CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups

KyoungSoo Park, Vivek S. Pai, Larry Peterson and Zhe Wang
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. Most DNS research has focused on server-side infrastructure, with the assumption that the aggressive caching and redundancy on the client side are sufficient. However, through systematic monitoring, we find that client-side DNS failures are widespread and frequent, degrading DNS performance and reliability.

We introduce CoDNS, a lightweight, cooperative DNS lookup service that can be independently and incrementally deployed to augment existing nameservers. It uses a locality and proximity-aware design to distribute DNS requests, and achieves low-latency, low-overhead name resolution, even in the presence of local DNS nameserver delay/failure. Using live traffic, we show that CoDNS reduces average lookup latency by 27-82%, greatly reduces slow lookups, and improves DNS availability by an additional '9'. We also show that a widely-deployed service using CoDNS gains increased capacity, higher reliability, and faster start times.

1 Introduction

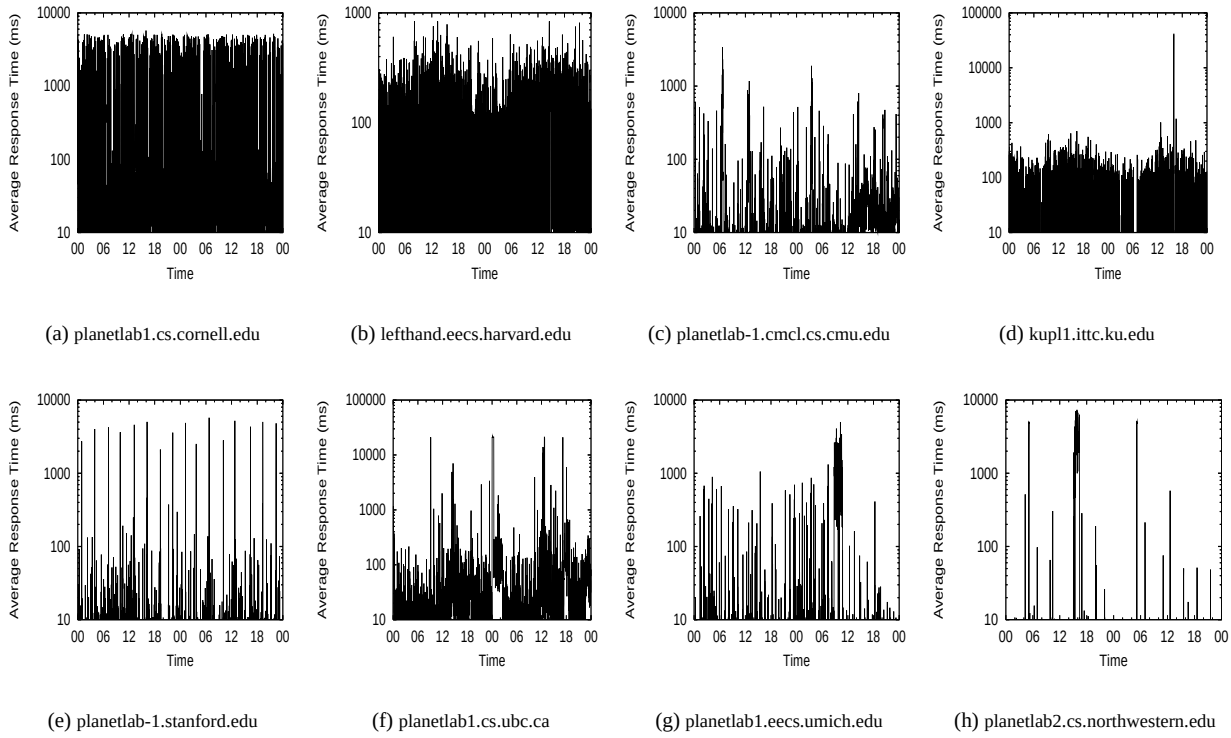
The Domain Name System (DNS) [15] has become a ubiquitous part of everyday computing due to its effectiveness, human-friendliness, and scalability. It provides a distributed lookup service primarily used to convert from human-readable machine names to Internet Protocol (IP) addresses. Its existence has 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, including researchers.

Most DNS research focuses on "server-side" problems, which arise on the systems that translate names belonging to the group that runs them. Such problems include understanding name hierarchy misconfiguration [5, 9] and devising more scalable distribution infrastructure [4, 10, 18]. However, due to increasing memory sizes and DNS's high cachability, "client-side" DNS hit rates are approaching 90% [9, 24], so fewer requests are dependent on server-side performance. The client-side components are responsible for contacting the appropriate servers, if necessary, to resolve any name presented by the user. This infrastructure, which has received less attention, is our focus – understanding client-side behavior in order to improve overall DNS performance and reliability.

Using PlanetLab [16], a wide-area distributed testbed, we locally monitor the client-side DNS infrastructure of 150 sites around the world, generating a large-scale examination of client-side DNS performance. We find that client-side failures are widespread and frequent, and that their effects degrade DNS performance and reliability. The most common problems we observe are intermittent failures to receive any response from the local nameservers, but these are generally hidden by the internal redundancy in DNS deployments. However, the cost of such redundancy is additional delay, and we find that the delays induced through such failures often dominate the time spent waiting on DNS lookups.

To address these client-side problems, we have developed CoDNS, a lightweight, cooperative DNS lookup service that can be independently and incrementally deployed to augment existing nameservers. CoDNS uses an insurance-like model of operation – groups of mutually trusting nodes agree to resolve each other's queries when their local infrastructure is failing. We find that the group size does not need to be large to provide substantial benefits – groups of size 2 provide roughly half the maximum possible benefit, and groups of size 10 achieve almost all of the possible benefit. Using locality-enhancement techniques and proximity optimizations, CoDNS achieves low-latency, low-overhead name resolution, even in the presence of local DNS delays/failures.

CoDNS has been serving live traffic on PlanetLab since October 2003, providing many benefits over standard DNS. CoDNS reduces average lookup latency by 27-82%, greatly reduces slow lookups, and improves DNS availability by an extra '9', from 99% to over 99.9%. Its service is more reliable and consistent than any individual node's. Additionally, CoDNS has salvaged "unusable" nodes, which had such poor local DNS infrastructure that they were unfit for normal use. Applications using CoDNS often have faster and more predictable start times, improving availability.



(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: Average cached DNS lookup response times on various PlanetLab nodes over two days. Note that while most Y axes span 10-1000 milliseconds, some are as large as 100,000 milliseconds.

2 Background & Analysis

While the Domain Name System (DNS) was intended to be a scalable, distributed means of performing name-to-IP mappings, its flexible design has allowed it to grow far beyond its original goals. While most people would be familiar with it for Web browsing, many systems depend on fast and consistent DNS performance. Mail servers, Web proxy servers, and content distribution networks (CDNs) must all resolve hundreds or even thousands of DNS names in short periods of time, and a failure in DNS may cause a service failure, rather just delays.

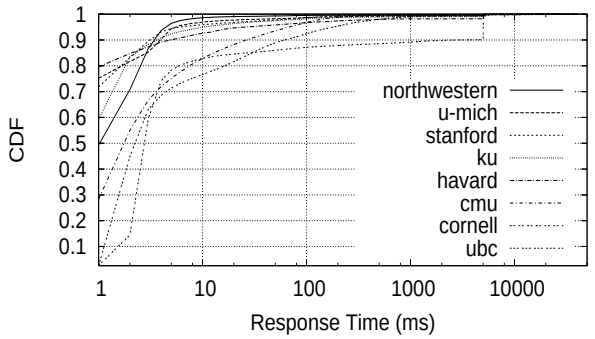
The server-side infrastructure of DNS consists of hierarchically-organized name servers, with central authorities providing "root" servers and others delegated organizations handling "top-level" servers, such as ".com" and ".edu". Domain name owners are responsible for providing servers that handle queries for their names. While DNS users can manually query each level of the hierarchy in turn until the complete name has been resolved, most systems delegate this task to local nameserver machines. This approach has performance advantages (e.g., caching replies, consolidating requests) as well as management benefits (e.g., fewer machines to update with new software or root server lists).

With local nameserver cache hit rates approaching 90% [9, 24], their performance impact can eclipse that of the server-side DNS infrastructure. However, local nameserver performance and reliability has not been well studied, and since it handles all DNS lookups for clients, its failure can disable other systems. Our experiences with building the CoDeeN content distribution network, running on over 100 PlanetLab nodes [23], motivated us to investigate this issue, since all CoDeeN nodes use the local nameservers at their hosting sites.

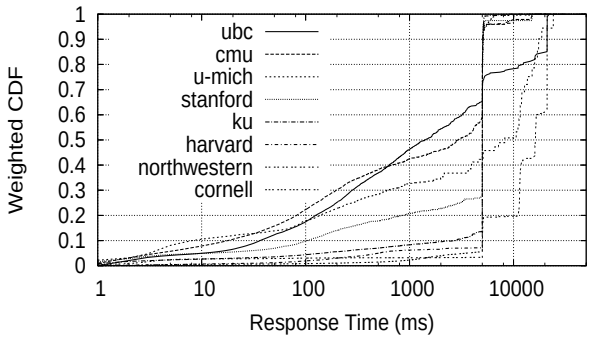
2.1 Frequency of Name Lookup Failures

To determine the failure properties of local DNS infrastructure, we systematically measure DNS lookup times on many PlanetLab nodes. In particular, across 40 North American sites, we perform a query once per second. We ask these nodes to resolve each other's names, all of which are cacheable, with long time-to-live (TTL) values of no less than 6 hours. Lookup times for these requests should be minimal, on the order of a few milliseconds, since they can be served from the local nameserver's cache. This diagnostic workload is chosen precisely because it is trivially cacheable, making local infrastructure failures more visible and quantifiable. Evaluation of DNS performance on live traffic, with and without CoDNS, is covered in Section 5.

Our measurements show that local DNS lookup times are generally good, but often degrade dramatically, and that this instability is widespread and frequent. To illus-



(a) Fraction of Lookups Taking $<$ X ms



(b) Fraction of the Sum of Lookups Taking $<$ X ms

Figure 2: Cumulative Distribution of Cached DNS Lookups

trate the widespread nature of the problem and its magnitude, Figure 1 shows the lookup behavior over a two-day period across a number of PlanetLab nodes. Each point shows the per-minute average response time of name lookups. All the nodes in the graph show some sort of problems in DNS lookups during the period, with lookups often taking thousands of milliseconds.

These problems are not consistent with simple configuration problems, but appear to be usage-induced or triggered by activity on the nameserver nodes. For example, the Cornell node consistently shows DNS problems, with more than 20% of lookups showing high lookup times of over five seconds, the default timeout in the client's resolver library. These failed lookups are eventually retried at the campus's second nameserver, masking the first nameserver's failures. Since the first nameserver responds to 80% of queries in a timely manner, it is not completely misconfigured. Very often throughout the day, it simply stops responding, driving the per-minute average lookup times close to five seconds. The Harvard node also displays generally bad behavior. While most lookups are fine, a few failed requests every minute substantially increase the per-minute average. The Stanford node's graph shows periodic spikes roughly every three hours. This phenomenon is long-term, and we suspect the nameserver is being affected by heavy cron jobs. The Michigan node shows a 90 minute DNS problem, driving its generally low lookup times to above one second.

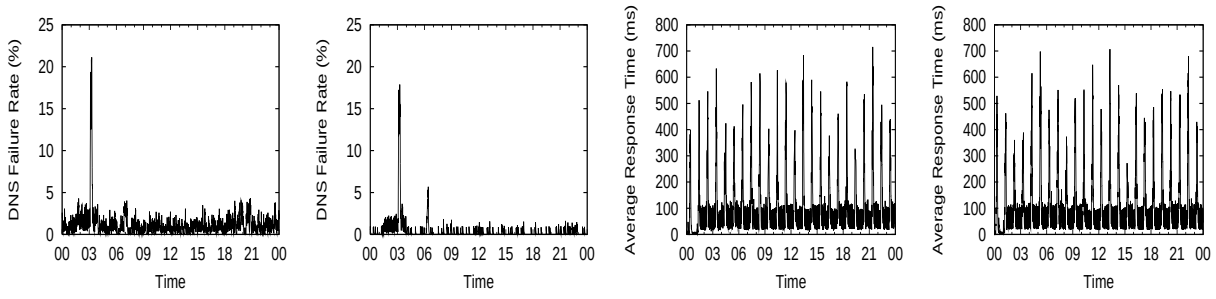
Although the average lookup times appear quite high at times, the individual lookups are mostly fast, with a few very slow lookups dominating the averages. Figure 2(a) displays the cumulative distribution function (CDF) of name lookup times over the same two days. With the exception of the Cornell node, 90% of all requests take less than 100ms on all nodes, indicating that caching is effective and that avaerage-case latencies are quite low. Even the Cornell node works well most of the time, with over 80% of lookups are resolved within 6ms.

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

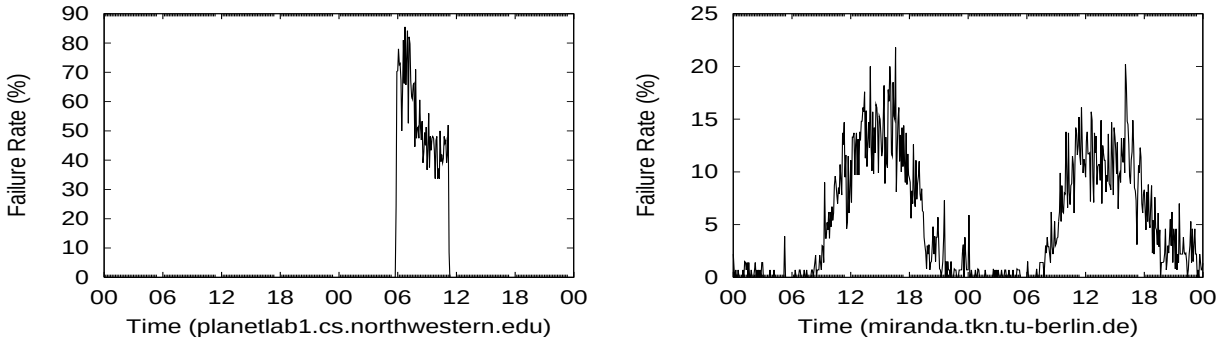
However, slow lookups dominate the total time spent waiting on DNS, and are large enough to be noticeable by end users. In Figure 2(b), we see the lookups shown by their contribution to the total lookup time, which indicates that **a small percentage of failure cases dominates the total time.** This 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. If we assume that a well-behaving local nameserver can serve cached responses in 100ms, then the figures are even more dramatic. This data, shown in Table 1, shows that slow lookups comprise most of the lookup time.

These measurements show that **client-side DNS infrastructure problems are significant** and need to be addressed. If we can reduce the amount of time spent on these longer cases, particularly 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. Finally, it is worth noting that these problems are not an artifact of PlanetLab – in all cases, we use the site's local nameservers, on which hundreds or thousands of other non-PlanetLab



(a) lefthand.eecs.harvard.edu (b) righthand.eecs.harvard.edu (c) planetlab1.cs.purdue.edu (d) planetlab2.cs.purdue.edu

Figure 3: All nodes at a site see similar local DNS behavior, despite different workloads at the nodes. Shown above are one day's failure rates at Harvard, and one day's response times at Purdue.



(a) northwestern-1 (b) tu-berlin

Figure 4: Failures seemingly caused by nameserver overload – in both cases, the failure rate is always less than 100%, indicating that the server is operational, but performing poorly.

machines depend. The PlanetLab nodes at a site see similar lookup times and failure rates, despite the fact that their other workloads may be very different. Examples from two sites are shown in Figure 3, and we can see that the nodes at a site see similar DNS performance. This observation further enhances our claim that the problems are site-wide, and not PlanetLab-specific.

2.2 Origins of the Client-Side Failures

While we do not have full access to all of the client-side infrastructure, we can try to infer the reasons for the kinds of failures we are seeing and understand their impact on lookup behavior. Absolute confirmation of the failure origins would require direct access to the nameservers, routers, and switches at the sites, which we do not have. Using various techniques, we can trace some problems to packet loss, nameserver overloading, resource competition and maintenance issues. We discuss these below.

Packet Loss – The simplest cause we can guess is the packet loss in the LAN environment. Most nameservers communicate using UDP, so even a single packet loss either as a request or as a response would eventually trigger a query retransmission from the resolver. The resolver's default timeout for retransmission is five seconds, which matches some of the spikes in Figure 1.

Packet loss rates in LAN environments are generally assumed to be minimal, and our measurements of Princeton's LAN support this assumption. We saw no packet loss at two hops, 0.02% loss at three hops, and 0.09% at four hops. Though we did see bursty behavior in the loss rate, where the loss rates would stay high for a minute at a time, we do not see enough losses to account for the DNS failures. Our measurements show that 90% of PlanetLab nodes have a nameserver within 4 hops, and 70% are within 2 hops. However, other contexts, such as cable modems or dial-up services, have more hops [20], and may have higher loss rates.

Nameserver overloading – Since most request packets are likely to reach the nameserver, our next possible culprit is the nameserver itself. To understand their behavior, we asked all nameservers on PlanetLab to resolve a local name once every two seconds and we measured the results. For example, on planetlab-1.cs.princeton.edu, we asked for planetlab-2.cs.princeton.edu's IP address.

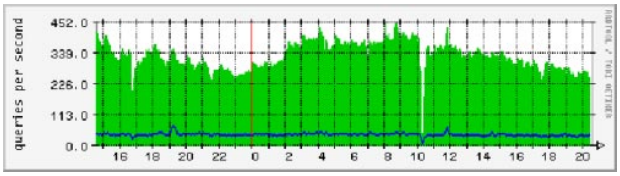


Figure 5: Daily Request Rate for Princeton.EDU

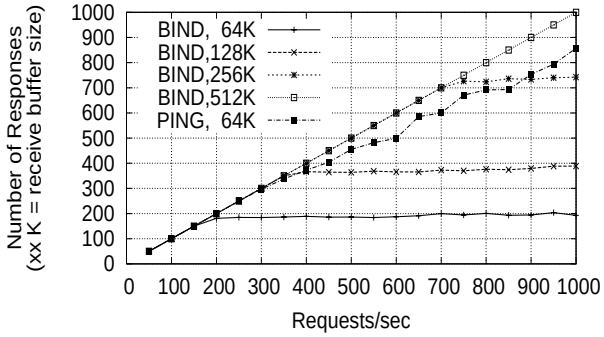


Figure 6: BIND 9.2.3 vs. PING with bursty traffic

In addition to the possibility of caching, the local nameserver is mostly likely the authoritative nameserver for the queried name, or at least the authoritative server can be found on the same local network.

In Figure 4, we see some evidence that nameservers can be temporarily overloaded. These graphs cover two days of traffic, and show the 5-minute average failure rate, where a failure is either a response taking more than five seconds, or no response at all. In Figure 4(a), the node experiences no failures most of time but a 30% to 80% failure rate for about five hours. Figure 4(b) reveals a site where failures start during the start of the workday, gradually increase, and drop starting in the evening. It is reasonable to assume that human activity increases in these hours, and affects the failure rate.

We suspect that a possible factor in this overloading is the UDP receive buffer on the nameserver. These buffers are typically sized in the range of 32-64KB, and incoming packets are silently dropped when this buffer is full. If the same buffer is also used to receive the responses from other nameservers, as the BIND nameserver does, this problem gets worse. Assuming a 64KB receive buffer, a 64 byte query, and a 300 byte response, more than 250 simultaneous queries can cause packet dropping. In Figure 5, we see the request rate (averaged over 5 minutes) for the authoritative nameserver for princeton.edu. Even with smoothing, the request rates are in the range of 250-400 reqs/sec, and we can expect that instantaneous rates are even higher. So, any activity that causes a 1-2 second delay of the server can cause requests to be dropped.

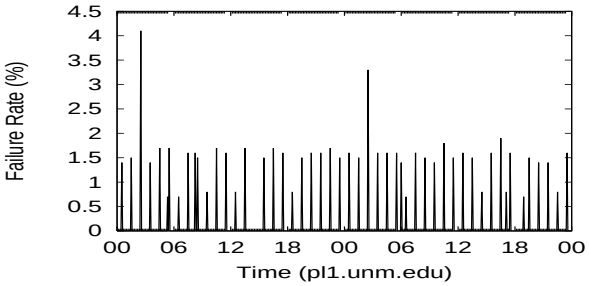


Figure 7: This site shows failures induced by periodic activity. In addition to the hourly failure spike, a larger failure spike is seen once per day.

To test this theory of nameserver overload, we subjected BIND, the most popular nameserver, to bursty traffic. On an otherwise unloaded box (Compaq au600, Linux 2.4.9, 1 GB memory), we ran BIND 9.2.3 and an application-level UDP ping that simulates BIND. Each request contains the same name query for a local domain name with a different query ID. Our UDP ping responds to it by sending a fixed response with the same size as BIND's. We send a burst of N requests from a client machine and wait 10 seconds to gather responses. Figure 6 shows the difference in responses between BIND 9.2.3 and our UDP ping. With the default receive buffer size of 64KB, BIND starts dropping requests at bursts of 200 reqs/sec, and the capacity linearly grows with the size of the receive buffer. Our UDP ping using the default buffer loses some requests due to temporary overflow, but the graph does not flatten because responses consume minimal CPU cycles. These experiments confirm that high-rate bursty traffic can cause server overload, aggravating the buffer overflow problem.

Resource competition – Some sites show periodic failures, similar to what is seen in Figure 7. These tend to have spikes every hour or every few hours, and suggests some heavy process is being launched from cron. BIND is particularly susceptible to memory pressure, since its memory cache is only periodically flushed. Any jobs that use large amounts of memory can evict BIND's pages, causing BIND to page fault when accessing the data. The faults can delay the server, causing the UDP buffer to fill.

In talking with system administrators, we find that even sites with good DNS service often run multiple services (some cron-initiated) on the same machine. Since DNS is regarded as a low-CPU service, other services are run on the same hardware to avoid underutilization. It seems quite common that when these other services have bursty resource behavior, the nameserver is affected.

Maintenance problems – Another common source of failure is maintenance problems which lead to service interruption, as shown in Figure 8. Here, the DNS lookup shows a 100% failure rate for 13 hours. Both name-

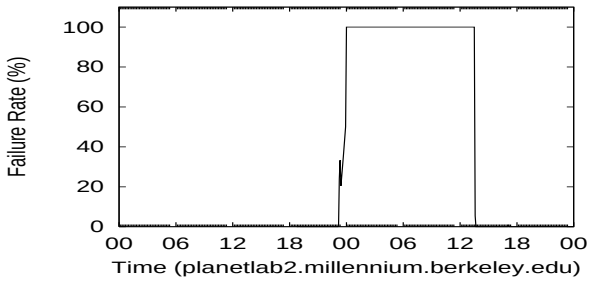


Figure 8: This site's nameservers were shut down before the nodes had been updated with the new nameserver information. The result was a 13-hour complete failure of all name lookups, until the information was manually updated.

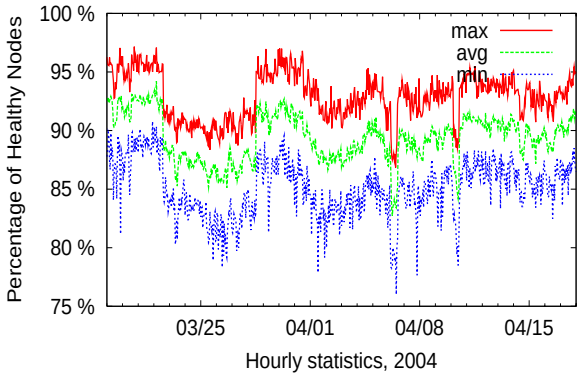


Figure 9: Hourly % of nodes with working nameservers

servers for this site stopped working causing DNS to be completely unavailable, instead of just slow. DNS service was restored only after manual intervention. Another common case, complete failure of the primary nameserver, generates a similar pattern, with all responses being retried after five seconds and sent to the secondary nameserver.

3 Design

In this section, we discuss the design of CoDNS, a name lookup system that provides faster and more reliable DNS service while minimizing extra overhead. We also discuss the observations that shape this approach. Using trace-driven workloads, we calculate the overheads and benefits of various design choices in the system.

One important goal shapes our design: our system should be incrementally deployable, not only by DNS administrators, but also by individual users. The main reason for this decision is that it bypasses the bureaucratic processes involved with replacing existing DNS infrastructure. Given the difficulty we have in even getting information about local DNS nameservers, the chances of convincing system administrators to send their live traffic to an experimental name lookup service seems low. Providing a migration path that coexists with existing infrastructure allows people the opportunity to grow comfortable with the service over time.

Another implication of this strategy is that we should aim for minimal resource commitments. In particular, we should leverage the existing infrastructure devoted to making DNS performance generally quite good. Client-side nameservers achieve high cache hit rates by devoting memory to name caching, and if we can take advantage of the existing infrastructure, it lessens the cost of deployment. While current client-side infrastructure, including nameservers, is not perfect, it provides good performance most of the time, and it can provide a useful starting point. Low resource usage also reduces the chances for failure due to resource contention.

Our usage model is cooperative, operating similarly to insurance – nodes join a pool that shares resources in times of need. If a node's local lookup performance is acceptable, it proceeds as usual, but may have to provide service to nodes that are having problems. When its local performance degrades, it can ask other nodes to help it. The benefit of joining is the ability to get help when needed, even if there is some overhead at other times.

3.1 Cross-site Correlation of DNS Failures

The "insurance" model depends on failure being relatively uncorrelated – the system must always have a sufficient pool of working participants to help those having trouble. If failure across sites is correlated, this assumption is violated, and a cooperative lookup scheme is less feasible. To test our assumption, we study the correlation of DNS lookup failures across PlanetLab. At every minute, we record how many nodes have "healthy" DNS performance. We define healthy as showing no failures for one minute for the local domain name lookup test. Using the per-minute data for March 2004, we show the minimum, average and maximum number of nodes available per hour. The percentage of healthy nodes (as a fraction of live nodes) is shown in Figure 9.

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 are configured to use the same set of nameservers, especially those colocated at Internet2 backbone 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 high, generally above 90%. This observation provides the key insight for CoDNS – with enough healthy nameservers, we can mask locally-observed delays via cooperation.

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 2: Comparison of nameserver software used by PlanetLab, packetfactory survey and the TLD survey

To ensure that these failures are not tied to any specific nameserver software, we survey the software running on the local nameservers used by the PlanetLab nodes (135 unique nameservers) with "chaos" class queries [14]. We find that they are mostly running a variety of BIND versions. We observe 11 different BIND 9 version strings, 13 different BIND 8 version strings and a number of humorous strings (included in "other") apparently set by the nameserver administrators. These measurements, shown in Table 2, are in line with two recent nameserver surveys conducted by Brad Knowles in 2002 [11] and by packetfactory in 2003 [19]. From this, we conclude that the failures are not likely to be specific to PlanetLab's choices of nameserver software.

3.2 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 applies a CDN approach to DNS – spreading the load among peers improves the size and performance of the "global cache". Many of the considerations 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. The proximity of a peer server is important in that DNS response time can be affected by its peer to peer latency. Since the DNS requests and responses are not large, we are more interested in picking nearby peers with low round-trip latency instead of nodes with particularly high bandwidth. We have observed coast-to-coast round-trip ping times of 80ms in CoDeeN, with regional times in the 20ms range. Both of these ranges are much lower than the DNS timeout value of five seconds, so, in theory, any node would be an acceptable peer. In practice, choosing closer peers will reduce the difference between cache hit times and remote peer times, making CoDNS failure masking more transparent. For request locality, we would like to increase the chances of remote queries being cache hits in the remote nameservers. Using any scheme that consistently partitions this workload will help reduce cache pollution, and increase the likelihood of cache hits.

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 44,486 unique hostnames from one day's HTTP traffic on CoDeeN and simulated various policies and their effects. We replayed DNS lookups of those names at 77 PlanetLab nodes with different nameservers, 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. During this time, we also use application-level heartbeat measurements between all pairs of nodes to determine their round-trip latencies. Since all of the nodes are doing DNS lookups at about the same time, by adding the response time 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.

An interesting question is how many simultaneous lookups are needed to achieve a given average response time and to reduce the total time spent on slow lookups (defined as taking more than 1 second). As shown in the previous section, it is desirable to reduce the number of slow responses to reduce the total lookup time. Figures 10 and 11 show two graphs answering this question. The lookup scheme here is to contact the local nameserver first for a name lookup, wait for a timeout and issue x-1 simultaneous lookups using x-1 randomly-selected peer nodes. Figures 10 shows that even if we use only one extra lookup, we can reduce the average response time by more than half. Also, beyond about five peers, adding more simultaneous lookups produces diminishing returns. Different initial timeout values do not produce much difference in response times, because the benefit largely stems from reducing the number of slow lookups. The slow response portion graph proves this phenomenon, showing similar reduction in the slow response percentage at any initial timeout less than 700ms.

We must also consider the extra overhead of the simultaneous lookups, since shorter initial timeouts and more simultaneous lookups causes more DNS traffic at all peers. Figure 12 shows the overhead in terms of extra lookups needed for various scenarios. Most curves start to flatten at a 500ms initial timeout, providing only diminishing returns for larger timeouts. Worth noting is that even with one peer and a 200ms initial timeout, we can still cut the average response time by more than half, with only 38% extra DNS lookups.

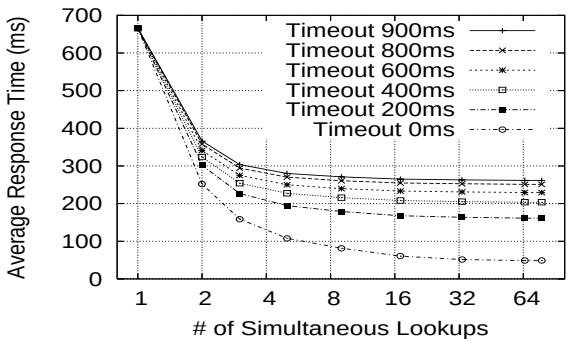


Figure 10: Average Response Time

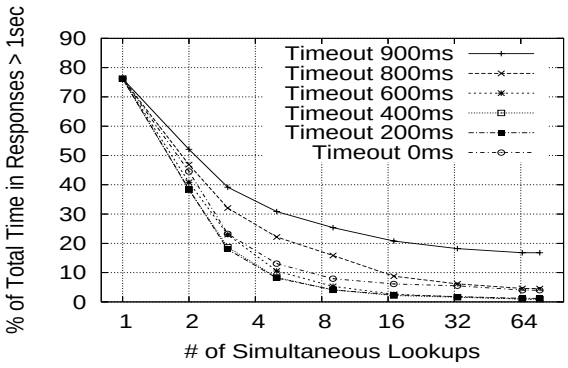


Figure 11: Slow Response Time Portion

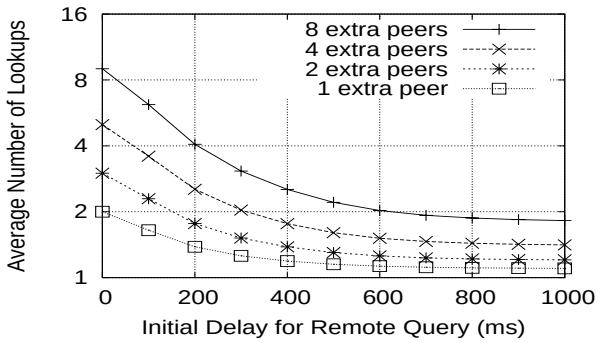


Figure 12: Extra DNS Lookups

These results are very encouraging, demonstrating that CoDNS can be effective even at very small scale – even a single extra site provides significant benefits, and it achieves most of its benefits with less than 10 sites. The reasons for this scale being important is twofold: only small commitments are required to try a CoDNS deployment, and DNS's limitations with respect to trust and verification (discussed in the next section) are unlikely to be an issue at these scales.

3.3 Trust, Verification, and Implications

Some aspects of DNS and its design invariably impact our approach, and the most important is trust and verification. The central issue is whether it is possible for a requestor to determine that its peer has correctly resolved the request, and that the result provided is actually a valid IP address for the given name. This issue arises if peers can be compromised or are otherwise failing.

Unfortunately, we believe that a general solution to this problem is not possible with the current DNS, though certain fault models are more amenable to checking than others. For example, if the security model assumes that at most one peer can be compromised, it may be possible to always send remote requests to at least three peers. When these nodes respond, if two results agree, then the answer must be correct. However, DNS does not mandate that any of these results have to agree, making the general case of verification impossible.

Many server-side DNS deployments use techniques to improve performance, reliability, or balance load and locality. For example, round-robin DNS can return results from a list of IP addresses in order to distribute load across a set of servers. Geography-based redirection can be used to reduce round-trip times between clients and servers by having DNS lookups resolve to closer servers. Finally, DNS-based content distribution networks will often incorporate load balancing and locality considerations when resolving their DNS names. In these cases, multiple lookups may produce different results, and lookups from different locations may receive results from different pools of IP addresses.

While it would be possible to imagine extending DNS such that each name is associated with a public key, and each IP address result is signed with this key, such a change would be significant. DNSSEC [6] attempts smaller-scale change, mainly to prevent DNS spoofing, but has been in some form of development for nearly a decade, and still has not seen wide-scale adoption.

Given the current impossibility of verifying all lookups, we rely on trusting peers in order to sidestep the problems mentioned. This approach is already used in various schemes. Name owners often use each other as their secondary servers, sometimes at large scale. For example, princeton.edu's DNS servers act as the secondary servers for 60 non-Princeton domains. BIND supports zone transfers, where all DNS information can be downloaded from another node, specifically for this kind of scenario. Similarly, large-scale distributed systems running at hosting centers already have a trust relationship in place with their hosting facility.

4 Implementation

We have built a prototype of CoDNS and have been running it on all nodes on PlanetLab for roughly eight months. During that time, we have been directing the CoDeeN CDN [23] to use CoDNS for the name lookup.

CoDNS consists of a stand-alone daemon running on each node, accessible via UDP for remote queries, and via 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 process receives name lookup requests from local clients and remote peers, and passes them to one of its idle slaves. A slave process resolves those names by calling `gethostbyname()` and sends the result back to the master. Then, the master returns the final result to either a local client or a remote peer depending on where it originated. Queries resolving the same hostname are coalesced into one query and answered together when resolved. Preference for idle slaves is given to locally-originated requests over remote queries to ensure good performance for local users.

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 nor 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 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 nameservers and peer responses. In general, 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, if the past 32 name lookups are all resolved locally without using any remote queries, then the initial delay is set to 200ms by default. We choose 200ms because the median response time on a well-functioning node is less than 100ms [9], so 200ms delay should respond fast during instability, 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 results show 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, to avoid swamping fast remote servers.

4.2 Proximity, Locality and Availability

Each CoDNS node gathers and manages a set of neighbor nodes within a reasonable latency boundary. When a CoDNS instance starts, it sends a heartbeat to each node in the preconfigured CoDNS node list every second. The response contains the round trip time (RTT) and the average response time of the local nameserver at the peer node, reflecting the proximity and the availability of the peer node's nameserver. The top 10 nodes with different nameservers are picked as neighbors by comparing the sum with all nodes. Liveness of the chosen neighbors is periodically checked to see if the service is still available. One heartbeat is sent each second, so we guarantee the availability in 10 second granularity. Dead nodes are replaced with the next best node in the list.

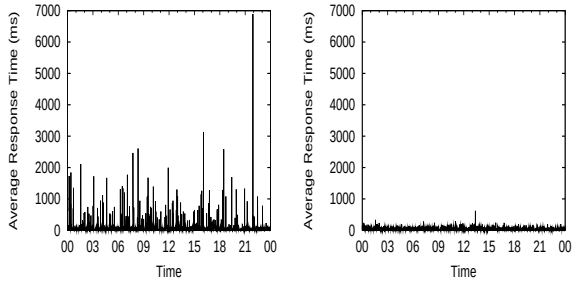
Among these neighbor nodes, one peer is chosen for each remote name lookup using the Highest Random Weight (HRW) hashing scheme [22]. HRW consists of hashing the node name with the lookup name, and then choosing the node name with the smallest resulting hash value. Because HRW consistently picks the same node for the same domain name, this process enhances request locality for remote queries. Another desirable property of this approach is that some request locality is preserved as long as neighbor sets have some overlap. Full overlap is not required.

The number of neighbors is configurable according to the distribution of nodes. In the future, we will make CoDNS dynamically find the peer nodes not depending on the preconfigured set of nodes. One possible solution is to make each CoDNS node advertise its neighbor set and have a few well known nodes. Then, a new CoDNS node with no information about available CoDNS peer nodes can ask the well known nodes for their peer nodes and recursively gather the nodes by asking each neighbor until it finds a reasonable pool of CoDNS nodes.

Note that our neighbor discovery mechanisms are essentially advisory in nature – once the node has enough peers, it only needs to poll other nodes in order to have a reasonable set of candidates in case one of its existing peers becomes unavailable. In the event that some sites have enough peers to make this polling a scalability issue, each node can choose to poll a nearby subset of all possible peers to reduce the background traffic.

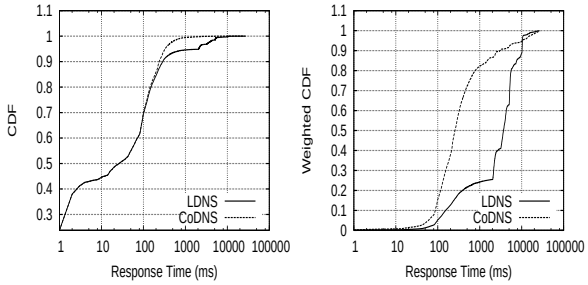
4.3 Policy & Tunability

In the future, we expect CoDNS node policy will become an interesting research area, given the tradeoffs between overhead and latency. We have made choices for initial delay and retry behavior for our environment, and we believe that these choices are generally reasonable. However, some systems may choose to tune CoDNS to have



(a) Local DNS (b) CoDNS

Figure 13: Minute-level Average Response Time for One Day on planetlab1.cs.cornell.edu



(a) Response Time CDF (b) Total Time CDF

Figure 14: CDF and Weighted CDF for One Week on planetlab1.cs.cornell.edu, LDNS = local DNS

much lower overhead, at the cost of some latency benefit. In particular, systems that want to use it only to avoid situations where all local nameservers have failed could use an initial delay threshold of several seconds. In this case, if the local nameserver repeatedly fails to resolve requests in multiple seconds, the initial delay will drop to zero and all lookups will be handled remotely for the duration of the outage.

Sites may also choose to limit CoDNS overhead to a specific level, which would turn parameter choices into an optimization problem. For example, it may be reasonable to ask questions of the form "what is the best latency achievable with a maximum remote lookup rate of 10%?" Our trace-driven simulations give some insight into how to make these choices, but it may be desirable to have an online system automatically adjust parameter values continuously in order to meet these constraints. We are investigating policies for such scenarios.

4.4 Bootstrapping

CoDNS has a bootstrapping problem, since it must resolve peer names in order to operate. In particular, when the local DNS service is slow, resolving all peer names before starting will increase CoDNS's start time. So, CoDNS begins operation immediately, and starts resolving peer names in the background, which greatly reduces its start time. The background resolver uses CoDNS itself, so as soon as a single working peer's name is resolved, it can then quickly help resolve all other peer names. With this bootstrapping approach, CoDNS starts virtually instantaneously, and can resolve all 350 peer names in less than 10 seconds, even for slow local DNS. A special case of this problem is starting when local DNS is completely unavailable. In this case, CoDNS would be unable to resolve even any peer names, and could not send remote queries. CoDNS periodically stores all peer information on disk, and uses that information at startup. This file is shipped with CoDNS, allowing operation even on nodes that have no DNS support at all.

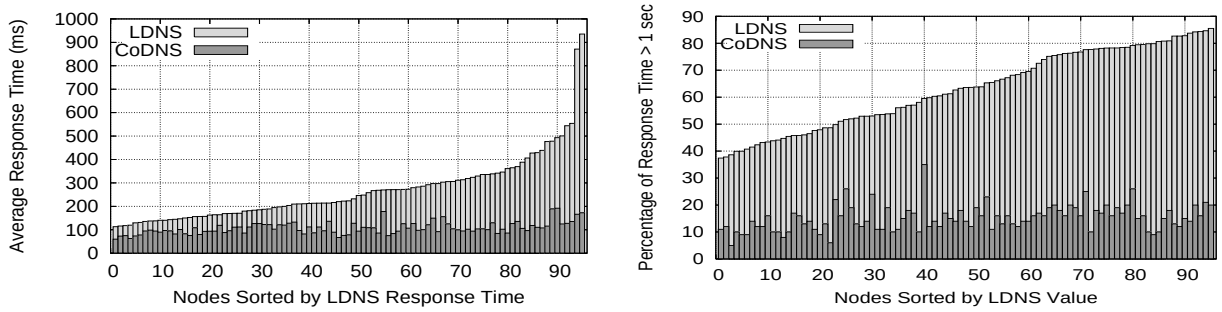
5 Evaluation / Live Traffic

To gauge the effectiveness of CoDNS, we compare its behavior with local DNS on CoDeeN's live traffic using a variety of metrics. CoDeeN receives 5-7 million requests daily from a world-wide client population of 7-12K users. These users have explicitly specified CoDeeN proxies in their browser, so all of their Web traffic is directed through CoDeeN. The CoDeeN proxies maintain their own DNS caches, so only uncached DNS names cause lookups. To eliminate the possible 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.

CoDNS effectively removes the spikes in the response time, and provides more reliable and predictable service for name lookups. Figure 13 compares per-minute average response times of local DNS and CoDNS for CoDeeN's live traffic for one day on one PlanetLab node. While local DNS shows response time spikes of 7 seconds, CoDNS never exceeds 0.6 seconds. The benefit stems from redirecting slow lookups to working peers.

The greater benefit of CoDNS lies in reducing the frequency of slow responses. Figure 14 shows a CDF and a weighted CDF for name lookup response distribution for the same node for one week. The CDF graph shows that the response distribution in both schemes is almost similar until the 90th percentile, but CoDNS reduces the lookups taking more than 1000ms from 5.5% to 0.6%. This reduction gives much benefit in total lookup time in the weighted CDF. It shows CoDNS now spends 18% of total time in lookups taking more than 1000ms, while local DNS still spends 75% of the total time on them.

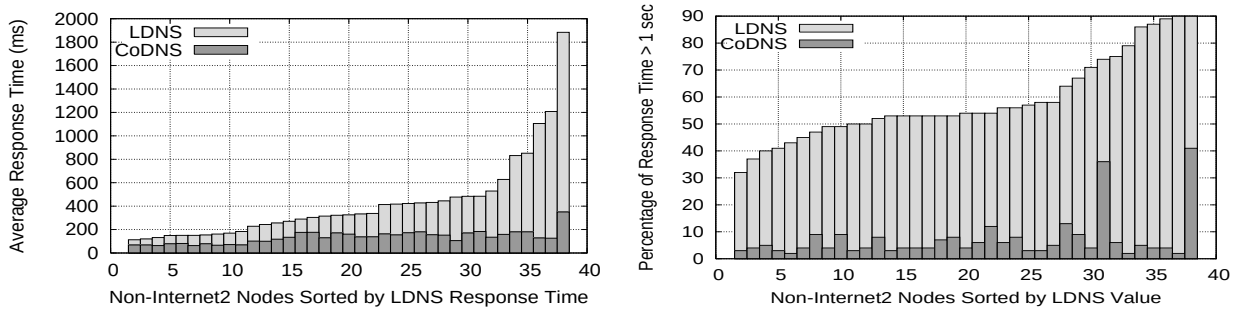
This improvement is widespread – Figure 15(a) shows the statistics of 95 CoDeeN nodes for the same period. The average number of total lookups per node is 22,208, ranging from 12,119 to 131,466 per node. The average response time in CoDNS is 60-221ms, while that of local DNS is 113-935ms. In all cases, CoDNS's response is



(a) Average Response Time

(b) Slow Response Time portion

Figure 15: Live Traffic for One Week on the CoDeeN Nodes, LDNS = local DNS



(a) Average Response Time

(b) Slow Response Time Portion

Figure 16: Non-Internet-2 Nodes, LDNS = local DNS

faster, ranging from a factor of 1.37 to 5.42. Figure 15(b) shows the percentage of slow responses in the total response time. CoDNS again reduces the slow response's portion dramatically to less than 20% of the total lookup time in most cases, delivering more predictable response time. In contrast, local DNS spends 37% to 85% of the total time in the slow queries.

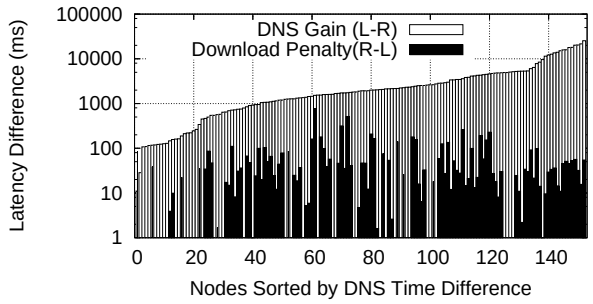
5.1 Non-Internet2 Benefits

Since most CoDeeN sites are hosted at North American universities with Internet2 (I2) connectivity, one may suspect that low-congestion I2 peer links are responsible for our benefits. To address this issue, we pick non-I2 PlanetLab nodes and replay 10,792 unique lookups of hostnames from one day's live traffic on a CoDeeN proxy. Figure 16(a) shows that CoDNS provides similar benefit on 38 non-I2 nodes as well. The average response time in CoDNS ranges from 63ms to 350ms, while local DNS is 113ms to 1884ms, an improvement of factor of 1.64 to 9.52. Figure 16(b) shows that CoDNS greatly reduces the slow response portion as well – CoDNS generally spends less than 10% of the total time in this range, while local DNS still spends 32% to 90%.

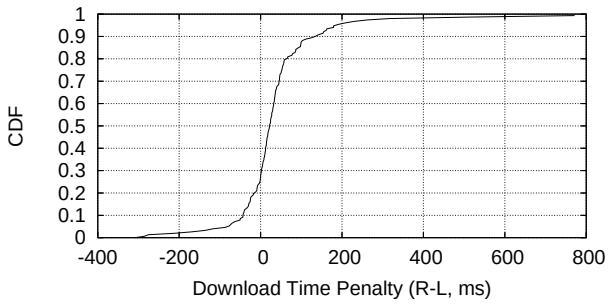
5.2 CDN Effect

CoDNS replaces slow local responses with fast remote responses, which may impact DNS-based CDNs [1] that resolve names based on which DNS nameserver sends the query. CoDNS may return the address of a far replica when it uses a peer's nameserver result. We investigate this issue by testing 14 popular CDN users including Apple, CNN, and the New York Times. We measure the DNS and download time of URLs for the logo image file on those web sites, and compare local DNS and CoDNS when their responses differ.

Since CoDNS is used only when the local DNS is slow or failing, it should come as no surprise that the total time for CDN content is still faster on CoDNS when they differ in returned IP address. The DNS time gain and the downloading time penalty presented in the difference between local and remote response time is shown in Figure 17(a). When local DNS is slow, CoDNS combined with a possibly sub-optimal CDN node is a much better choice, with the gain from faster name lookups dwarfing the small difference in download times when any difference exists. If we isolate the downloading time difference between the DNS-provided CDN node versus the



(a) DNS Lookup Time Gain vs. Downloading Time Penalty



(b) Cumulative Distribution of Downloading Time Difference

Figure 17: CDN Effect for www.apple.com, L = Local Response Time, R = Remote Response Time, DNS gain = Local DNS time - CoDNS time, Download penalty = download time of CoDNS-provided IP - download time of DNS-provided IP, shown in log scale. Negative penalties indicate CoDNS-provided IP is faster, and are not shown in the left graph.

CoDNS-provided CDN node, we get Figure 17(b). Surprisingly, almost a third of the CoDNS-provided nodes are closer than their DNS counterparts, and 83% of them show less than a 100ms difference. This matches the CDN's strategy to avoid notably bad servers instead of choosing the optimal server [8]. Results for other CDN vendors are similar.

5.3 Reliability and Availability

CoDNS dramatically improves DNS reliability, measured by the local nameserver availability. To quantify this effect, we measured the availability of name lookups for one month across all CoDeeN nodes, with and without CoDNS. We assume that a nameserver is available unless it fails to answer requests. If it fails, we consider the periods of time when no requests were answered as its unavailability. Each period is capped at a maximum of five seconds, and the total unavailability is measured as the sum of the unavailable periods. This data, shown in Figure 18, is presented using the reliability metric of "9's" of availability. Regular DNS achieves 99% availability on about 60% of the nodes, which means roughly 14 minutes per day of no service. In contrast, CoDNS is able to achieve over 99.9% availability on over 70% of nodes, reducing downtimes to less than 90 seconds per day. On some nodes, the availability approaches 99.99%, or roughly 9 seconds of unavailability per day. CoDNS provides roughly an additional '9' of availability, without any modifications to the local DNS infrastructure.

5.4 Overhead Analysis

To analyze CoDNS's overhead, we examine the remote query traffic generated by the CoDeeN live activity. For this workload, CoDNS issued 11% to 85% of the total lookups as remote queries, as shown in Figure 19. The variation reflects the health of the local nameserver, and less stable nameservers require more remote queries from CoDNS. Of the six nodes that had more than 50%

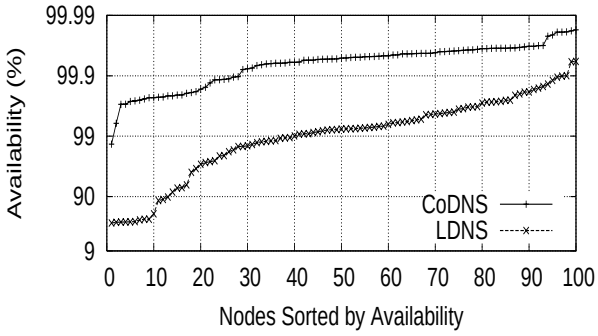


Figure 18: Availability of CoDNS and local DNS (LDNS)

remote queries, all experienced complete nameserver failure at some point, during which remote queries increased to over 100% of the local requests. These periods skew the average overhead.

We believe that the additional burden on nodes with working DNS is tolerable, due to the combination of our locality-conscious redirection and already high local nameserver hit rates. Using our observed median overhead of 25% and a local hit rate of 80% - 87% [9], the local DNS will incur only 3.25 - 5.00% extra outbound queries. Since remote queries are redirected only to lightly loaded nodes, we believe the extra lookups will be tolerable on the peer node's local nameserver.

We also note that many remote queries are not answered, with Figure 19 showing this number varies from 6% to 31%. These can be due to WAN packet losses, unresolvable names, and remote node rate-limiting. CoDNS nodes drop remote requests if too many are queued, which prevents a possible denial of service attack. CoDNS peers never reply if the request is unresolvable, since their own local DNS may be failing, and some other peer may be able to resolve the name.

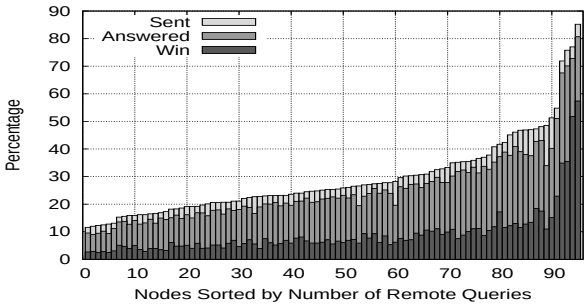


Figure 19: Analysis for Remote Lookups

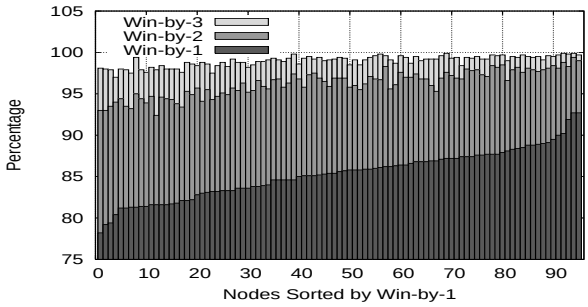


Figure 20: Win-by-N for Remote Lookups

The queries in which CoDNS "wins", by beating the local DNS, constitute 2% to 57% of the total requests. On average, 9% of the original queries were answered by the remote responses, removing 47% of the slow response portion in the total lookup time shown in the Figure 15(b). Of the winning remote responses, more than 80% were answered by contacting the first peer, specified as "win-by-1" in Figure 20. Of all winning responses, 95% are resolved by the first or second peer, and only a small number require contacting three or more peers. This information can be used to further reduce CoDNS's overhead by reducing the number of peers contacted – if it has not been resolved within the first three peers, then further attempts are unlikely to resolve it, and no more peers should be contacted. We may explore this optimization in the future, but our current overheads are low enough that we have no pressing need to reduce them.

In terms of extra network traffic generated for remote queries, each query contains about 300 bytes of a request and a response. On average, each CoDNS on a CoDeeN node handles 414 to 10,287 requests per day during the week period, amounting to 243KB to 6027KB. CoDNS also consumes heartbeat messages to monitor the peers each second, which contains 32 bytes of data. In sum, each CoDNS on a CoDeeN node consumes on average 7.5 MB of extra network traffic per day, consuming only 0.2% of total CoDeeN traffic in relative terms.

5.5 Application Benefits

By using CoDNS, CoDeeN obtains other benefits in capacity and availability, and these may apply to other applications as well. The capacity improvements come from CoDeeN being able to use nodes that are virtually unusable due to local DNS problems. At any given time, roughly 10 of the 100 PlanetLab nodes that run CoDeeN are experiencing significant DNS problems, ranging from high failure rates to complete failure of the primary (and even secondary) nameservers. CoDeeN nodes normally report their local status to each other, and before CoDNS, these nodes would tell other nodes to avoid them due to the DNS problems. With CoDNS, these nodes can still be used, providing an additional 10% extra capacity.

The availability improvements come from reducing startup time, which can be dramatic on some nodes. CoDeeN software upgrades are not announced downtimes, because on nodes with working local DNS, CoDeeN normally starts in 10-15 seconds. This startup process is fast enough that few people notice a service disruption. Part of this time is spent in resolving the names of all CoDeeN peers, and when the primary DNS server is failing, each lookup normally requires over five seconds. For 120 peers, this raises the startup time to over 10 minutes, which is a noticeable service outage. If CoDNS is already running on the node, startup times are virtually unaffected by local failure, since CoDNS is already sending all queries to remote servers in this environment. If CoDNS starts concurrently with CoDeeN, the startup time for CoDeeN is roughly 20 seconds.

6 Other Approaches

6.1 Private Nameservers

Since local nameservers exhibit overload, one may be tempted to run a private nameserver on each machine, and have it contact the global DNS hierarchy directly. This approach is more feasible as a backup mechanism than as a primary nameserver for several reasons. Using shared nameservers reduces maintenance issues, and the shared cache can be larger than individual caches. Not only does cache effectiveness increase due to capacity, but the compulsory misses will also be reduced from the sharing. With increased cache misses, the global DNS failure rate becomes more of an issue, so using private nameservers may reduce performance and reliability.

As a backup mechanism, this approach is possible, but has the drawbacks common to any infrequently-used system. If it is not being exercised regularly, failure is less likely to be noticed, and the system may be unavailable when it is needed most. It also consumes resources when not in use, so other tasks on the same machine will be impacted, if only slightly.

6.2 Secondary Nameservers

Since most sites have two or more local nameservers, another approach would be to modify the resolver libraries to be more aggressive about using multiple nameservers. Possible options include sending requests to all nameservers simultaneously, being more aggressive about timeouts and using the secondary nameserver, or choosing whichever one has better response times.

While we believe that some of these approaches have some merit, we also note that they cannot address all of the failure modes that CoDNS can handle. In particular, we have often seen all nameservers at a site fail, in which case CoDNS is still able to answer queries via the remote nameservers. Correlated failure of local nameservers renders these approaches useless, while correlated failure among groups of remote servers is less likely.

Overly aggressive strategies are likely to backfire in the case of local nameservers, since we have seen that overload causes local nameserver failure. Increasing the request rate to a failing server is not likely to improve performance. Load balancing among local nameservers is more plausible, but still requires modifications to all clients and programs. Given the cost of changing infrastructure, it is perhaps appealing to adopt a technique like CoDNS that covers a broader range of failures.

Finally, upgrade cost and effort are real issues we have heard from many system administrators – secondary nameservers tend to be machines that are a generation behind the primary nameservers, based on the expectation of lower load. Increasing the request rate to the secondary nameserver will require upgrading that machine, whereas CoDNS works with existing infrastructure.

6.3 TCP Queries

Another possible solution is to use TCP instead of UDP as a way of communicating with local nameservers. If the failure is caused by packet losses in the LAN or silent packet drops caused by UDP buffer overflow, TCP can improve the situation by reliable data delivery. In addition, the flow control mechanism inherent in TCP can ask the name lookup clients to slow down when the nameserver is overloaded.

Although the DNS RFC [14] allows the use of TCP in addition to UDP, in practice, TCP is used only when handling AXFR queries for the zone transfer or when the requested record set is bigger than 512 bytes. The reason why TCP is not favored in name lookups is mainly because of the additional overhead. If a TCP connection is needed for every query, it would end up handling nine packets instead of two : three to establish the connection, two for the request/response, and four to tear down the connection. A persistent TCP connection might remove the per-query connection overhead, but it also needs to consume one or two extra network packets for ACKs.

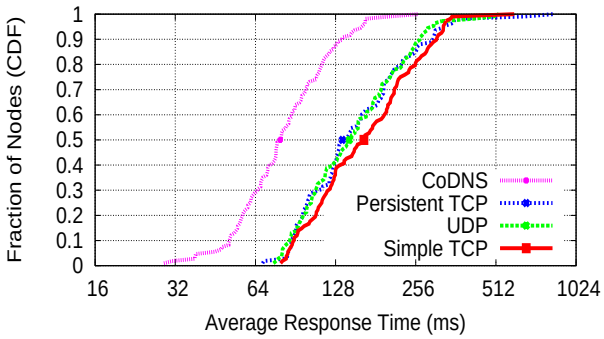


Figure 21: Comparison of UDP, TCP, and CoDNS latencies

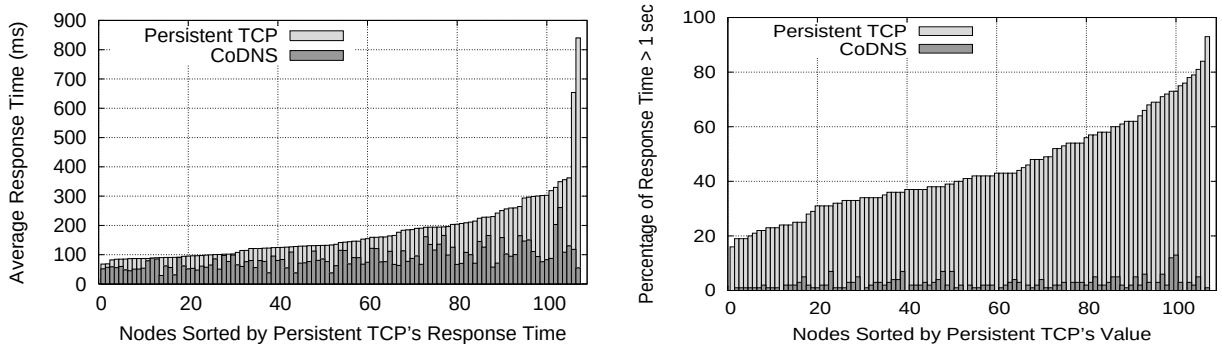
Also, there is another issue of reclaiming the idle connections, since they consume system resources and can degrade performance. The DNS RFC [14] specifies two minutes as a cutoff but in practice most servers disconnect the idle connection within 30 seconds.

To compare the performance between UDP and TCP, we replay 10,792 unique hostnames obtained from one day's live traffic of a CoDeeN proxy at 107 PlanetLab nodes. Carrying out a completely fair comparison is difficult, since we cannot issue the same query for all of them at the same time. Instead, to give a relatively fair comparison, we run the test for CoDNS first, and subsequently run other parts, making all but CoDNS get the benefit of cached responses from the local nameserver after having been fetched by CoDNS. Figure 21 shows the CDF of the average response time for all approaches. Persistent TCP and UDP have comparable performance, while simple TCP is noticeably worse. The CoDNS latencies, included for reference, are better than all three.

The replay scenario described above should be favorable to TCP, but even in this conservative configuration, CoDNS still wins. Figure 22(a) shows that all nodes report that CoDNS is 10% to 500% faster than TCP, confirming CoDNS is a more attractive option than TCP. The large difference is in the slow-response portion, where CoDNS wins the most and where TCP-based lookups cannot help. Figure 22(b) shows that a considerable amount of time is still spent on the long delayed queries in TCP-based lookups. CoDNS reduces this time by 16% to 92% when compared to the TCP-based measurement. Though TCP ensures that the client's request reaches the nameserver, if the nameserver is overloaded, it may have trouble contacting the DNS hierarchy for cache misses.

7 Related Work

Traditional DNS-related research has focused on the problems in the server-side DNS infrastructure. As a seminal study in DNS measurement, Danzig *et al.* found that a large number of network packets were being wasted due to DNS traffic, blaming nameserver software bugs and misconfigurations as major culprits [5].



(a) Average Response Time

(b) Slow Response Time Portion

Figure 22: CoDNS vs. TCP

Since then, bugs in the resolvers and nameservers have been reduced [12], but recent measurements show that there is still much room for improvement. In 2000, Wills *et al.* [24] and Huitema *et al.* [7] reported 29% of DNS lookups take over 2 seconds, and Cohen *et al.* [3] reported 10% of lookups exceed more than 3 seconds. Jung *et al.* also present data indicating 23% of all server-side lookups receive no results, indicating the problems of improper configurations and incorrect nameservers still persist [9]. They measure the client-side performance in terms of response time and caching hit ratio as well. However, that work does not trace the origins of name lookup delays from the client-side, concentrating only on the wide-area DNS traffic. Given the fact that local nameserver cache hit ratios are 80% - 87% [9, 24], even a small problem in the local nameserver and its environment can skew the latency of a large number of lookups. Our study addresses this problem. Liston *et al.* indirectly provide some evidence of local nameserver problems by attributing the major sources of response time delay to end nameservers rather than the root/gTLD servers [13].

The research community has recently renewed its focus on improving server-side infrastructure. Cox *et al.* investigate the possibility of transforming DNS into a peer-to-peer system [4] using a distributed hash table [21]. 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 on the root servers are removed so that the load is distributed over the entities joining the system.

In CoDoNS, Ramasubramanian *et al.* improve the poor latency performance of this approach by using proactive replication of DNS records [18]. They exploit the Zipf-like distribution of the domain names in web browsing [2] to reduce the replicating overhead while providing O(1) proximity [17]. Our approaches differ in several important aspects – we attempt to reduce overlapping information in caches, in order to maximize the overall aggregate cache size, while they use replication to reduce latency. Our desire for a small process footprint stems from our observation that memory pressure is one of the causes of current failures in client-side infrastructure. While their system appears not to be deployed in production, they perform an evaluation using a DNS trace with a Zipf factor above 0.9 [18]. In comparison, our evaluation of CoDNS uses the live traffic generated by CoDeeN *after* its proxies have used their local DNS caches, so the request stream seen by CoDNS has a Zipf factor of 0.50-0.55, which is a more difficult workload. We intend to compare the live performance of CoDNS versus CoDoNS when the latter system enters production and is made available to the public. In any case, since CoDNS does not depend on the specifics of the name lookup system, we expect that it can interoperate with CoDoNS if the latter provides better performance than the existing nameservers at PlanetLab sites. One issue that will have to be addressed by any proposed DNS replacement system is the use of intelligent nameservers that dynamically determine which IP address to return for a given name. These nameservers are used in CDNs and geographic load balancers, and can not be replaced with purely static lookups, such as those performed in CoDoNS. Since CoDNS does not replace existing DNS infrastructure, we can interoperate with these intelligent nameservers without any problem.

Kangasharju *et al.* pursue a similar approach to reducing the DNS lookup latency by more aggressively replicating DNS information [10]. Inspired by the fact the entire DNS record database fits into the size of a typical hard disk and with the recent emergence of terrestrial multicast and satellite broadcast systems, this scheme reduces the need to query the distant nameservers by keep-

ing the DNS information up to date by efficient worldwide replication.

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 benefits in improving the infrastructure "from above" will complement our "bottom up" approach. One advantage of our system is that misconfigurations can be masked without name server outage, allowing administrators more time to investigate the problem.

8 Conclusion

We have shown that client-side instability in DNS name lookups is widespread and relatively common. The failure cases degrade average lookup time and increase the "tail" of response times. We show that these failures appear to be caused by temporary nameserver overload, and are largely uncorrelated across multiple sites. Through analysis of live traffic, we show that a simple peering system reduces response times and improves reliability.

Using these observations, we develop a lightweight name lookup service, CoDNS, that uses peers at remote sites to provide cooperative lookups during failures. CoDNS operates in conjunction with local DNS nameservers, allowing incrementally deployment without significant resource consumption. We show that this system generates low overhead, cuts average response time by half or more, and increases DNS service availability.

Acknowledgments

This research was supported in part by NSF grant CNS-0335214. We would like to thank Jeffrey Mogul and Hewlett-Packard for donation of the Alpha workstation used for testing. We thank our shepherd, Geoff Voelker, for his guidance and helpful feedback, and we thank our anonymous reviewers for their valuable comments on improving this paper.

References

[1] Akamai. Content Delivery Network. http://www.akamai.com.

[2] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *Proceedings of IEEE INFOCOM*, 1999.

[3] E. Cohen and H. Kaplan. Prefetching the Means for Document Transfer: A New Approach for Reducing Web Latency. In *Proceedings of IEEE INFOCOM*, 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)*, 2002.

[5] P. B. Danzig, K. Obraczka, and A. Kumar. An Analysis of Wide-Area Name Server Traffic: A Study of Internet Domain Name System. In *Proceedings of ACM SIGCOMM*, 1992.

[6] D. Eastlake. Domain Name System Security Extensions. RFC 2535, January 1999.

[7] C. Huitema and S. Weerahandi. Internet Measurements: the Rising Tide and the DNS Snag. In *Proceedings of the 13th ITC Specialist Seminar on Internet Traffic Measuremnet and Modelling*, 2000.

[8] K. Johnson, J. Carr, M. Day, and F. Kaashoek. The Measured Performance of Content Distribution Networks. In *Proceedings of the 5th International Web Caching and Content Delivery Workshop (WCW)*, 2000.

[9] 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.

[10] J. Kangasharju and K. W. Ross. A Replicated Architecture for the Domain Name System. In *Proceedings of IEEE INFOCOM*, 2000.

[11] 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/.

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

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

[14] P. Mockapetris. Domain Names - Implementation and Specification. RFC 1035, November 1987.

[15] P. Mockapetris and K. Dunlap. Development of the Domain Name System. In *Proceedings of ACM SIGCOMM*, 1988.

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

[17] V. Ramasubramanian and E. G. Sirer. Beehive: O(1) Lookup Performance for Power-Law Query Distributions in Peer-to-Peer Overlays. In *1st Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[18] V. Ramasubramanian and E. G. Sirer. The Design and Implementation of a Next Generation Name Service for the Internet. In *Proceedings of ACM SIGCOMM*, 2004.

[19] M. Schiffman. A Samping of the Security Posture of the Internet's DNS Servers. http://www.packetfactory.net/papers/DNS-posture/.

[20] A. Shaikh, R. Tewari, and M. Agrawal. On the Effectiveness of DNS-based Server Selection. In *Proceedings of IEEE INFOCOM*, 2001.

[21] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of ACM SIGCOMM*, San Diego, California, 2001.

[22] D. Thaler and C. Ravishankar. Using Name-based Mappings to Increase Hit Rates. In *IEEE/ACM Transactions on Networking*, volume 6, 1, 1998.

[23] 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.

[24] C. E. Wills and H. Shang. The Contribution of DNS Lookup Costs to Web Object Retrieval. Technical Report WPI-CS-TR-00-12, Worcester Polytechnic Institute (WPI), 2000.