CoDNS : Masking DNS Delays via Cooperative Lookups

KyoungSoo Park, Zhe Wang, Vivek Pai and Larry Peterson
Department of Computer Science
Princeton University

Abstract

The Domain Name System (DNS) is a ubiquitous part of everyday computing, translating human-friendly machine names to numeric IP addresses. With its redundant design, aggressive caching, and widely-assumed reliability, few suspect its internal failures as a source of delays. We show, through careful measurement, that the infrastructure responsible for resolving DNS names often encounters various failures which then induce delays. A systematic examination of the problem shows that the failures are widespread, uncorrelated, and can be a significant source of DNS-related delays.

We address this problem via the development of CoDNS, a cooperative DNS lookup service. It uses a locality and proximity-aware design to achieve low-latency, low-overhead name resolution in the presence of local DNS nameserver delay/failure. We show via repeated measurement and live traffic that CoDNS is an effective solution to DNS problems, and eliminates a major source of delay.

1 Introduction

The Domain Name System (DNS) has become a ubiquitous part of everyday computing due to its effectiveness, human-friendliness, and scalability. It provides a distributed lookup service primarily used to convert from human-readable machine names to Internet Protocol (IP) addresses. Its existence has permeated much of computing via the World Wide Web's near-complete dependence on it. Thanks in part to its redundant design, aggressive caching, and flexibility, it has become a ubiquitous part of everyday computing that most people take for granted. Given its generally high reliability, few people suspect simple failures or oversights in deployment for being responsible for noticeable delays in the Web connections.

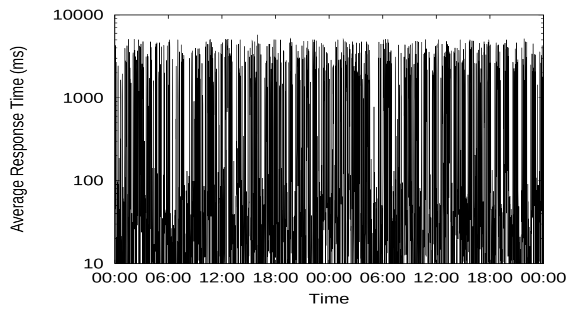
DNS employs multiple levels of redundancy and caching to improve its performance and hide short-term failures. Beginning with the 13 root nameservers, lower levels of the hierarchy are responsible for deploying at least one pair of nameservers so that name lookup queries can be resolved in the event of failures. Lookup results also carry explicit time-to-live (TTL) information, to aid in caching and to reduce load on all levels of the DNS hierarchy.

DNS "users" can choose to perform lookup manually by querying each level of the hierarchy in turn until the complete name has been resolved, but most systems are configured to delegate this task to a set of local nameservers. This approach has several performance advantages, since a centralized lookup service may consolidate requests and serve replies from its cache. It also has management advantages, since any updates to root server information or to the nameserver software are applied to fewer machines. Given the importance of centralized lookup nameservers in organizations, several are often deployed to provide redundancy.

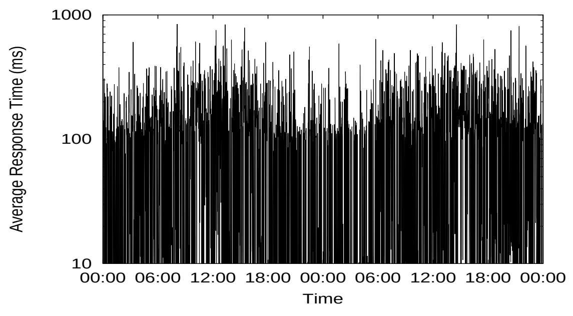
The advent of wide-area distributed testbeds such as PlanetLab [12] provides a means for a large-scale examination of DNS performance, and we find a number of surprising behaviors through repeated measurements. These measurements are taken as part of the CoDeeN Content Distribution Network [16], and are used to assess the quality of the PlanetLab nodes on which the CoDeeN software runs. Part of the monitoring process involves querying local nameservers for the addresses of long-lived, well-advertised PlanetLab node names. The observed behaviors largely consist of lookup "failures" which are hidden by the internal redundancy in DNS deployments. However, the cost for such redundancy is additional delay, and we find that the delays induced through such failures often dominate the time spent waiting on DNS lookups.

In addition to diagnosing failures, we also show how this sort of testbed can be used to augment existing nameservers, providing a cooperative lookup scheme to mask the failure-induced local delays. Using such an approach, PlanetLab nodes can provide an aggregate DNS lookup performance that is more reliable and consistent than any individual node's performance. We term our implementation CoDNS, and use it to augment the performance of the CoDeeN CDN.

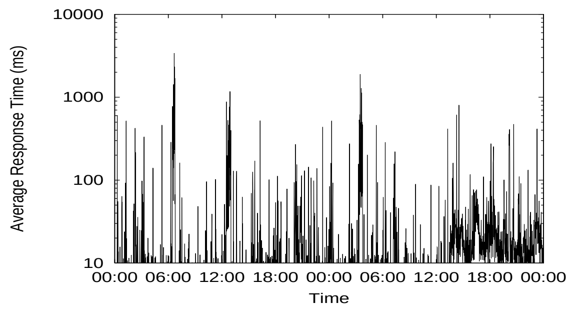
The rest of this paper is organized as follows: in Section 2, we provide evidence of widespread failures in local DNS nameservers, analyze their behavior, and show that the measurements are not an artifact of the PlanetLab nodes. In Section 3, we propose a model for cooperative



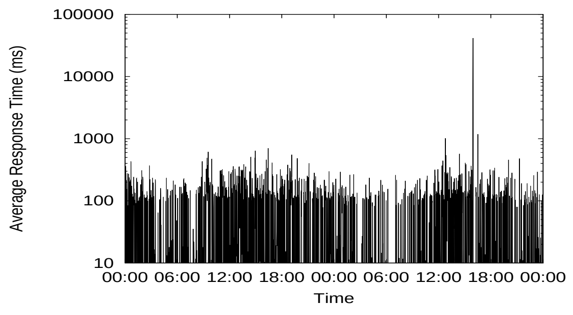
(a) planetlab1.cs.cornell.edu



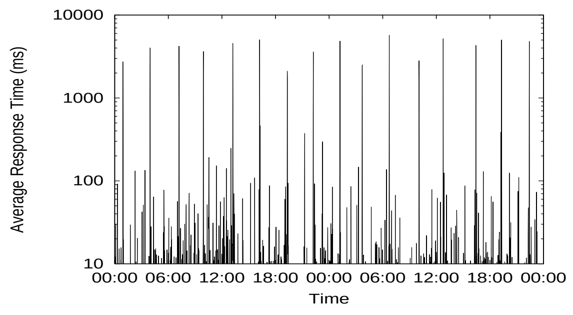
(b) lefthand.eecs.harvard.edu



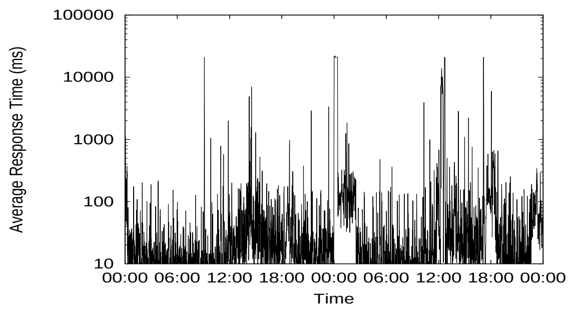
(c) planetlab-1.cmcl.cs.cmu.edu



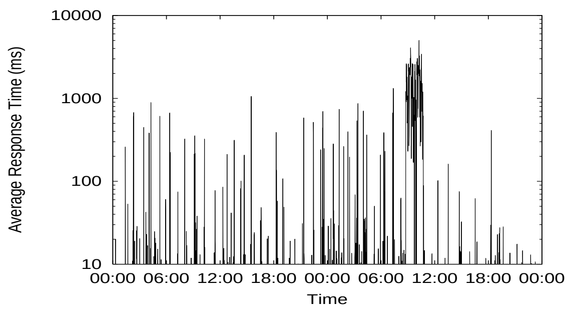
(d) kupl1.ittc.ku.edu



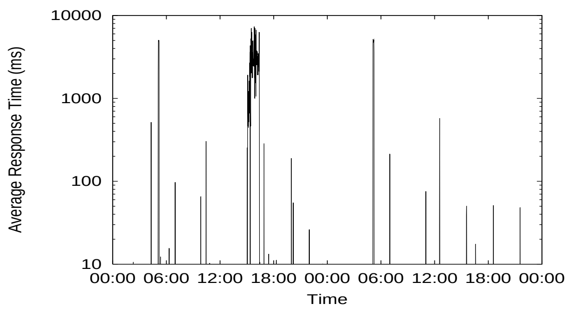
(e) planetlab-1.stanford.edu



(f) planetlab1.cs.ubc.ca

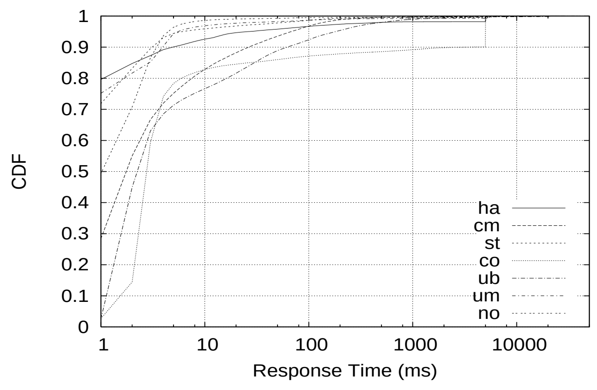


(g) planetlab1.eecs.umich.edu

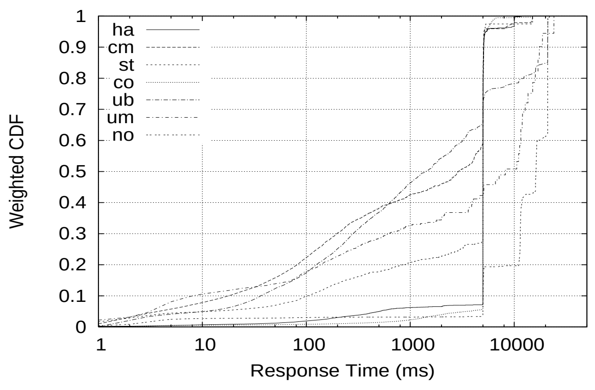


(h) planetlab2.cs.northwestern.edu

Figure 1: 12/11 - 12/12, Average Response Time for Repeated Name Lookup Testing of Various CoDeeN Nodes



(a) Percentage of lookups taking $<$ x ms



(b) Percentage of the sum of lookups taking $<$ x ms

Figure 2: 12/11 - 12/12, CDF for distribution of lookups (School names are abbreviated with first two characters)

DNS lookups, analyze the expected performance and associated overheads, and compare other scenarios. In Section 4, we discuss our implementation of such a service, CoDNS, and we analyze its performance in Section 5.

2 Background & Analysis

Our observation of the variety of local problems associated with DNS stem from our work on the CoDeeN content distribution network (CDN). CoDeeN consists of a network of Web proxy servers that include custom code to control request forwarding between nodes. When a CoDeeN node receives an HTTP request, it attempts to service it locally if the requested object is cached. On cache misses, requests are forwarded to other CoDeeN nodes by considering locality, load, and proximity [15]. The CoDeeN node receiving the forwarded request acts as a reverse proxy for the origin server, and tries to satisfy the request from its cache. If this node also does not have the object, the request is sent to the origin server.

When CoDeeN must forward requests to the origin server, it performs a DNS lookup to convert the server's name into an IP address, and our attempt to ensure that this process was successful brought the DNS problems to our attention. The CoDeeN node that receives the request originally needs to ensure that whichever peer node receives the request is capable of handling it in a timely manner. CoDeeN runs on PlanetLab, a worldwide network testbed for large-scale services, and Planetlab nodes are hosted by various universities, companies, and other groups. Each node is configured to use local nameservers, so a local DNS nameserver failure/delay at the peer node would render it a poor candidate for handling requests. To avoid these nodes, we include information about the local DNS performance in our intra-CoDeeN health messages.

Our desire to have a standard for comparison across all CoDeeN nodes led us to have a process at each node periodically perform name lookups of other CoDeeN nodes, since having the same lookups performed at all nodes would ensure a standard baseline for comparison. These lookups were included as part of an end-to-end test of proxy availability, but we included timing information for DNS separately. Given the small set of CoDeeN nodes (around 40), their long TTL values, and the fact that most are hosted on Internet-2 connected sites at US universities, we expected that these lookups should have been locally cached and would return quickly.

2.1 Name Lookups of CoDeeN Nodes

Our measurements showed that local DNS lookup times were generally good, but would often degrade dramatically, and that this instability behavior was widespread and frequent. The heartbeat monitoring process in CoDeeN performs one lookup per second of another CoDeeN node name, and given that CoDeeN runs on approximately 40 nodes, each name is queried roughly once every 40 seconds. These nodes have stable TTL values, with over half being one day or more, one quarter at six hours or longer, and none below an hour. So, most of these name lookups are highly cacheable, and given that local nameservers should be on the same campus as the CoDeeN node, we would expect most lookups to take a few milliseconds.

To illustrate the widespread nature of the problem and its magnitude, Figure 1 shows the lookup behavior over a recent two-day period across roughly one-fifth of the CoDeeN nodes. Each point indicates the average response time of the name lookups performed every minute. All the nodes in the graph show some sort of problems in DNS lookups for the period.

Node	Avg	Low	High	T-Low	T-High
cornell	531.7ms	82.4%	12.9%	0.5%	**99.2%**
harvard	99.4ms	92.3%	3.3%	0.7%	**97.9%**
cmu	24.0ms	81.9%	3.2%	8.3%	**71.0%**
ku	53.1ms	94.6%	1.8%	2.9%	**95.0%**
stanford	21.5ms	95.7%	1.3%	5.3%	**89.5%**
ubc	88.8ms	76.0%	7.6%	2.4%	**91.2%**
umich	43.6ms	96.7%	1.3%	2.4%	**96.1%**
northwestern	43.1ms	98.5%	0.5%	4.5%	**94.8%**

Table 1: Statistics over two days, Avg = Average, Low = Percentage of lookups < 10 ms, High = Percentage of lookups $>$ 100 ms, T-Low = Percentage of total low time, T-High = Percentage of total high time

Ping-response	Contribution(%)
No packet loss	88.9
Lose all packets	9.9
Lose one packet	1.1
Other	0.1

Table 2: Study of local nameserver status by ICMP ping

The types of problems indicated in these graphs are not consistent with simple configuration problems, but appear to be usage-induced or triggered by activity on the nameserver nodes. The Cornell node consistently shows DNS problems, with more than 20% of lookups showing high lookup times of over five seconds. We classify these lookups as failures, since five seconds is the time resolvers use to retry a request to another nameserver. The redundancy of the local nameserver configuration masks the failure of the first nameserver to respond. The 20% rate indicates that the first nameserver works most of the time, so its failure pattern is not consistent with a complete misconfiguration. Very often throughout the day, it simply stops responding, driving the per-minute average lookup time close to 5 seconds. The Harvard node also displays generally bad behavior. Further investigation discloses that most of the lookups are fine, but almost every minute, a couple of 5 seconds of delays appeared, which substantially influenced the per-minute average. The Stanford node shows periodic big spikes roughly every three hours. This phenomenon has persisted for the past few weeks, and we suspect the nameserver is being affected by heavy cron jobs. Lengthy DNS malfunction is noticed on the U-Michigan node, which lasted from 9:00 am to 10:30 am on Dec 12, indicating more than 1 second of average delay.

Although Figure 1 shows dramatic variation in DNS latency over time, the number of requests which fail and require retries is small. Figure 2(a) displays the cumulative distribution function (CDF) of name lookup times over the same two days. With the exception of the Cornell node, 90% of all requests take less than 100ms on all nodes, indicating that caching is effective and that latencies are quite low in the average case. On the Cornell node, over 80% of lookups are resolved within 6ms, also indicating that it works well most of the time.

However, Figure 2(b) shows the same data, but measured by the fraction of the total lookup time, instead of just the total number of lookups, and it indicates that **a small percentage of failure cases dominates the total time.** This weighted CDF shows, for example, that none of the nodes crosses the 0.5 value before 1000ms, indicating that more than 50% of the lookup time is spent on lookups taking more than 1000ms. The total lookup time is dominated by failure. If we assume that a well-behaving local nameserver can serve cached responses in 100ms, then the figures are even more dramatic. This data is shown in Table 1, and shows that most nodes have total times dominated by these slower lookups.

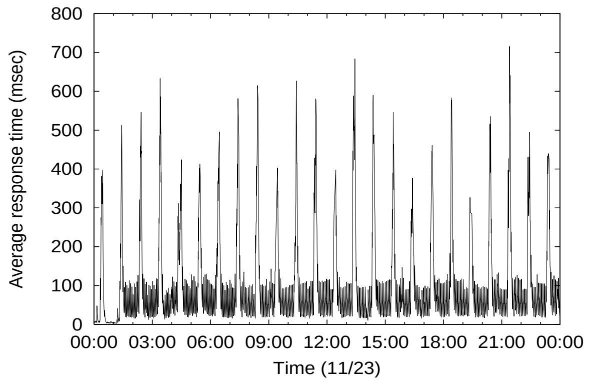
The implications of these measurements is significant – if we can reduce the amount of time spent on these longer cases, particular in the failures that require the local resolver to retry the request, we can dramatically reduce the total lookup times. Furthermore, given the sharp difference between "good" and "bad" lookups, we may also be able to ensure a more predictable (and hence less annoying) user experience.

One might assume this behavior stems from the node performing the measurement rather than the name service itself, but by comparing multiple nodes using the same nameserver, we can see if the observed behavior is consistent. Figure 3 and Figure 4 show the average times and failure rates for two pairs of nodes at two different PlanetLab sites. Nodes within each pair use the same nameserver, and as we can see, they see roughly the same behavior, including average lookup times, average failure rates, and timings of spikes/anomalies. We have noticed similar behavior on all sites we have tested, which suggests that the problem lies in the name service itself.

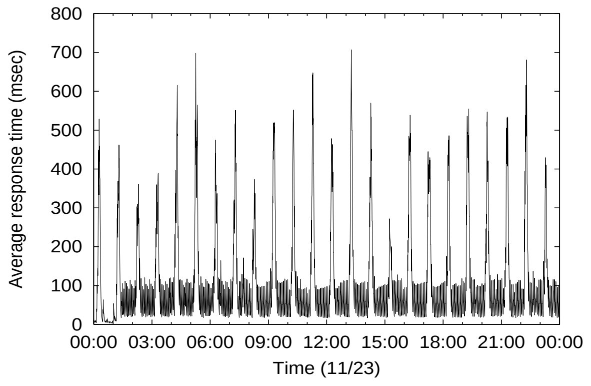
2.2 Failure characterization

To better understand the nature of these failures, we can employ techniques to investigate if other factors are involved, and if the failures present any behavior we can exploit in designing a system to address them.

One possible cause for these failures is network packet loss from the local node to the nameserver, since our queries are UDP-based, and packet loss would require application-generated retransmission. In order to test the correlation between packet loss and DNS failure, we perform an ICMP ping with 5 ping packets to the local nameserver after every DNS lookup failure across all the nodes for 12 hours. We show the result of 9471 such ping invocations due to DNS lookup failures in table 2. From table 2, we can see that most of the time when the nodes have local DNS lookup failures, they can reach the local name server without network packet loss. Sometimes

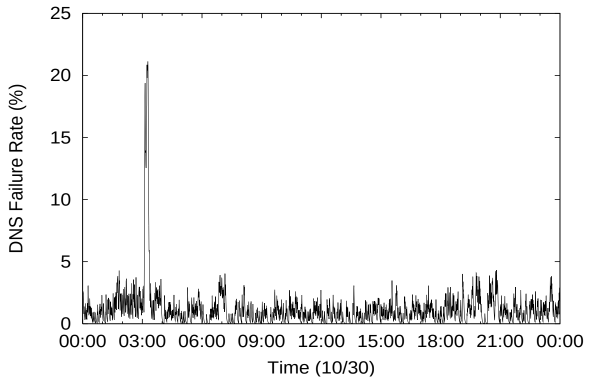


(a) purdue1

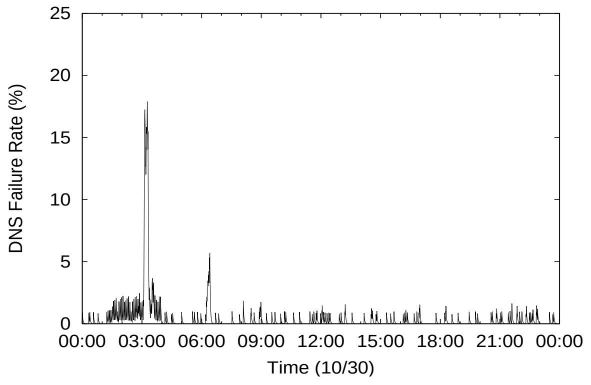


(b) purdue2

Figure 3: 11/23, Average Response Time of Two Nodes Sharing the Same Nameservers, planetlab1.cs.purdue.edu, planetlab2.cs.purdue.edu



(a) harvard1



(b) harvard2

Figure 4: 10/30, Failure Rates of Two Nodes Sharing the Same Nameservers, lefthand.eecs.harvard.edu, righthand.eecs.harvard.edu

the local nameserver is completely unreachable by ICMP ping, but some of these are due to filtering of ICMP pings at some sites. The low local network packet drop rate ($<$ 1.2% during periods of DNS failures) suggests that packet loss plays an insignificant role in the local DNS lookup failures in our environment. The fact that we have a non-trivial amount of DNS lookup failures leads us to believe that the local nameserver is the one which has the problem.

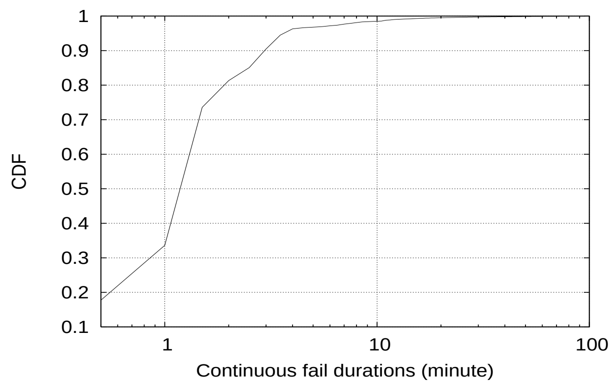
From the failures shown in figure 1, we can see that there are three kinds of failures:

1. Periodic failures: The regularity of these failures suggests that they are possibly caused by cron jobs running on the local nameserver. We can see them in the Stanford and Northwestern graphs.

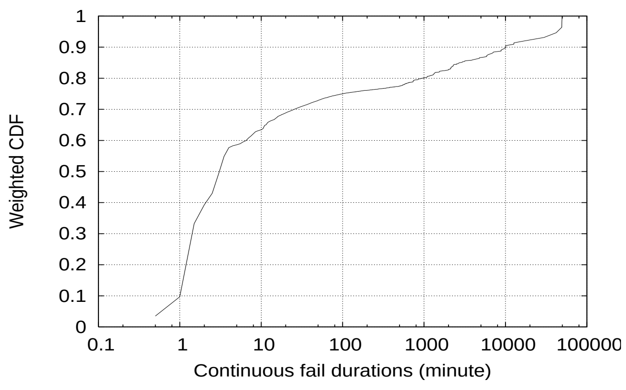
2. Long lasting continuous failures: They are possibly caused by local nameserver malfunctioning or extended overloading.

3. Sporadic short failures: They are likely caused by temporary overloading of the local name server.

To further understand how long the failures typically last, we conduct tests on 188 nodes on PlanetLab, including nodes not involved with CoDeeN. We perform a DNS lookup every second on every node, and recorded the DNS failure rate for every 30 seconds. The failure rate is a decaying average of about past 100 DNS lookups. We record all the periods which have failure rate greater than 1% and examine how long each failure period lasts. The data collected in the month of November 2003 is shown in Figure 5, where graph (a) is a CDF of durations counts, while (b) is a CDF of the total time spent in failure periods. We can see from the Figure 5 that although most of



(a) CDF



(b) Weighted CDF

Figure 5: CDF for distribution of failure duration

the failures have short duration, the longer failures dominate the total time of all the failures. This suggests that we need to be responsive to short time failures in order to be effective, but that we may also be able to perform optimizations to handle longer-term failures more effectively.

The measurements can also be used to answer the question of whether our deployment of CoDeeN on mostly US university sites generates representative DNS performance. The graphs in Figure 6 break down the 188 nodes by type, including US universities (edu), US companies (com), European PlanetLab sites, Asian PlanetLab sites, and others. We find that all categories show similar short-term failure behavior, with some differences in the lengths of longer-term failures. Interestingly, the US university sites have generally better performance, with much less time spent in longer-term DNS failure periods.

2.3 Correlation of the DNS lookup failures

While we have confirmed that nodes at the same site see similar DNS properties, given the global nature of PlanetLab and the number of multi-node experiments, another possible source of failure is the existence of experiments running across many nodes. To investigate this possibility, we study the correlation of DNS lookup failures over all the nodes we have. For every 30 seconds interval, we record how many nodes are "healthy" in terms of DNS performance. We define healthy as having a failure rate less than 1%, or less than 1.25 times the global average failure rate. The reason for the second test is that we have observed some of the DNS sites have problems with the authoritative servers resolving their names, so when these sites "disappear," all nodes experience DNS failure for those sites. Using the same 30 seconds failure rate data collected in the month of November 2003, we group by 1 hour intervals, and record the minimum, average and maximum number of nodes available within every hour. The percentage of healthy nodes (as a fraction of live nodes) is shown in Figure 7.

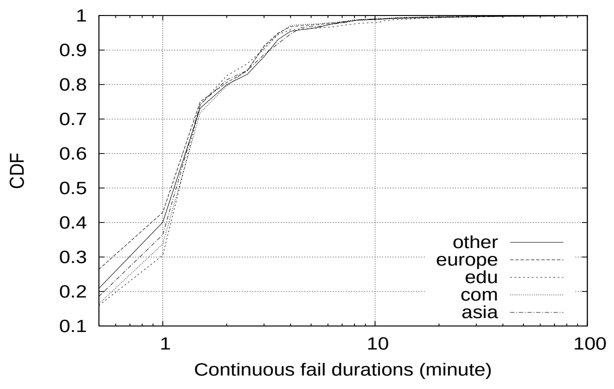
Software	PlanetLab	Packetfactory	TLD
BIND-4.9.3+/8	31.1%	36.4%	55.9%
BIND 9	48.9%	25.1%	34.0%
Other	20.0%	38.5%	10.1%

Table 3: Comparison of nameserver software used by planetlab, packetfactory survey and TLD survey

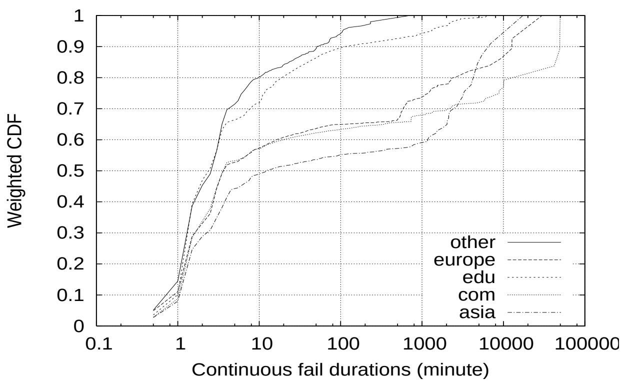
From this graph, we can see some minor correlation in failures, shown as downward spikes in the percentage of available nodes, but most of the variation in availability seems largely uncorrelated. An investigation into the spikes reveals that many nodes on PlanetLab have /etc/resolv.conf configured to use the same set of nameservers, especially those colocated at Internet2 facilities (not to be confused with Internet2-connected university sites). When these nameservers experience problems, the correlation appears large due to the number of nodes affected.

More important, however, is the observation that the fraction of healthy nameservers is always quite high, generally above 90%. **This observation provides the key insight for CoDNS – as long as we have a reasonable number of healthy nameservers, we can use them to mask locally-observed delays**.

We also survey the software running on the local nameservers used by the PlanetLab nodes (135 unique nameservers) with "chaos" class queries [11]. We find that they are mostly running a variety of BIND. We observe 11 different BIND 9 version strings, 13 different BIND 8 version strings and a number of humorous strings (which are included in "other") apparently set by the nameserver ad-



(a) CDF



(b) Weighted CDF

Figure 6: CDF for distribution of failure duration for different category

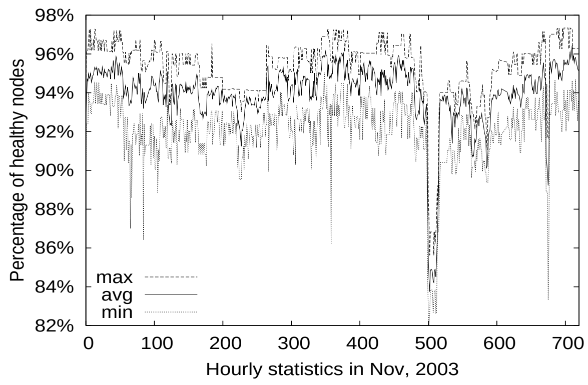


Figure 7: Hourly min/avg/max percentage of nodes with good NS

ministrators. We compare the nameserver software used in our test environment and other surveys in table 3 to ensure that the problems we observe on PlanetLab are not tied to a particular piece of software. The surveys we use for comparison are a global survey of 20,405 responding nameservers done by packetfactory in early 2003 [13] and a survey of TLD DNS software done by Brad Knowles in 2002 [8]. We find PlanetLab's diversity of nameservers is reasonable and generally in line with other measurements. We conclude that the failures are not likely to be specific to PlanetLab's choices of nameserver software.

3 Design

In this section, we discuss the design of a system which allows us to have a highly reliable and available name lookup service while minimizing the extra overhead. We discuss alternative approaches as well and compare the tradeoffs.

3.1 CoDNS

The main idea behind CoDNS is to forward name lookup queries to peer nodes when the local name service is experiencing a problem. Essentially, this strategy is applying a CDN approach to DNS – spreading the load among peers can improve the size and performance of the "global cache". Many of the same considerations used in CDN systems apply in this environment. We need to consider the proximity and availability of a node as well as the locality of the queries. A different consideration is that we need to decide when it is desirable to send remote queries. Given the fact that most name lookups are fast in the local nameserver, simply spreading the requests to peers might generate unnecessary traffic with no gain in latency. Worse, the extra load may cause marginal DNS nameservers to become overloaded. We investigate considerations for deciding when to send remote queries, how many peers to involve, and what sorts of gains to expect.

To precisely determine the effects of locality, load, and proximity is difficult, since we have no control over the nameservers and have little information about their workloads, configurations, etc. Given the fact that most CoDeeN nodes are connected via Internet2, we believe that proximity is less of a factor for us than other environments, due to the low round-trip times we experience between nodes. As long as we use relatively close nodes as peers, we should have reasonable behavior. We have observed coast-to-coast round-trip ping times of 80ms in CoDeeN, with regional times in the 20ms range. Likewise, locality is less of a factor in CoDNS than regular CDN systems, since we only use remote servers when our local server is experiencing problems. By process of elimination, the most interesting factor is the policy for determining when to begin using remote servers, and how

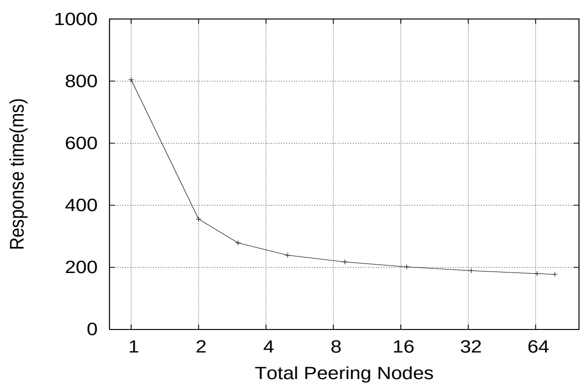


Figure 8: Average response time vs. number of nodes queried (200ms initial delay)

many to involve.

To understand the relationship between CoDNS response times, the number of peers involved, and the policies for determining when requests should be sent remotely, we collected live data from CoDeeN and use it to simulate various policies and their effects. Using one day's HTTP traffic on CoDeeN, we extract the 44486 unique hostnames requested by our users. We then replay this traffic using `gethostbyname()`, starting requests at the same time of day in the original logs. The replay happened one month after the data collections to avoid local nameserver caches which could skew the data. We perform this replay at 77 PlanetLab nodes with different nameservers. During this time, we also use application-level heartbeat measurements between all pairs of nodes to determine all-pairs round-trip latencies. Since all of the nodes are doing DNS lookups at about the same time, by adding the time spent in `gethostbyname()` at peerY to the time spent for the heartbeat from peerX to peerY, we will get the response time peerX can get if it asks peerY for a remote DNS lookup for the same hostname.

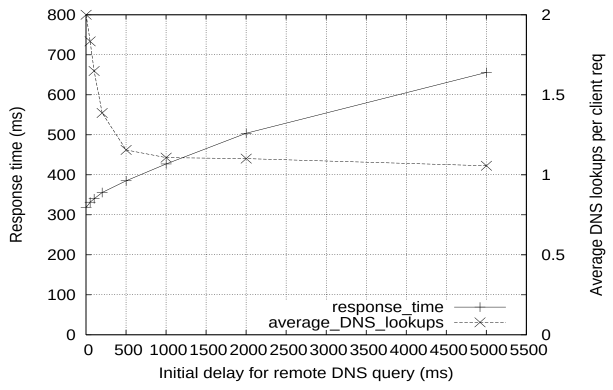


Figure 9: Response time and DNS lookups vs initial delay

The results of this experiment are shown in Figure 8, where the average response time is shown as a function of the number of peers performing lookups in parallel. We use an initial delay of 200ms for this figure which means that we will send out a DNS query to remote peer only if the local DNS lookup does not finish within 200ms. The X axis shows the total number of nodes queried (including the local nameserver) when we need to do a remote DNS lookup; the Y axis is the average response time in ms. This figure tells us that even if we ask just one peer (X = 2) when we need to issue a remote DNS query, we can reduce the average DNS lookup time more than half. Involving more peers in parallel provides only diminishing returns.

Another issue is the amount of extra DNS overhead needed to get good DNS response time. Since we want to minimize CoDNS's overhead, we want to have minimal amount of extra DNS lookups while maintaining reasonable response time performance. In Figure 9, we show a dual Y axes graph with both the average response time and average DNS lookups with regard to the initial delay. The X axis shows the different initial delay used for sending the first remote DNS query, the dual Y axes show the average response time and the average number of DNS lookup done per request by CoDNS. This graph shows that we can control the amount of extra DNS lookups by changing the initial delay for sending remote DNS query to peers. If we send the first remote query after a 500ms delay, we can still reduce the average DNS lookup time to about half while causing only about 15% extra DNS lookups to be done compared with not using any peers.

3.2 Other Approaches

We briefly describe other approaches to avoid local nameserver failure issues, and compare them with the CoDNS concept. One approach is to add the recursive DNS query ability into every local node. In the case of local nameserver failure, the local node will contact the root nameservers directly and try to resolve the hostname lookups by itself. This approach, compared to CoDNS, has several disadvantages:

1. This reduces the caching effectiveness because each node will be doing DNS lookups individually when the local nameserver fails. It defeats the purpose of using shared nameservers to allow caching among different local nodes. This approach will also cause more pressure on the global domain name system because of low cache utilization.

2. This increases the configuration efforts and also causes extra management problems. Instead of only configuring and managing a couple of local nameservers, the sysadmin will need to manage the name resolution subsystem on each individual node.

3. This will use more resources on each node. Since the name lookup service running on individual nodes are not the only services running, it will cause visible problems when other programs cause memory pressure for the name lookup service. Best practices suggests that name service is better run alone as a dedicated service.

Another possible solution is to make the resolver library on the local node act more aggressively. Instead of trying the second local name server after a 5 seconds timeout, it will use a shorter timeout value. We can also make the resolver library smart enough to detect the complete failure of the first nameserver and use the second local nameserver immediately. This solution has some disadvantages as well compare to CoDNS approach:

1. Many failures observed are caused by overload rather than network packet loss between the local node and local nameserver. With a more aggressive resolver, it will only aggravate the overload problem on the local nameserver.

2. By switching to the second nameserver immediately in case of first nameserver overload will not solve the problem completely since second nameserver will be overloaded as a result. The fact that most DNS lookups currently succeeded at the second nameservers is because they are not overloaded in the existing scheme. Once we aggressively shift the load to second nameserver when the first one is overloaded, the second nameserver will be overloaded as well.

3. The correlation graph in Figure 7 shows that the problems are local, not global. We should try to spread the load if possible rather than adding more load to the local nameservers.

4 Implementation

We have built a prototype of CoDNS and have been running it on all nodes on PlanetLab for roughly one month. During that time, we have been directing the CoDeeN CDN to use CoDNS for the name resolution.

CoDNS consists of a stand-alone daemon running on each node, accessible via UDP for remote queries, and loopback TCP for locally-originated name lookups. The daemon is event-driven, and is implemented as a non-blocking master process and many (blocking) slave processes. The master receives name lookup requests from local clients and remote peers, and hands them over to one of its idle slaves. The slave process resolves those names by calling `gethostbyname()` and sends the result back to the master. Then, the master sends the final result to either a local client or a remote peer depending on where it originated. Preference for idle slaves is given to locally-originated requests over remote queries.

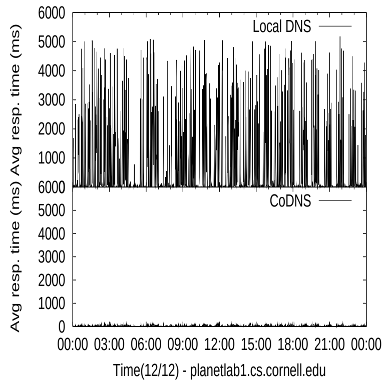
The master process records each request's arrival time from local clients and sends a UDP name lookup query to a peer node when the response from the slave has not returned within a certain period. This delay is used as a boundary for deciding if the local nameserver is slow. In the event that neither the local nameserver or the remote node has responded, CoDNS doubles the delay value before sending the next remote query to another peer. In the process, whichever result that comes first will be delivered as the response for the name lookup to the client. Peers may silently drop remote queries if they are overloaded, and remote queries that fail to resolve are also discarded. Slaves may also intentionally add delay if they receive a locally-generated request that fails to resolve, with the hope that remote nodes may be able to resolve such names.

4.1 Remote query initiation & retries

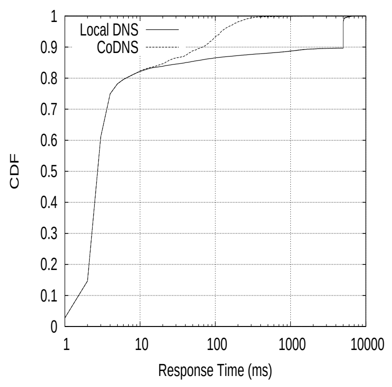
The initial delay before sending the first remote query is dynamically adjusted based on the recent performance of local name servers and peer responses. The guiding concept is that when the local nameserver performs well, we increase the delay so that fewer remote queries are sent. When most remote answers beat the local ones, we reduce the delay preferring the remote source. Specifically, when the past 32 name lookups are all resolved locally without using any remote queries, then the initial delay is set to 200 ms. Considering the fact that the average response time on a well-functioning node is about 130 - 150 ms, 200 ms delay should respond fast when the temporal instability kicks in, while wasting minimal amount of extra remote queries. However, to respond quickly to local nameserver failure, if the remote query wins more than 50% of the last 16 requests, then the delay is set to 0 ms. That is, the remote query is sent immediately as the request arrives. Our test result shows it is rare not to have failure when more than 8 out of 16 requests take more than 300ms to resolve, so we think it is reasonable to believe the local nameserver is having a problem in that case. Once the immediate query is sent, the delay is set to the average response time of remote query responses plus one standard deviation.

4.2 Proximity, Locality and Availability

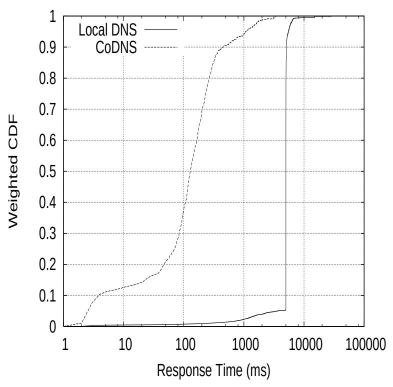
Each CoDNS node gathers and manages a set of neighbor nodes within a reasonable latency boundary. Among these neighbor nodes, one peer is chosen using HRW [15] for each remote name lookup. Liveness of peer nodes is periodically checked to see if the service is available. When CoDNS starts, it sends a heartbeat to each node in the node list every second. The heartbeat response contains the round trip time and average response time of the local DNS at the peer node. Currently, if the sum of these is less



(a) Average Response Time

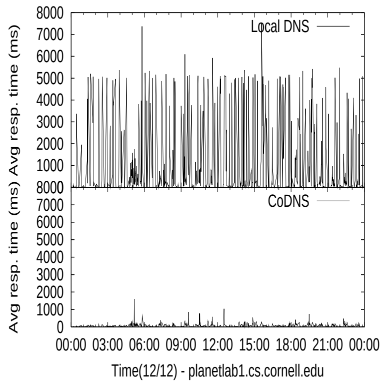


(b) Percentage of lookups taking $<$ x ms

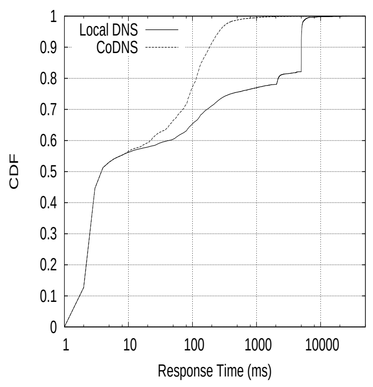


(c) Percentage of the sum of all lookups taking $<$ x ms

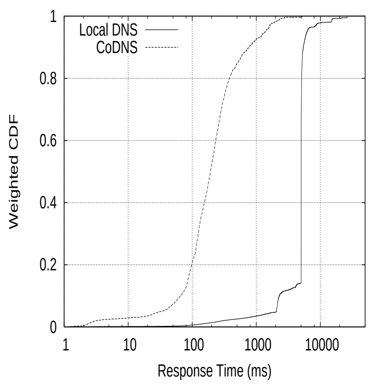
Figure 10: 12/12, Average response time, CDF, Weighted CDF for CoDeeN Host Names Lookup on planetlab1.cs.cornell.edu



(a) Average Response Time



(b) Percentage of lookups taking $<$ x ms



(c) Percentage of the sum of all lookups taking $<$ x ms

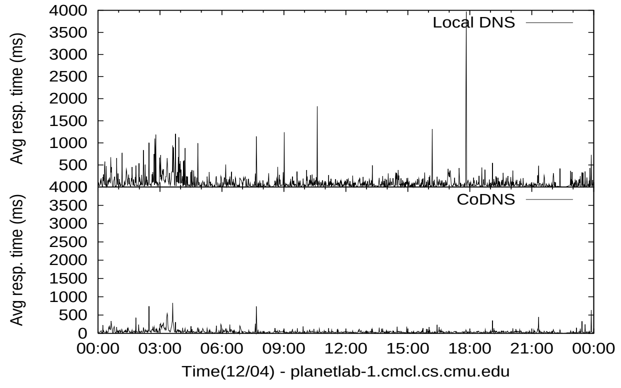
Figure 11: 12/12, Average response time, CDF, Weighted CDF for Real Traffic on planetlab1.cs.cornell.edu

than 90 ms, it is chosen as a neighbor. As soon as CoDNS fills up 30 such nodes, it stops scanning the list and only monitors the nodes in the neighbor set from then on. If it couldn't find enough neighbors, the latency boundary is increased a little bit and scanning is repeated. The default latency boundary and the number of minimum neighbors are configurable according to the distribution of nodes.

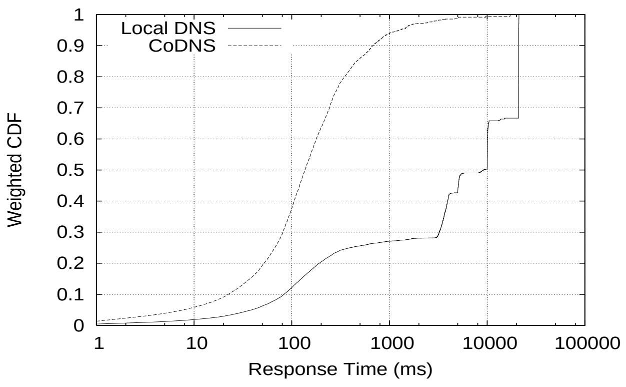
Given that neighbors are relatively "soft" state, CoDNS does not expend excessive effort guaranteeing their availability. After having found enough neighbors, it monitors the liveness of each node by sending the heartbeat every 30 seconds. If the heartbeat does not arrive within a short time period, then the node is excluded in the next remote query selection. Periodically, the dead nodes are updated with fresh ones by partial scanning of the node list. The node selection is done in the neighbor set by a scheme similar to the one used in CoDeeN [15]. By consistently choosing the same node for the same name, we can expect a certain amount of request locality.

5 Results

To gauge the effectiveness of CoDNS, we compare its behavior with local DNS on the CoDeeN traffic using a variety of metrics. To eliminate the caching effect on a nameserver from other users sharing the same server, we measure both times only in CoDNS, using the slaves to indicate local DNS performance. By comparing the local name lookups versus all CoDNS responses, we isolate the effects the name lookup process without involving the other machinery of CoDeeN. To separate the information on real traffic versus the heartbeat traffic generated by CoDeeN, we modify the CoDeeN proxies to indicate the

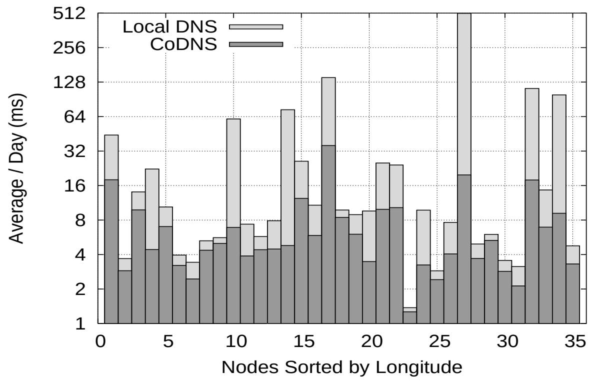


(a) Average Response time

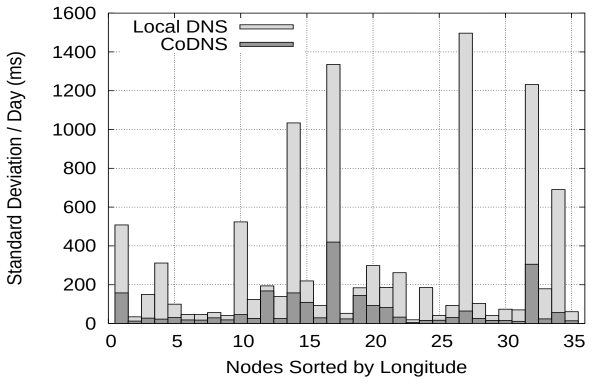


(b) Percentage of the sum of all lookups taking $<$ x ms

Figure 12: 12/04, Real Traffic on planetlab-1.cmcl.cs.cmu.edu



(a) Average Response Time over 12/11



(b) Standard Deviation over 12/11

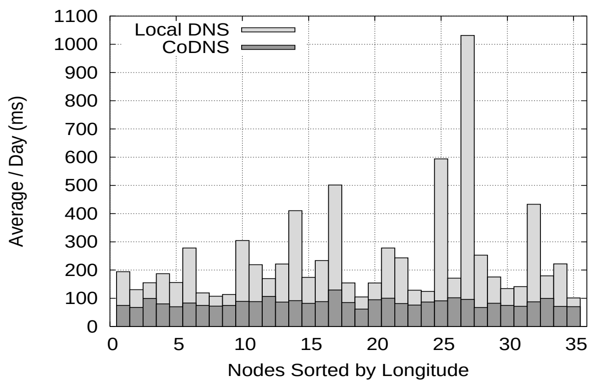
Figure 13: Average Response Time and Standard Deviation for CoDeeN Host Names Lookup

intent of the lookup for measurement purposes.

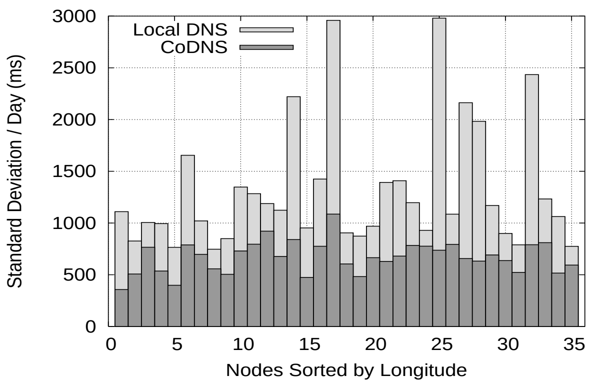
For both real lookups and CoDeeN heartbeat lookups, we observe that CoDNS effectively flattens out the spikes and produces response time comparable to a well-functioning local nameserver. The (a) graphs in Figure 10 and Figure 11 show the average response times for each case on planetlab1.cs.cornell.edu on Dec 12, 2003. Average response time is calculated for each minute and displayed as one point as before. The daily average response time was reduced from 554 ms in local DNS to 21 ms in CoDNS for CoDeeN host name lookups, and from 1095 ms to 79 ms for real traffic. However, in the real traffic graph, CoDNS also shows a few spikes in the response time. In particular, at 05:10 and 12:31, the per-minute average in CoDNS exceeds 1 second. But in both cases, there came a few non-existent name lookup requests which skewed the average. Finding the names non-existent took about 20 seconds in each case. We have observed three types of failures that prevent CoDNS from providing benefit:

1. The name is non-existent.
2. If the local nameserver is bad at the start phase of gathering neighbor list.
3. When network problems prevent CoDNS from contacting the peer node.

Where CoDNS wins is where the local nameserver fails, and its demonstrated benefits are quite considerable. The (b) and (c) graphs in the both figures show CDFs and weighted CDFs for name lookups. In both cases, name lookups taking more than 100ms dominate the total lookup time. The CoDNS CDFs show a much smoother



(a) Average Response Time over 12/11



(b) Standard Deviation over 12/11

Figure 14: Average Response Time and Standard Deviation for Real Traffic

transition after 100ms, which is due to the peer nodes providing replies when the local nameserver is failing. In the local nameserver, replies tend to be bimodal – either very quick or very slow. The weighted versions reveals dramatic difference in total amount of time spent on lookups. Because most resolvable failures are effectively removed in CoDNS, the time spent for lookups $>$ 1000ms is less than 10% of total time, whereas in the local nameserver, these consume more than 90% of the time.

5.1 Average response time and Standard Deviation

The benefits of CoDNS are noticed on most CoDeeN nodes. Figure 12 show the similar result measured on another CoDeeN node in case of the real traffic. Despite having much better initial behavior than the Cornell node, even it sees substantial savings in total lookup times.

A similar graph for every CoDeeN node would clearly require too much space, but we can observe the benefits by comparing the average response times and their standard deviations across nodes. This data for one day's traffic is shown in Figure 13 and Figure 14. Included is the real lookup traffic as well as CoDeeN's heartbeat traffic. Generally, the graph shows CoDNS gives reliable service over time, much smaller variation than local DNS nameservers. The average of all nodes in CoDNS is 7 ms for CoDeeN node names, and 84 ms for real traffic, in contrast to 37 ms and 237 ms, respectively, using local DNS. Especially in real traffic, the performance of CoDNS is stable over the CoDeeN nodes regardless of local nameserver fluctuation. Also, standard deviations range from 16% to 75% lower than that of local DNS, implying much less fluctuations in response time.

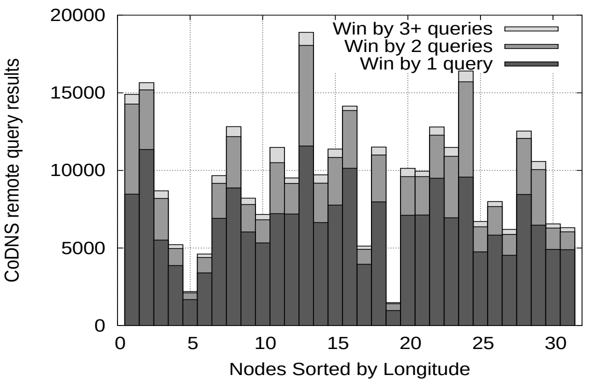


Figure 15: Analysis for multiple remote DNS queries

5.2 Analysis of CoDNS results

We study the effectiveness of our CoDNS strategy, particularly the conditions in which it beats the local DNS nameserver, using one week of log gathered through the real traffic CoDNS deployment. The traffic is generated from the normal daily CoDeeN HTTP traffic. We find that under real traffic, CoDNS uses the remote peers 18.9% of all lookups. For the rest of the time, local DNS is fast enough so that CoDNS does not send any remote queries. And for all the remote queries sent, 34.6% of the remote queries "win" by returning a valid DNS response faster than the local DNS lookup.

We further study the effect of querying multiple remote peers in CoDNS retransmission. In Figure 15, we see the distribution of "winning" the local DNS lookups by sending out one remote query, two remote queries and three or more remote queries. From the figure, we can see that in most of the cases, the CoDNS strategy will win within two remote queries if it eventually wins.

5.3 Overhead

There are two types of overheads involved in the design:
1. Extra DNS lookups done per user requests in CoDNS system as compared to the original system. From the analysis done in Section 3.1, we have shown that the extra amount of DNS lookups can be controlled by setting different initial delay time. By setting the initial delay time to 500ms, extra amount of DNS lookups would be reduced to only 10% of all user requests. In the real CoDNS deployment, we find that we only send out about 18.9% of extra traffic even when we are using an aggressive algorithm to optimize the response time.
2. Extra network traffic generated for remote DNS queries and heartbeats between CoDNS peers. For the amount of network traffic generated, we can compare the total amount of traffic used by CoDNS with the total amount of network traffic used by CoDeeN. All the traffic used by remote peer query and heartbeats only takes about 520MB per day across all the nodes, while the daily CoDeeN traffic takes about average of 150GB of traffic. This is easily explained by the observation that most server lookups will result in multiple object fetches, so the cost of each DNS/CoDNS lookup is heavily amortized. In relative terms, the amount of CoDNS traffic only add an overhead of 0.3% onto the CoDeeN traffic it supports.

6 Deployment

CoDNS currently runs as a standalone daemon process operating across PlanetLab. Its use is not transparent, and we had to make minimal changes to enable CoDeeN to use CoDNS for name lookups instead of using regular DNS servers.

We are currently adding support for automatic CoDNS use into an existing nameserver, Dan Bernstein's DNSCache [5]. This system is believe to be more secure than BIND, and since we are interested primarily in name lookups, Bernstein's approach of splitting lookup and authoritative service into two separate programs is appealing for us. Our approach involves running a local DNSCache program which has been modified to use CoDNS. Traffic is automatically directed to this program by adding an entry for localhost into the /etc/resolv.conf file. The one complication in this scheme is that the CoDNS slaves use `gethostbyname()` to resolve names, which in turn will use the contents of resolv.conf, causing a loop. We solve this problem by statically linking the CoDNS slaves with a modified version of `gethostbyname()` that ignores the localhost entry in resolv.conf. We hope to begin deploying this transparent CoDNS support on PlanetLab early next year.

7 Related work

Ever since the seminal study of DNS measurement in 1992 by Danzig [2], the performance of DNS has been greatly improved and the implementation bugs have been largely reduced [9]. However, recent measurements show there is still much room for improvement. Wills [17] indicates 29% of DNS lookups take over 2 seconds, and Cohen [3] shows 10% of lookups exceed more than 3 seconds. Jung [6] also presents data indicating 23% of all name lookups end up with receiving no results, identifying improper configurations and incorrect nameservers as culprits. It further describes that the effectiveness of caching is limited by the long-tailed nature of the name distribution [1], so having short TTLs does not add much network traffic. Liston[10] investigates the correlations between DNS lookups and the locations, and found out the mean response time of a name lookup has much variation depending on the location. However, the difference in our study from all these is while all of them focused on the general DNS lookup process involving other sets of nameservers in WAN environment, none of them suspect the problems incurred by the local nameserver and the LAN environment. Our study shows the possibility of the name lookup time being greatly skewed by the temporary instability of the local nameservers and possibly by other activities affecting the nameserver.

Cox [4] investigates the possibility of transforming DNS into a peer-to-peer system [14] using a distributed hash table. The idea is to replace the hierarchical DNS name resolving process with a flat peer-to-peer query style, in pursuit of load balancing and robustness. With this design, the misconfigurations from mistakes by administrators can be eliminated and the traffic bottleneck concentrating on the root servers are removed so that the load is distributed over the entities joining the system. The difference in our approach is to temporarily use functioning nameservers of peer nodes, separate from the issue of improving the DNS infrastructure itself. We expect that any benefits in improving the infrastructure “from above” will be complementary to our “bottom up” approach. One advantage of our system is that misconfigurations can be masked without outage of the name service, while triggering an alarm when the misconfiguration is suspected should leave the minimal work for the administrators to casually examine.

Kangasharju [7] mentions another approach to reducing the DNS lookup latency by replicating DNS information. Inspired by the fact the entire DNS record database fits into the size of a typical hard disk and recent emerging of terrestrial multicast and satellite broadcast systems, this scheme reduces the need to query the distant nameserver by keeping the DNS information up-to-date throughout the world by efficient replication.

8 Conclusion

In this paper we have shown the instability in DNS name lookups is widespread and relatively common. We demonstrated that the problems are not caused by the nodes performing the lookups but by the local name service itself. Even though the failure cases are a small fraction of all lookups, their longer delays result in most lookup time being spent on failures.

We also showed that these local failures appear to be uncorrelated, and not specific to our test environment. Using this information, we develop an alternative lookup system, CoDNS, which benefits by harnessing the reliability of peer nodes. CoDNS reduces the total time consumption in name lookups as well as improving the mean latency with minimal extra overhead. It provides high availability and reduces variability, all while generating very low levels of overhead.

References

[1] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *Proceedings of INFOCOM '99*, pages 126–134, 1999.

[2] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical internet object cache. In *USENIX Annual Technical Conference*, pages 153–164, 1996.

[3] E. Cohen and H. Kaplan. Prefetching the means for document transfer: A new approach for reducing web latency. In *Proceedings of INFOCOM '00*, pages 854–863, 2000.

[4] R. Cox, A. Muthitacharoen, and R. Morris. Serving DNS Using Chord. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, 2002.

[5] D. J. Bernstein. djbdns. http://tinydns.org/.

[6] J. Jung, E. Sit, H. Balakrishnan, and R. Morris. DNS performance and the effectiveness of caching. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop*, 2001.

[7] J. Kangasharju and K. W. Ross. A replicated architecture for the domain name system. In *Proceedings of INFOCOM '00*, pages 660–669, 2000.

[8] B. Knowles. Domain Name Server Comparison: BIND 8 vs. BIND 9 vs. djbdns vs. ???, 2002. http://www.usenix.org/events/lisa02/tech/presentations/knowles_ppt/.

[9] A. Kumar, J. Postel, C. Neuman, P. Danzig, and S. Miller. Common DNS Implementation Errors and Suggested Fixes, October 1993. http://www.faqs.org/rfcs/rfc1536.html.

[10] R. Liston, S. Srinivasan, and E. Zegura. Diversity in DNS Performance Measures. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop '02*, 2002.

[11] P. Mockapetris. Domain names - implementation and specification, November 1987. http://www.faqs.org/rfcs/rfc1035.html.

[12] PlanetLab. http://www.planet-lab.org.

[13] M. Schiffman. A samping of the security posture of the internet's dns servers. http://www.packetfactory.net/papers/DNS-posture/.

[14] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.

[15] L. Wang, V. Pai, and L. Peterson. The Effectiveness of Request Redirecion on CDN Robustness. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, Boston, MA, December 2002.

[16] L. Wang, K. Park, R. Pang, V. Pai, and L. Peterson. Reliability and security in the CoDeeN content distribution network. In *USENIX Annual Technical Conference*, 2004.

[17] C. E. Wills and H. Shang. The contribution of DNS lookup costs to Web object retrieval. Technical Report WPI-CS-TR-00-12, 2000.