer bug was introduced in an old release of SunOS [13] and is still responsible for a good deal of DNS packets. In this error, a server responds that the requested *resource record* bound to the requested name doesn't exist, but that the name is valid. Many resolvers retry this query or contact the other servers on the list, causing recursion.

We place a query in this class when it receives a reply with no answer, no error indication, and no list of additional servers to contact. This is reported in the second entry of Table 3.

4.2.3 Server failure detection

Many servers and resolvers do not detect server failure[3], do not exponentially back off their query rate, and are not primed with a good list of root servers. Figure 6 plots the number of packets sent to the root server's Internet interface during two long crashes. The upper graph plots the probability that a host sends less than a certain number of packets. The lower graph plots the cumulative distribution of the contribution to total packet count of hosts that send less than a certain number of packets.

Only 10,616 unique queries caused 437,820 packets. Because the arrival rate at the Milnet interface is usually close to 90% of the rate at the Internet interface, we estimate that 771,000 packets were actually sent.

During the first failure, 61% of name requests sent just one packet; some other server must have resolved the name. For the second failure, this number was 37%. Despite the fact that about half of the hosts responded correctly to the server failure, the other half sent a metric ton of packets. This is reported in the third entry of Table 3.

4.2.4 Name error bug

When a server responds that a name is bad (it almost always does this authoritatively for lack of negative caching), the requester should be satisfied. If the server does

[3]Paul Mockapetris told us of one name server implementation that, when it fails to receive a response from a server, sets its estimated round trip time to a compile time constant like 6 seconds.

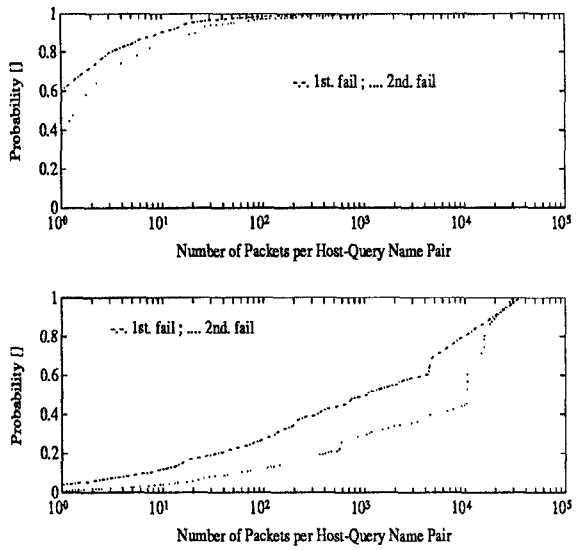


Figure 6: Response to server failure.

not return a list of additional servers to contact and the query is retransmitted more than three times, we classify this as a special form of the zero count bug. We call this a name error bug, and the fourth entry in Table 3 lists the number of additional query packets that it caused.

Recall that Table 2 identifies the number of preventable queries caused by badly implemented resolver search lists. The two traces had 10,270 and 1,274 queries for unpreventable name errors, and 9,945 and 4,395 queries for preventable name errors. Preventable name errors can evoke other bugs, as described above. We do not attribute packet counts in Table 3 for preventable name errors. If a preventable query evokes one of the listed errors, its query counts are reflected in the appropriate row of Table 3.

4.2.5 Centralized caching

Most domains already channel their requests through centralized caches. Figure 7 histograms the probability of receiving the same query from one or more hosts within a domain. Note that the log-log plot accentuates the high host count cases. The average number of hosts within a single domain issuing the same query is 1.067 and 1.049 on the old and new trace respectively. There are singular exceptions. Over 600 machines at MIT and over 100 machines at CERN queried for the list of root servers. Over 100 machines at CMU made an *in-addr.arpa* query for *dist.fac.cs.cmu.edu*. Otherwise, the low average number of hosts per query indicate that centralized caching is nearly completely in place. Only a handful of domains exhibit more than ten hosts per query, and except for the examples mentioned above, these were all the result of defective resolvers unsatisfied with zero-count answers. We do not measure this effect in Table 3. Frequently, requests from many hosts within a domain indicate some other type of error is occurring.

4.2.6 Negative caching

Assume that everyone employed centralized caching and corrected their DNS implementation errors. What additional benefits would negative-caching accrue? In both traces, nearly 500 machines at CMU queried the name error *THISHOST*, and over 100 machines at MIT queried the name error *0.0.0.0.in-addr.arpa*. On the old trace, about 100 instances of twenty-six and twenty-seven hosts from Johns Hopkins Applied Physics Lab tried to resolve variations on the theme "LABEL INVALID".

Our measurements indicate negative caching would eliminate 7111 packets. This pales in comparison to the number of packets caused by other types of bugs. Recall from Table 2 that more than half of current name error queries are preventable and due to poor resolver search paths. The remaining name error queries are mostly typos that probably will not be repeated. Certainly the *THISHOST* and *0.0.0.0* queries are due to buggy, shared software.

The conclusion we draw about negative caching is that it serves as a safety valve to protect the wide-area network from malicious and wild user programs. Implementing negative caching on a server does not reduce wide-area traffic if some remote resolver or server is the source of the queries. If a site employs negative caching on its central name servers, then this site protects the wide-area network from resolvers on its machines. We believe it is first preferable to focus on fixing bugs, and afterwards, as needed, to implement negative caching. Currently, the benefits of negative caching are overstated.

4.2.7 RPC Timeouts and Cache Leaks

We started this project to study the performance of wide-area RPC and the effectiveness of name caching. Currently, implementation flaws reduce the importance of these two issues. Nevertheless, we attempted to study the effectiveness of RPC timeout selection algorithms by restricting our attention to queries that received positive answers. Insisting on an answer avoids DNS flaws having to do with zero answer counts.

Distinguishing between excessively short RPC retransmission timeouts and cache leaks requires making an arbitrary distinction. We declared queries retrans-

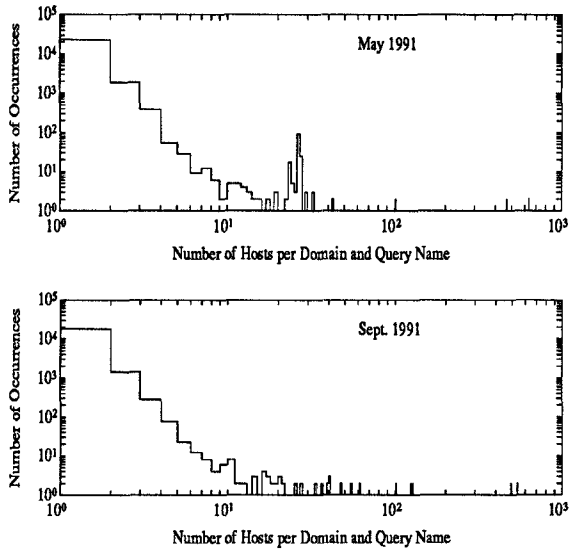


Figure 7: Multiple name servers and resolvers.

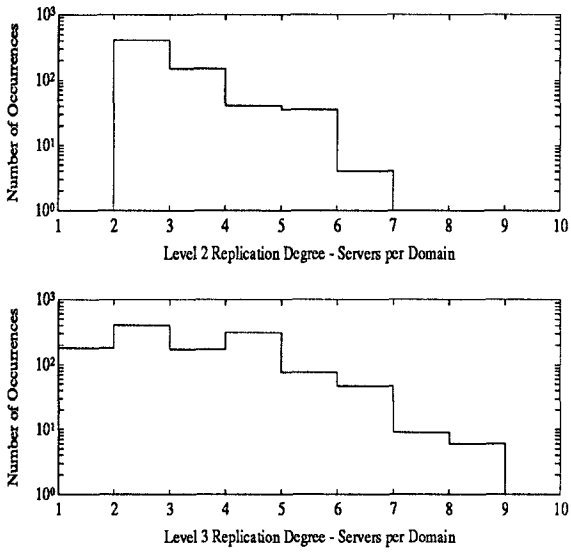


Figure 8: Number of servers replicating second and third level domains.

mitted within a minute as due to poor timeout estimation and all others as due to cache leaks[4]. The last two entries of Table 3 estimate the number of extra query packets due to retransmitting requests too quickly and due to cache leaks.

These numbers are estimates for many reasons. DNS uses the unreliable datagram protocol, and responses captured on the trace may not be received at the destination. Users may abort a program, but then reexecute it so quickly that it looks like a retransmission. The quality of retransmission timeouts should be reevaluated as bad DNS implementations disappear.

4.3 DNS Replication

The number of times that a domain is replicated across alternate servers affects DNS's performance in two ways. Resiliency to individual server failures increases with the degree of domain replication. Unfortunately, the severity of DNS bugs having to do with zero answer counts and recursion from forwarding lists increases with the degree of replication. Our traces showed that hundreds of buggy name servers sent packets to both network addresses of the ISI root name server. This leads us to speculate that some buggy implementations probably inundate all replicas of a domain with the same bad queries.

The root domain, now and when our traces were taken, is replicated seven times. Our root server received approximately 657 and 532 thousand queries in our two traces, or about 7 requests a second. It responded to three-quarters and one-quarter of these queries in the May and September traces respectively, for about 1,149 and 665 thousand wide-area packets. Note that in the later trace the server was only working for nine hours. We suspect that the other six root servers were involved in loops and zero count bugs for many of the same queries. However, we do not have data to support this.

In May 1991, NSFnet reported 11 million DNS packets per day. The ISI root name server sent and received approximately a million wide-area packets the day we traced. If the other six root servers also sent and received about the same number of packets, then the root servers sent or received 7 million DNS packets per day. Suppose half of these 7 million packets traversed the NSFnet. Then the root name servers contributed about a third, 3.5/11, of the DNS packets on the NSFnet.

We wrote a program that traversed the naming tree of the *.edu* domain, and computed the degree of replication of second level (e.g. *berkeley.edu*) and third level (e.g. *cs.berkeley.edu*) servers. This is plotted in Figure 8. Note how all second level domains are replicated at least twice, and many are replicated three or more times (Surprisingly, many third level domains are not replicated at all!). Defective name servers will inundate all replicas of a given domain.

Note, however, that domain replicas do not run on individual servers, but that most servers replicate several domains. Figure 9 histograms the number of domains that various *.edu* servers replicate. This means that when any one of these domains becomes the target

[4] Someone really ought to study the choice of time-to-live values.

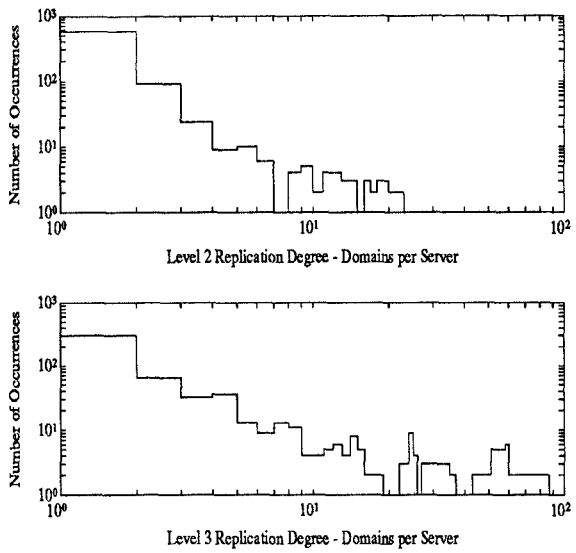


Figure 9: Number of DNS domains replicated on individual second and third level name servers.

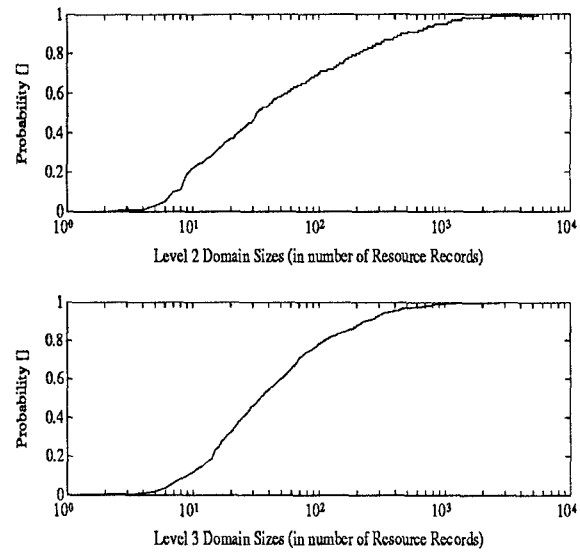


Figure 10: Distribution of number of resource record in the *.edu* domain.

of a defective name server, it may not just degrade performance for one domain, but may degrade performance for many domains.

When a domain is updated, all secondary servers for the domain must update their copy. Secondary servers periodically check with the primary server to see if the domain has been updated. The primary copy typically is queried four times an hour by each secondary server. When the domain changes, the domain database is transferred using TCP in an operation called a *zone transfer*. Most domains let anyone invoke a zone transfer, although many large vendors do not. For example, our program to traverse the DNS name space employed zone transfers. This leads us to ask if zone transfers impact wide-area DNS performance?

Although zone transfers employ TCP, zones are transferred one resource record at a time. If a zone contains 100 resource records, the zone transfer consists of about a 100 TCP packets and 100 acknowledgements. Our traces of ISI, UCLA, and USC DNS traffic show only about 1 to 2% of DNS packets are TCP packets. These leads us to believe that zone transfers are not a significant source of DNS traffic. Figure 10 shows the distribution of the number of resource records stored in second and third level *.edu* domains.

In summary, while replication is supposed to lead to resiliency, defective servers usually query all replicas of a domain exhaustively while searching for an answer.

4.4 Assessment

As older resolvers and name servers are replaced with bug-free implementations, wide-area traffic due to DNS will dramatically decrease. One perniciously paired name server and resolver can easily cause a million wide-area DNS packets a day. Clearly, network managers must install bug-free DNS software that limits retransmissions from servers and resolvers, uses good backoff algorithms, channels naming requests through a small number of local name servers, and implements resolver search lists correctly. We believe, however, that this will not quickly solve the problem.

The success of this approach depends on the managers of the various computers and name servers, and trusts that vendors introduce no new bugs in their naming software. Such assumptions are naive, and this approach ignores the essential ingredient for getting good performance out of computer systems: active error detection.

5 Error Detecting Servers

Systems must be designed to monitor their own behavior and seek to identify misbehaving components. Most implementations overlook this ingredient on the grounds that it degrades common case performance. The truth is that DNS now consumes about twenty times more bandwidth than it should. The cost of mechanisms to identify misbehaving implementations would be quickly recovered, and these mechanisms would increase Inter-

net reliability by identifying sources of trouble.

Servers should detect common implementation errors in other name servers and resolvers. This is not to say that we recommend that every packet be traced. Of course, doing so reveals fascinating things: for example, some IBM PC/RT at Stoneybrook sends unsolicited empty DNS response packets to the ISI root name server every ten minutes. Although fascinating, this is not a common error and really doesn't affect wide-area DNS performance. On the other hand, experience shows several types of common errors are worth checking for. Many DNS name servers and resolvers suffer from zero count and recursion bugs, don't back off their retransmission timers, and don't deal with server failures gracefully. These bugs cause trouble and merit attention.

5.1 Feigning Death

A name server can detect bad behavior to server failure, bad root server priming caches, and bad retransmission timeout algorithms, if it occasionally feigns to be dead. It need not feign total rigor mortis. Rather every 200 or 300 queries it selects a name from some query and decides not to answer queries for this name for the next five minutes. During this time it records all attempts for this name in a data structure. If a server tries to query the name more than two or three times or if many servers from the same domain request it, it is likely that this domain does not deal with server failure well. The server can keep a log file of the network addresses, domain names, and packet counts of suspicious cases. After 5 minutes, it can garbage collect the data structure and return to normal behavior.

5.2 Bean Counting

The typical name server is too patient. You can ask it the same stupid question ten thousand times, and each time it will spit back the same answer. People are not this patient. Name servers should maintain a data structure, indexed by query name, that maintains the network addresses, request and response packet counts, and the timestamp of the most recent query. This data structure can be garbage collected as the timestamps grow old. If a garbage collected entry has too high of a packet count, this should be written to the log file. If a live entry's packet count grows too high, and the response code of this query indicates the query has zero answers, the remote resolver or name server may be suffering from a zero count or recursion bug.

A name server could even elect to tell a white lie that breaks endless cycles and then log the occurrence. For example, for name-to-address queries, the server could return the requestor's IP address in place of a zero count answer. Unfortunately, because application programs do not anticipate receiving incorrect answers, this may not work well.

Servers could also flag time-to-live values orders of magnitude larger or smaller than the recommended 24-48 hours.

5.3 Policing

Once a day, our name server should spawn a process that reads the log and sends electronic mail messages to the managers of machines or domains with defective naming implementations. Similar functionality could be implemented at wide-area network switches. These switches could snoop on name traffic and check for anomalies. Policing will quickly accomplish what hope and patience has not; it will fix naming.

6 Conclusions

The portion of wide-area network traffic due to DNS will decrease as defective name servers and resolvers are replaced, assuming no vendor releases another devastating bug. If, for example, all implementations followed the Berkeley name server and resolver algorithms (see Section 2), wide-area DNS traffic would decrease by a factor of twenty and perhaps more. Analysis of our traces indicate that negative caching is unnecessary in an Internet of properly run name servers. Negative caching can serve, however, as a firewall between a wide-area network and malicious or badly broken programs.

We doubt, however, that an entire Internet of correctly implemented resolvers and name servers will emerge. As old implementations are corrected, buggy new implementations will arise from vendors, operating system changes, and bridges between heterogeneous naming systems. Given that a defective resolver and name server can generate a million wide-area packets a day, we recommend that name servers for DNS, other naming systems [5], and resource discovery systems [6] actively check for defective implementations. We are currently writing such software for the Berkeley BIND name server [2]. We hope to install this software into a root name server in the coming months.

Acknowledgements

Our thanks to Andrew Cherenson, Paul Mockapetris and Mike Schwartz for their thoughtful comments. Our

thanks to Daniel Zappala for proofreading drafts of this paper. Chyifeng Ding worked on this project and deserves credit for conceiving the loss rate estimation method. Vipul Gore helped revise Chyifeng's trace analysis program. We are also grateful to Danny Mitzel, Walt Prue, and Jim Koda, who helped collect various DNS traces at ISI, UCLA, and USC.

References

[1] John Crowcroft and Ian Wakeman. Traffic analysis of some UK-US academic network data. Technical report, University College London, September 1991.

[2] Peter Danzig. Probabilistic error checkers: Fixing DNS. Technical report, USC, February, 1992.

[3] Peter B. Danzig, Sugih Jamin, Ramon Caceres, Danny J. Mitzel, and Deborah Estrin. An artificial workload model of TCP/IP internetworks. *To appear in the Journal of Internetworking: Practice and Experience*, 1992.

[4] Roy Levin et al. Policy/mechanism separation in HYDRA. *Proc. 5th Symp. on Operating Systems Principles*, 1975.

[5] International Organization for Standardization. Information Processing Systems – Open Systems Interconnection – The Directory – Overview of Concepts, Models, and Service. Technical report, International Organization for Standardization and International Electrotechnical Committee, December 1988. International Standard 9594-1.

[6] Robert E. Kahn and Vinton G. Cerf. The digital library project Volume 1: The world of Knowbots. Technical report, Corporation for national Research Initiatives, 1988.

[7] W. Lazear. Milnet name domain transition. RFC 1031, November 1987.

[8] Mark Lottor. Internet domain system. *In ACM Forum, CACM*, 34(11), November 1991.

[9] Merit/NSFnet Information Services. *Link Letter*, 3(6), January/February 1991.

[10] Sun Microsystems. Network Interface Tap. SunOS 4.0 Documentation.

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

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

[13] Paul V. Mockapetris and Kevin J. Dunlap. Development of the Domain Name System. *1988 ACM SIGCOMM Symposium*, August 16-19, 1988.

[14] J. B. Postel. Transmission Control Protocol. RFC 793, September 1981.